A Double-Layer Neural Network Framework for High-Frequency Forecasting

HAO CHEN, KELI XIAO, JINWEN SUN, and SONG WU, Stony Brook University

Nowadays, machine trading contributes significantly to activities in the equity market, and forecasting market movement under high-frequency scenario has become an important topic in finance. A key challenge in high-frequency market forecasting is modeling the dependency structure among stocks and business sectors, with their high dimensionality and the requirement of computational efficiency. As a group of powerful models, neural networks (NNs) have been used to capture the complex structure in many studies. However, most existing applications of NNs only focus on forecasting with daily or monthly data, not with minute-level data that usually contains more noises. In this article, we propose a novel double-layer neural (DNN) network for high-frequency forecasting, with links specially designed to capture dependence structures among stock returns within different business sectors. Various important technical indicators are also included at different layers of the DNN framework. Our model framework allows update over time to achieve the best goodness-of-fit with the most recent data. The model performance is tested based on 100 stocks with the largest capitals from the S&P 500. The results show that the proposed framework outperforms benchmark methods in terms of the prediction accuracy and returns. Our method will help in financial analysis and trading strategy designs.

CCS Concepts: • **Information systems → Decision support systems**; **Data mining**

Additional Key Words and Phrases: High-frequency forecasting, neural networks, S&P 500

ACM Reference Format:
Hao Chen, Keli Xiao, Jinwen Sun, and Song Wu. 2017. A double-layer neural network framework for high-frequency forecasting. ACM Trans. Manage. Inf. Syst. 7, 4, Article 11 (January 2017), 17 pages.
DOI: http://dx.doi.org/10.1145/3021380

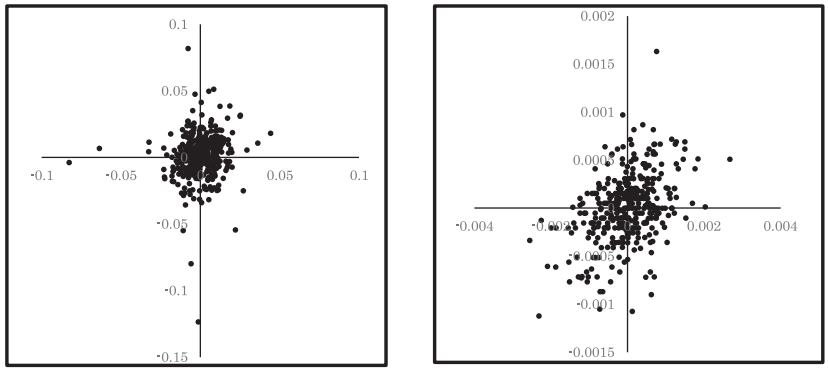
1. INTRODUCTION

With the rapid growth of Internet and computing technologies, high-frequency algorithmic trading has become an indispensable component in the stock market. Practices on the high-frequency data have also provided new directions and challenges for research in finance such as market forecasting. Conceptually, many classical financial models that were originally developed on low-frequency (daily, monthly, and yearly) forecasting problems seem extendable to high-frequency financial data; however, their direct applications are usually less satisfactory for ignoring unique characteristics of the high-frequency data. It is, therefore, important to formalize appropriate frameworks that can better handle the forecasting task under the high-frequency scenario.

One major difference between the low- and high-frequency data is about dependency structures among stocks. The between-stock dependence, which can be measured by

Authors' addresses: H. Chen, J. Sun, and S. Wu (corresponding author), Department of Applied Mathematics & Statistics, Stony Brook University, Stony Brook, NY 11794; emails: shchenhao1201@gmail.com, jinwensun007@gmail.com, Song.Wu@stonybrook.edu; K. Xiao, College of Business, Stony Brook University, Stony Brook, NY 11794; email: keli.xiao@stonybrook.edu.

DOI: http://dx.doi.org/10.1145/3021380



(a) Daily Data (b) 5-min Data

Fig. 1. Correlations between returns of Apple and IBM. (a) Scatter plot of daily returns from January 1, 2013 to June 1, 2013. (b) Scatter plot of returns in 5-min data (June 3, 2013).

their correlation, is usually much stronger in high-frequency data than that in low-frequency data. Figure 1 demonstrates an example of this dependency between the returns of IBM, Inc. and Apple, Inc. It can be seen that the correlation is much stronger in the 5-min data than that in the daily data. These dependencies are expected to bring more information in interpreting and predicting stock movements. A few studies have investigated dependencies of financial data. For example, dynamic relations among stock returns, trading volume, and price volatility have been examined [Chen et al. 2001; Karpoff 1987; Lamoureux and Lastrapes 1990]; however, these studies mainly focus on the existence of dependencies and information gained from them are rarely applied in forecasting problems.

Incorporation of dependency structures into high-frequency forecasting is challenging for two reasons. First, a particular stock typically has several associated stocks, and simultaneous modeling dependency structure among multiple stocks (e.g., more than 100) is usually difficult because of the high dimensionality. Second, the dependency among stocks is a dynamic, but not static, relation. That is, a pair of stocks that are dependent within a period may become less dependent in another period. This issue is particularly prominent for intra-day data given the rapidly changing market environment every day. Therefore, for comprehensiveness, it is preferable to include not a small pre-selected group of stocks, but all stocks in a market (e.g., S&P 500) into the model. This brings another layer of difficulty, which makes the problem of high dimensionality even more severe.

Classic forecasting methods in finance belong roughly to two categories: the time series models [Chen et al. 2007; Hassan et al. 2007; Hsu et al. 2009; Pai and Lin 2005; Tsai and Hsiao 2010], and the machine learning-based approaches [Kara et al. 2011; Patel et al. 2015; Arévalo et al. 2016; Pai and Lin 2005; Hsu et al. 2009]. The time series models are mainly designed to capture within-stock correlations, and rely on strict model assumptions/settings, e.g., specific residual distributions and fixed number of lags, for its optimal performance. For example, Gaussian models were popular for its simplification, but it is rarely satisfied for real-world data and usually has a poor model fitting [Andersen et al. 2001]. Other distributions discussed in recent work include stable Levy distribution [Chiang et al. 2009; Voit 2013], *t*-distribution [Weitzman 2007], and power law distributions [Toda 2012; Toda and Walsh 2014]. For high-frequency data, it is difficult to determine which time series model would fit it the best; otherwise, biases may be generated in forecasting. Additionally, time series

models rarely borrow information from other stocks and are therefore limited in handling the high-dimensionality problem in the high-frequency data.

Machine-learning-based approaches have become more and more popular in financial applications, due to their high prediction power and flexibility in handling diverse data types. Particularly, neural network (NN) based methods are quite advantageous for its model-free properties [see RELATED WORK for more details]. For instance, neural network models do not require specific assumptions on residual distributions for its inferences. Currently, this group of methods is mainly developed on the low-frequency data, but could serve as a good base point to construct more efficient models for high-frequency data given their many good properties.

In this article, we propose a new machine-learning-based framework for market forecasting with high-dimensional high-frequency stock data and focus on designing a novel double-layer NN (DNN) structure that dynamically captures the dependency structure among stock returns for prediction. Particularly, it has the following properties:

—For forecasting returns of a particular stock, returns from other stocks in the market are included as inputs to capitalize on potential dependencies among them.
—Interactions between stocks are summarized at sector level to reduce the number of model parameters and improve the efficiency of implementation. This design is based on the observation that stocks within the same sector, in general, are highly correlated, so this step can remove substantial redundancies in pairwise stock interactions.
—Various technical indicators (TIs) have also been included as model inputs to further improve prediction accuracy. These TIs are incorporated at different layers of the DNN structure for optimal performance.

The rest of this article is organized as follows: In Section 2, we discuss researches that are related to our proposed work. In Section 3, we formulate the problem and propose our neural network framework. In Section 4, we present the data, benchmark algorithms, experiments, and results. Lastly, conclusions and possible extensions are discussed in Section 5.

2. RELATED WORK

Related work falls into two categories. The first category involves forecasting models based on neural network and other machine learning algorithms. The predicting power of neural networks has been noticed by financial industry for quite a while, and many successful applications have been implemented for price forecasting problems. There are extensive literatures on different NN structures, such as fuzzy NN [Ghiassi et al. 2013], partially connected NN [Chang et al. 2012], hybrid of non-linear ICA and NN [Dai et al. 2012], and Legendre NN with random time strength function [Liu and Wang 2012]. Results from these studies suggested that NN based approaches outperform other baselines such as multiple regression and autoregressive integrated moving average (ARIMA) models, although arguments on the model performance still exist. Kara et al. [2011] compared single-layer NN with SVM in stock prediction and concluded that NN was better than SVM. Recently, Patel et al. [2015] compared four machine learning methods in stock prediction: Random forests, SVM, single-layer NN, and naive-Bayes, and suggested that Random Forests was the best performance and naive-Bayes was the worst. While NN models remain a competitive group in financial applications, studies on applying NN models on high-frequency trading is still limited. Particularly, NN structures used in these studies have only single hidden layer, and their test samples are usually small [Arévalo et al. 2016; Choudhry et al. 2012; Balasubramanian et al. 2015], which do not realize the full power of NN models. Arévalo et al. [2016] applied a deep neural nets structure and tested based on a two-month

AAPL stock high-frequency data; Choudhry et al. [2012] tested their neural network model with a one-month high-frequency foreign exchange data. Balasubramanian et al. [2015] implemented a one-layer structure of Neural Network model that was compared with some other machine-learning algorithms in high-frequency forecasting, but the NN model was not carefully developed in these papers.

The second category of related work includes classical econometric models for forecasting. Franses and Van Dijk [1996] studied the performance of the GARCH model and two of its extensions – Quadratic GARCH and GJR models – to forecast weekly stock volatility. Andersen et al. [2007] used bipower variation measures and corresponding nonparametric tests to estimate jumps in high-frequency series of exchange rates, equity index returns and bond yield. Giacometti et al. [2012] compared the performance of AR(1)-ARCH(1) model and the Lee-Carter model with respect to forecasting age-specific mortality in Italy and concluded that the AR(1)-ARCH(1) model with t-student innovations provides a better fit. From the perspective of the market microstructure, Huang and Stoll [1994] investigated the effects of stock characteristics, seasons, and tax regimes on stock returns. They found that some intermediate-term momentum and long-term reversal effects evolve independently, and end-of-year tax-loss selling was identified to be the key link. Kelly and Pruitt [2015] used the three-pass regression filter (3PRF) to forecast a single time series. The techniques were applied to forecast low-frequent macroeconomic aggregates and stock market returns.

Our article differs from the related research by providing new information in three ways. First, we develop a novel DNN structure specially designed for high-frequency data, which may be inspiring for other developments. Second, we proposed a new ensemble NN framework for stable model performance, which may be constructive to other NN applications. Third, our results are very robust since they are based on a large dataset that includes 100 stocks with the largest capitals in the S&P 500.

3. FORECASTING WITH HIGH-FREQUENCY DATA

In this section, we introduce the details of the double-layer neural network framework for stock market forecasting with high-frequency data.

3.1. Problem Statement

Under the high-frequency trading environment, high-quality one-step forecasting is usually of great concern to algorithmic traders, providing significant information to market makers for risk assessment and management. In this article, we aim to forecast the price movement of individual stocks or the market index one step ahead, based solely on their historical price information. Our problem can be mathematically formalized as follows.

Let $S_{t,j}$ denote the closing price of stock $j\,(j = 1, \ldots, J)$ for a 5-minute interval at time $t\,(t = 1, \ldots, T)$, where J is the number of stocks and T is the maximum lag of time; the periodic return of stock j at t can be computed by $X_{t,j} = \frac{S_{t,j} - S_{t-1,j}}{S_{t-1,j}}$. Given the historical price information $S_{i,j}\,(t = 1, \ldots, T; j = 1, \ldots, J)$, our goal is to predict the closing price, $S_{t+1,j}$, or the return $X_{t+1,j}$, for the next 5-minute time interval.

3.2. Overview of the DNN Framework

Different from commonly used neural networks in market forecasting, our DNN structure not only has a deep structure but also has the flexibility in controlling the input features (see Figure 2). Specifically, our design is able to incorporate input variables into the model in a hierarchical way, which dramatically reduces the parameter dimension, and more importantly improves the model performance. The new design also avoids unnecessary information loss for some important correlated features during

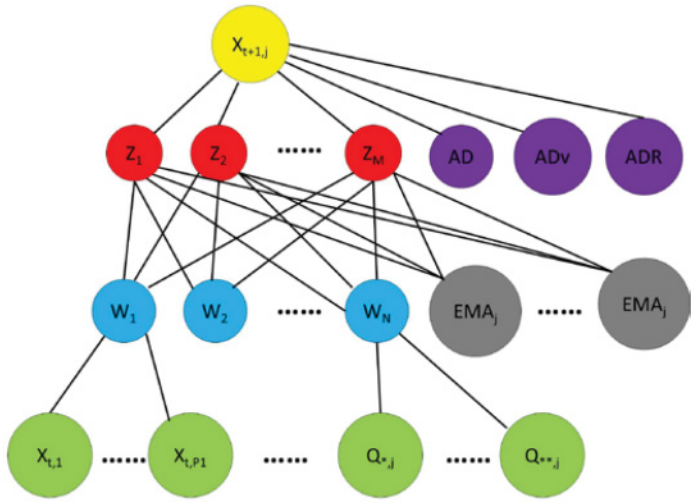


Fig. 2. The Double Layer Neural Network Framework. Inputs in the bottome layers are summarized into hidden variables Ws, which then subsequently interacts with the input EMAs and form the second-layer hidden variables Zs. The Z variables and market indicators serve as the final input for forecasting.

the model training process. Moreover, the new design can efficiently handle the high-dimension data, of which the internal dependence structure is complex and difficult to be interpreted.

As shown in Figure 2, the output of our DNN is $X_{t+1,j}$, i.e., the one-step forecasting on the return for a specific stock j. The bottom layer consists of two sets of information including $\mathcal{X}(t, j) = \{X_{t,j'} : j' \neq j\}$ and $\mathcal{Q}(t', j) = \{Q_{t',j} : t' = t - (\tau - 1), t - (\tau - 2) \ldots, t\}$, where $\mathcal{X}(t, j)$ includes the price information of other stocks in the market at time t, excluding stock j; and the $\mathcal{Q}(t', j)$ includes information of the intra-interval proportions for stock j in r most recent 5-minute intervals. Especially, local connections from the input layer to the first hidden layer (the W layer, Figure 2) are applied. That is, one W hidden neuron gathers information only from a group of linked input neurons. To determine the appropriate number of neurons in the first hidden layer, we assume that the information of $\mathcal{X}(t, j)$ can be categorized into groups according to b predefined business sectors. We will discuss the correlations among different business sectors later. In addition, based on previous studies on market momentum and time series seasonality, we assume the information of τ lags, $\mathcal{Q}(t', j)$, would provide different levels of predicting power. Hence, in total, we set $b + r$ hidden neurons (the Ws) in the first-hidden layer ($N = b + r$). Mathematically, the connections between the Ws and the $\mathcal{X}(t, j)$ can be expressed as follows:

$$W_n = \sigma\left(\sum_{p=s_n}^{e_n} \alpha_{np} * X_p\right), n = 1, \ldots, b \tag{1}$$

and connections between the Ws and the $\mathcal{Q}(t', j)$ can be described as

$$W_n = \sigma\left(\sum_{p=s_n}^{e_n} \alpha_{np} * Q_p\right), n = b + 1, \ldots, N \tag{2}$$

where $\sigma(x) = \frac{1-e^{-x}}{1+e^{-x}}$ is the activation function; s_n and e_n are the starting and ending indices for the n^{th} group of input variables.

In addition to the NWs, another 15 inputs neurons, corresponding to different exponential moving averages for stock j, are added into the first hidden layer to connect with five hidden neurons (the Zs) in the second hidden layer. The values of the second hidden layer, Zs, can be expressed as follows:

$$Z_m = \sigma\left(\sum_{n=1}^{N} \beta_{mn} * W_n + \sum_{n'=1}^{15} \beta_{mn'} * EMA_{n'}\right) \tag{3}$$

Finally, in the second hidden layer, three more input neurons, corresponding to the advanced-decline (A/D) information in S&P 500, are taken into consideration with Zs, which are jointly connected to the output Y. Hence, the final output is expressed as follows:

$$g(Y) = \sum_{m=1}^{5} \theta_{km} * Z_m + \sum_{m'=1}^{3} \theta_{km'} * AD_{m'} \tag{4}$$

where $g(\cdot)$ is the link function. Here, since $Y = X_{t+1,j}$ is continuous, $g$can simply take an identity function, i.e., $g(x) = x$. However, this model structure can be readily extended to binary or multi-categorical outputs, where g can take a soft-max function, such as $g(x) = \frac{e^x}{1+e^x}$.

3.2.1. Input Features. The input variables/features consist of several technical indicators, including Intra-interval proportions (IIP) and exponential moving averages (EMAs), and also some market indicators that gauge the overall upward/downward market trend, such as Advance/Decline (AD), Advance/Decline volume (ADv) and Advance/Decline Ratio (ADR). Initially, we tried to build a regular double NN (RDNN) without the hierarchical structure, i.e., all inputs at the bottom layer and two hidden layers in the middle (Section 4.2.4 for more details), however, the performance of RDNN is not satisfactory (Section 4.4 for more details). This motivates us to differentiate the inputs in the NN design. We built linear regression models on 100 stocks with largest capitals in S&P 500 (same group of stocks in Section 4) and found that on average, AD indicators could explain more variances than EMA, which in turn explain more variances than the Q variables. Therefore we adopted the current DNN design (Figure 2) to reflect the "closeness" of each variable to the outcome, and the new DNN structure seems to work well (Section 4.4 for more details).

Input Layer: The IIPs. This group of features summarizes periodic moving ranges of stocks. For a particular 5-minute interval t, the relationship between its open, high, low, and close prices may provide meaningful information regarding to the price movement, particularly for short-term momentum movements. For example, a close located nearby the low or nearby the high may suggest different market sentiment that is useful in the prediction. Three intra-interval proportions, $Q_{t,j} = (\frac{a_{t,j}}{d_{t,j}}, \frac{b_{t,j}}{d_{t,j}}, \frac{c_{t,j}}{d_{t,j}})$, are used to summarize the information for a time interval t, where $a_{t,j} = High - Open$; $b_{t,j} = Open - Close$; $c_{t,j} = Close - Low$ and $d_{t,j} = High - Low$.

First Hidden Layer: the EMAs. This group of indicators describes the market momentum [Holt 2004]. We selectively include several EMAs as inputs to capture both short-term and long-term momentum. EMAs are usually smoother than the raw prices and could generate more stable performance in the perdition. EMAs are calculated with the following formula and the parameter m controls the decaying rate:

$$EMA_t\left(m, S_{t,j}\right) = \alpha S_{t,j} + (1-\alpha) EMA_{t-1}\left(m, S_{t,j}\right) \tag{5}$$

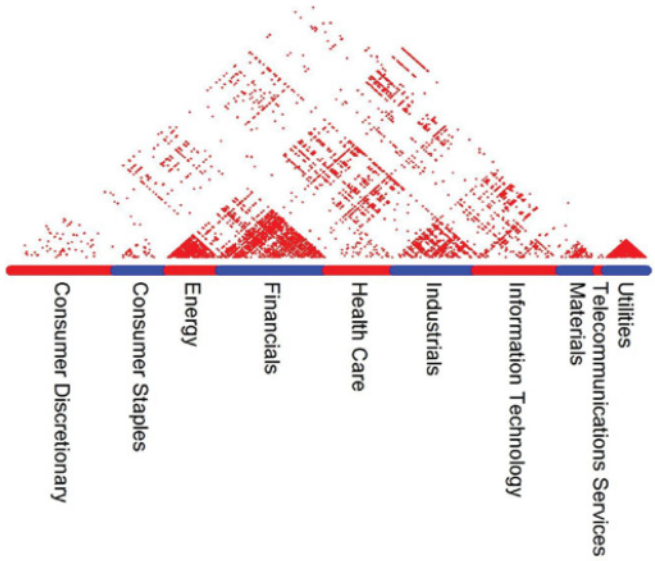


Fig. 3. Correlation analysis for stocks in S&P 500 sectors. The pairwise correlations among 500 stocks, which are grouped according to their sectors, are calculated. The redness indicates the correlation strength.

where $t \geq m$ and $\alpha = \frac{2}{m+1}$. We use $m = 1, 2, \ldots, 10, 20, 30, 40, 50, 100$ as inputs, so 15 EMAs are included in our DNN framework. Notice that $m = 1$ leads to $\alpha = 1$ which means $EMA_t(1, S_{t,j}) = S_{t,j}$.

The Second Hidden Layer: the AD, ADv and ADR indicators. These indicators are used to measure the market breadth [Brown and Cliff 2004; Zakon and Pennypacker 1968]. The AD indicator counts the net number of advancing stocks within the 5-minute interval at time t, which is the number of advancing stocks subtracting the number of declining stocks. ADv is the net volume of advancing stocks within a time interval t, which is the volume of advancing stocks subtracting the volume of declining stocks. The ADR is calculated by dividing the volume of advancing stocks by the volume of declining stocks.

3.2.2. Sector Analysis. Since the price movements of different stocks are correlated, we try to incorporate these correlations into our DNN model to borrow information from other stocks in predicting the stock of interest.

Here we focus on the S&P 500 and discuss the dependence relationship in the market. Based on returns in the first 2000 5-minute training intervals in the studied time period, pair-wise correlations among the 500 stocks have been estimated and plotted in the Figure 3. Our analyses show that stocks tend to move in the same direction within each of the ten S&P 500 sectors, including Consumer Discretionary, Consumer Staples, Energy, Financials, Health Care, Industrials, Materials, Information Technology, Telecommunication Services, and Utilities. While there are sporadic correlations between stocks in different sectors, stock correlations are largely within sectors. For example, stocks in Energy or Utilities have very litter correlations with stocks outsides these sectors.

This correlation structure is expected to be useful for prediction; however, how to incorporate of these correlations into the DNN structure is challenging. First, if all 500 stocks are directly used as inputs, there would be a lot of redundant interactions due to clustered correlation structure. Second, because of the multiplicity of financial assets,

the correlation information is high dimensional, and it is computationally demanding to consider all correlations simultaneously. To address these issues, in our DNN design, we propose to first summarize stocks information, which are highly correlated within sectors, to the sector units (W neurons in Figure 2), and then feed these units to the first hidden layer during the self-learning process. This structure avoids the redundant stock-stock interactions and is also computationally efficient since local connections employ much fewer parameters.

3.3. Target Function and Regularization

The complete model parameter set, Ω, consists of all links in the DNN structure, that is, $\Omega = \{\alpha_{np}, \beta_{mn}, \beta_{mn'}, \theta_{km} \text{ and } \theta_{km'} : n = 1, \ldots, N; n' = 1, \ldots, 15; p = 1, \ldots, P; k = 1, \ldots, K; m = 1, \ldots, M; m' = 1, \ldots, 3\}$.

Since the output is a continuous variable, the target function can be simply set to be the sum-of-squared errors:

$$R(\Omega) = \sum_{i=1}^{I} (y_i - \hat{y}_i)^2 \tag{6}$$

where y_i is the actual return for i^{th} training data point, and $\hat{y}_i$is the predicted return based on neural network model.

The number of parameters in Ω is typically large, so it is very easy to over-fit the network model and in those situations and a global minimizer of $R(\Omega)$ is undesirable. Usually, early stopping rules or regularization methods can be employed to obtain local minimizers. Here we implement the algorithm with the regularization method by adding a L_2. -penalty terms to the target function, which now becomes

$$R_p(\Omega) = R(\Omega) + \lambda J(\Omega) = R(\Omega) + \lambda \Omega^2 \tag{7}$$

where $R(\Omega)$ is the original error function, $J(\Omega) = \Omega^2$ is the penalty term, and $\lambda \geq 0$ is a tuning parameter. Larger values of λ tend to shrink the weights toward zero. To choose an appropriate value for the hyper-parameter λ, cross-validation (CV) can be applied. However, in practice we found a pre-specified penalization coefficient λ of 0.5 works well. Therefore, to save the computing time on CV, we prefix λ to be 0.5 in our implementation.

3.4. Learning Algorithm

Minimizing the penalized target function in the above is a non-linear, non-convex problem, and numerical algorithms need to be applied. We employed a gradient descent algorithm, which is given as follows.

Let $\eta \in \Omega$ represent a parameter in neural network models. The target function is minimized in an iterative process:

$$\eta_{r+1} = \eta_r - \gamma_r * \left. \frac{\partial R_p}{\partial \eta} \right|_{\eta = \eta_r} \tag{8}$$

where γ_r is the learning rate at step r which is chosen to be: $\gamma_r = 1/r$. The process stops when the relative change of R_p is less than 10^{-6}.

Algorithm 1 describes the Gradient Descent Algorithm applied in our DNN framework in details. The algorithm is implemented using the C language, which was compiled into a dynamic link file and loaded into R version 3.1.2 to carry out the subsequent analyses.

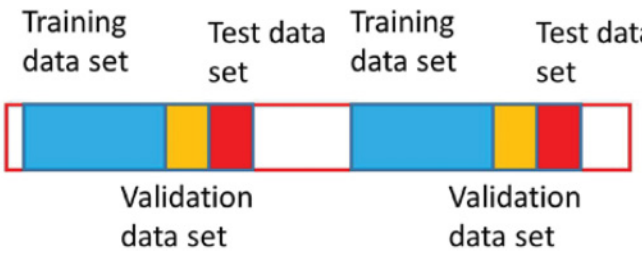


Fig. 4. Dynamic back-testing process. The most recent intervals are divided into training (Blue) and validation (Yellow) periods. The NN models are trained with the training data and then their performances are evaluated with the validation data. The top NN models are then combined to predict the movement in the test period (Red).

ALGORITHM 1: Gradient Descent Algorithm

Input: Initialized Ω_0: *set each* $\eta_0 \in \Omega_0$ *to be a random value from distribution* $N(0,\ 1)$.
Output: Optimized parameters for our DNN model Ω
Calculate $R_0 = R_p(\Omega_0)$
For *r from 0 to 200* **do**
  For *each* η *in* Ω **do**
    Calculate the first derivatives of $R_p(\Omega)$ *on point* Ω_r: $\left.\frac{\partial R_p}{\partial \eta}\right|_{\Omega=\Omega_r}$
    Update η: $\eta_{r+1} = \eta_r - \frac{1}{r+1} * \left.\frac{\partial R_p}{\partial \eta}\right|_{\eta=\eta_r}$
  End
  Ω_{r+1} = *the set of all* η_{r+1}
  Calculate new residual function: $R_{r+1} = R_p(\Omega_{r+1})$
  Relative change of R = $\left|\frac{R_{r+1}-R_r}{R_r}\right|$
  If $R < 10^{-6}$ **then**
    return Ω_{r+1};
    break;
  End
End

3.5. The Ensemble NN and Back-Testing

In practice, we found that predictions based on one single NN model are usually unstable. That is, for the same period of data, if we train NNs multiple times, the prediction results from each trained NN may vary quite dramatically. The reason may be twofold: (1) given the high dimensionality of the parameter space, the random start point of the algorithm may yield significant differences; (2) since the prediction pattern in the financial market is well known to be low, different local minima from the target functions may perform quite differently. Therefore, to increase the stability of our NN model, for each training data set, 1000 NN models will be trained and the top 50% best performers will be pooled for prediction. We call this process as the ensemble NN, which led to the following dynamic backtesting process and is shown schematically in Figure 4. The percentage of best performer controls the trade-off between accuracy and stability. That is, if more models are pooled together, the average return would be less than that from the procedure when fewer models are pooled, but the resulting variation would also be less. We have tested a range of percentages from 1% to 90% and found that 20–50% can yield similarly good and stable returns. To be conservative (prefer more stable result), we choose to use the 50% best performers in this work.

Algorithm 2 provides more details about the process. Specifically, at time interval t, data in intervals from $t - 2100$ to $t - 101$ form the training data set, based on which 1000 NN models are then trained, each built upon 1000 randomly selected intervals from the training data set. The 1000 NN models are subsequently applied on a validation data set with intervals from $t - 100$ to $t - 1$, and 50% of the 1000 models with the best prediction accuracy on the validation set are chosen to form a model "committee". Average values or majority votes from this "committee" are then taken as the one-step predictions for future returns or up/down prediction, respectively. Because of the "moving" design in the back-testing, theoretically our model should be updated at each new time point, but this leads to very high computation cost. So practically, we choose to update the "committee" every 100 time intervals. That is, models for intervals from $t + 1$ to $t + 100$ are kept the same. This is not unreasonable as the fitted model is generated based on the committee and would only be affected when a reasonable proportion of training data have been replaced. This ensemble framework has been applied to the three NN models in Section 4.2 to ensure fair comparison among them.

ALGORITHM 2: Dynamic Back-Testing Process

Input: Real data beginning at $t = 1$
Output: Our DNN model predicted values from $t = 2101$ to the end

For *t from 2101 to the end of available time point* **do**
  If t mod 100 != 1:
    Each model in the existing committee predicts on time point t;
    The mean of all predictions is the final prediction for time t
  Else:
    Training data range from $t - 2100$ *to* $t - 101$
    Validation data range from $t - 100$ *to* $t - 1$
    For *i from 1 to 1000* **do**
      Train ith DNN model based on a random starting point and a bootstrapped sample from the training data;
      Calculate prediction accuracy p_i *on validation data*
    End
    Choose 500 models with the highest p_i'*s to form the new committee;*
    Each model in the new committee predicts on time point t;
    The mean of all predictions is the final prediction for time t
  End
End

4. EXPERIMENTS AND RESULTS

4.1. Data

Next, we evaluate the performance of the proposed method based on the U.S. S&P 500 stock data, ranging from January 1, 2013 to May 31, 2013. There are totally 104 trading days and each day contains 78 5-minute intervals, corresponding to 8112 time points (observations). The raw data include open, high, low and close prices and volume at each time point. Table I presented a summary of statistics for the 500 stocks' trading volumes and their market capitals on May 31, 2013.

4.2. Benchmark Methods

To benchmark the performance of our proposed method, we include four baseline methods for comparison, which are given as follows.

Table I. Descriptive Statistics for the 500 Stocks in Our Sample

	Mean	Standard Deviation	Median	Min	Max
Volume	11.8	60.2	3.9	0	24951.1
Market cap	32.1	51.2	15.3	3.0	422.1

Notes: Trading volumes (unit: thousands shares); market capitalizations (unit: billions USD).

4.2.1. ARMA-GARCH. The first one is ARMA(1,1)-GARCH(1,1), a well-known approach for financial time series. This approach combines the techniques of autoregressive-moving-average (ARMA) model and generalized autoregressive conditional heteroscedasticity (GARCH) model. For the time series of return for stock j, denoted by $X_{t,j}$, the basic model is expressed as

$$X_{t,j} = \phi_{0,j} + \phi_{1,j} X_{t-1,j} + \epsilon_{t,j} + \varphi_{1,j}\epsilon_{t-1,j} \tag{9}$$

where $\phi_{0,j}$ is a constant term; $\phi_{1,j}$ is the coefficient of $X_{t-1,j}$ on ; $\epsilon_{t,j}$ and $\epsilon_{t-1,j}$ represent the error term obtained at t and $t-1$; $\varphi_{1,j}$ is the coefficient of $\epsilon_{t-1,j}$. This is the form of standard ARMA model. To better estimate the error variance, the GARCH process is added.

Considering the error term in (9) follows i.i.d $N(0, \sigma_{t,j}^2)$,

$$\epsilon_{t,j} = \sigma_{t,j} u_{t,j} \tag{10}$$

and the time-dependent variance is estimated by

$$\sigma_{t,j}^2 = \alpha_{0,j} + \alpha_{1,j}\sigma_{t-1,j}^2 + \beta_{1,j}\epsilon_{t-1,j}^2 \tag{11}$$

The unknown parameters $\{\phi_0, \phi_1, \varphi_1, \alpha_0, \alpha_1, \beta_1\}$ are estimated via maximum likelihood method. Notice that unit innovation $u_{t,j}$ is a strong white noise process which can be estimated by $\hat{u}_{t,j} = \frac{\hat{\epsilon}_{t,j}}{\hat{\sigma}_{t,j}}$. The ARMA–GARCH model not only implements the basic format of time series model but considers the volatility dynamics or the error term. The model is widely used in financial modeling and has been considering as the most reliable benchmark in finance. We implemented it with the "fGarch" package in R.

4.2.2. Armax-Garch. The second baseline method is an extension of ARMA(1,1)-GARCH(1,1). For the given time series $X_{t,j}$, ARMAX-GARCH model considers endogenous variable $X_{t,j}$ and a set of exogenous variables $Y_{1t,j}, Y_{2t,j}, \ldots, Y_{1t,j}$. Then, the original model is modified as:

$$X_{t,j} = \phi_{0,j} + \phi_{1,j} X_{t-1,j} + \sum_{i=1}^{n} \mu_{i,t-1,j} Y_{i,t-1} + \epsilon_{t,j} + \varphi_{1,j}\epsilon_{t-1,j} \tag{12}$$

where n is the number of exogenous variables. Referred to Section 3.2.1, our ARMAX-GARCH model uses intra-interval proportions, exponential moving averages, Advance/Decline, Advance/Decline volume and Advance/Decline Ratio indicators as input variables. The ARMAX–GARCH model is implemented with the "fGarch" package in R.

4.2.3. The Single-Layer NN Model (SNN). The third baseline method is the single-layer NN model (SNN), which is the simplest but probably also the most widely used NN structure. It consists of three layers: the bottom input layer, the middle hidden layer, and the top output layer. Every neuron in the hidden layer is fully connected to all other neurons in the input and output layers. The input neurons take the same variables as the ARMAX-GARCH model in the previous section. The SNN model is implemented with the "nnet" package in R and applies the same ensemble and back-testing process in Section 3.5.

4.2.4. The Regular Double-Layer NN Model (RDNN). The fourth baseline method is the regular double-hidden-layer neural network. It consists of four layers: the bottom input layer, two middle hidden layers, and the top output layer. Every neuron in the hidden layers is fully connected with all other neurons in the upper and lower layers, which is different from the structure we propose. The input neurons take all the variables in Section 2.2: 499 X variables, 30 Q variables, 15 EMA variables and 3 advance/decline variables. The first hidden layer has 25 neurons and the second layer has 6 neurons. The regular double-layer NN (RDNN) model is implemented with the "neuralnet" package in R with the same ensemble and back-testing process introduced in Section 3.5.

4.3. Evaluation Metrics

For each stock j at each time t, a prediction is made for the next time point $t + 1$ based on a specific method m. If the model prediction is consistent with the true price movement, a profit is yielded; otherwise a loss is incurred. Let us denote the return for model m at time t as $r_{m,t,j} = sgn(\hat{y}_{m,t,j} y_{t,j}) * |y_{t,j}|$. $y_{t,j} = X_{t+1,j}$ is the true observed return, and $\hat{y}_{m,t,j}$ is the predicted return from model m for the next 5-minute interval. Assume the total number of time points being tested is T_0, we used the following criteria to evaluate/compare the performance of different models.

4.3.1. Absolute Return.

$$r_{m,j} = \prod_{t=1}^{T_0} (1 + r_{m,t,j}) \tag{13}$$

This is equivalent to a strategy of buying or short selling a fixed number of shares according to the model prediction on the 5-minute interval at time point $t + 1$.

4.3.2. Sharpe Ratio.

$$SP_m = \frac{\bar{r}_{m,j}}{s_{m,j}} \tag{14}$$

where $\bar{r}_{m,j}$ is the average return defined as $\sum_{t=1}^{T_0} r_{m,t,j}/T_0$, and $s_{m,j}$ is the standard deviation defined as $\sqrt{\frac{1}{T_0-1} \sum_{t=1}^{T_0} (r_{m,t,j} - \bar{r}_{m,j})^2}$. Sharpe ratio quantifies the risk efficiency of a model performance. It represents the risk-adjusted returns that can be considered as a more reliable evaluation measurement.

4.3.3. Prediction Accuracy.

$$A_{m,j} = \frac{\sum_{t=1}^{T_0} I(r_{m,t,j} > 0)}{T_0} \tag{15}$$

Prediction accuracy is calculated as simply the ratio of the total number of correct predictions on the direction of the price movement over the total of tested intervals.

4.4. Overall Performance

4.4.1. Forecasting Comparison. To study the forecasting performances, our DNN method, and four benchmark methods were applied to 100 stocks with the largest capitals in the S&P 500. The prediction accuracy is used as the main metric for the comparison. In addition, we also explored the performances of the five methods in predicting large returns. The rationale is that small returns close to zeros may mainly represent random noises that are hard to predict or even unpredictable. This is also compatible with a trading strategy that we trade only when predicted returns are larger than a certain threshold. Hence, we designed an adjusted testing process as follows.

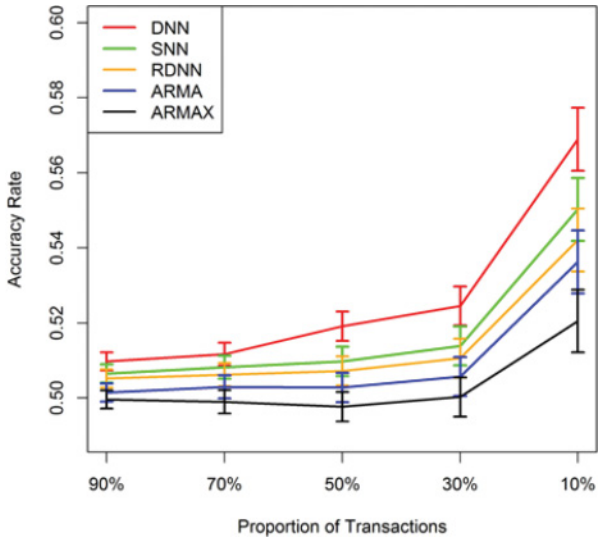


Fig. 5. Prediction accuracy rates for five methods at different proportions of transaction. The mean and 95% confidence intervals were plotted for DNN (red), SNN (green), RDNN (yellow), ARMA-Garch (blue) and AMRAX-Garch (black).

Let $C_{m,j}$ be the lower bound return threshold for stock j based on method m. Then the adjusted returns can be defined by:

$$r^*_{m,t,j} = \begin{cases} r_{m,t,j}, & when \left|\hat{y}_{m,t,j}\right| > C_{m,j} \\ 0, & when \left|\hat{y}_{m,t,j}\right| \leq C_{m,j} \end{cases} \tag{16}$$

where $m = 1, \ldots, 5$ represented five methods: DNN, SNN, RDNN, ARMA-GARCH and ARMAX-GARCH. $j = 1, \ldots, 100$ represents the 100 stocks on which predictions have been made. The new accuracy rate for method m and stock j is defined as:

$$A^*_{m,j} = \frac{\sum_{t=1}^{T_0} I(r^*_{m,t,j} > 0)}{\sum_{t=1}^{T_0} I\left(r^*_{m,t,j} \neq 0\right)} \tag{17}$$

To ensure a fair comparison, the five tested methods were assessed with approximately the same number of transactions. Correspondingly, different thresholds $C_{m,j}$ were used to yield the same proportions of transactions in stock j for each method m. For example, to execute transactions in $100 * (1 - p)\%$ intervals with the largest absolute predictions, $C_{m,j}$ is then the p^{th} percentile of all predicted values. We set $p =$ 90%, 70%, 50%, 30%, or 10%, and obtained a trend of prediction accuracies. We pool the accuracy estimates for the 100 stocks and summarize the results for the five methods in Figure 5. It shows that: (1) as the proportion of transaction decreased, performances of all five methods become better in terms of the prediction accuracy, which suggests that strategies focusing on larger predicted values would have better predictions on stock movement; (2) Our newly proposed DNN model outperforms the other four benchmark methods, especially when the proportion of transactions is below 50%.

Table II presented the annualized Sharpe ratios for the five methods at corresponding proportions of transactions. Medians with 25%, 75% quantiles of the 100 stocks were shown in the table. We observe that DNN outperforms the other methods in 4 out of 5 proportions of transactions (90%, 70%, 50%, and 30%), and DNN reached the highest median annualized Sharpe ratio when the proportion of transactions is 50%.

Table II. Annualized Sharpe Ratios for Five Methods at Different Proportions of Transactions

	Proportion of Transactions				
	90%	70%	50%	30%	10%
DNN	1.50	1.29	1.70	1.55	0.89
	(−1.07, 4.11)	(−1.24, 4.33)	(−0.50, 4.55)	(−0.73, 4.05)	(−1.10, 3.00)
SNN	0.92	1.08	1.20	1.23	1.66
	(−0.60, 3.05)	(−1.17, 3.69)	(−1.09, 3.10)	(−0.83, 3.51)	(−0.14, 3.39)
RDNN	1.12	0.92	0.92	1.26	1.50
	(−0.96, 2.51)	(−1.04, 2.34)	(−1.31, 2.70)	(−1.02, 2.79)	(−1.15, 3.18)
ARMA	0.63	0.16	0.23	0.14	0.48
	(−1.21, 1.47)	(−1.40, 1.53)	(−1.35, 1.88)	(−1.63, 1.63)	(−1.51, 1.80)
ARMAX	−0.44	−0.25	−0.55	0.06	0.27
	(−1.83, 1.31)	(−1.75, 0.86)	(−2.13, 0.93)	(−1.97, 1.48)	(−1.92, 1.93)

Notes: Each cell presented the median with 25%, 75% quantiles of annualized Sharpe ratios among the 100 stocks with corresponding method and proportion of transactions applied.

4.4.2. Computational Efficiency. Although efficient R programs have been developed to implement our proposed model, the computation load related to the ensemble DNN is still heavy. Roughly, one DNN fitting takes approximately 0.8 seconds in a regular workstation, which means that fitting 1000 DNNs at one step would take about 13.3 minutes. If this model is applied to support real-time trading, a single-processor system can afford to update the model every three 5-minute intervals.

Our proposed framework consists of 1000 independent Neural Network models, which could be easily parallelized. Running our framework on a server with enough cores (e.g. greater than 1000 cores) would reduce the computational time to the equivalence of fitting only one model, which is about one second. Moreover, the computational cost of combining results from the 1000 cores and producing final predictions is relatively small. Therefore, our framework should be efficient in handling 5-min forecasting decision-making tasks or even higher frequencies.

There are other ways to reduce computation burdens in our forecasting framework. Suppose 1000 Neural Network models have been fitted to predict $t + 1$ interval's close price at time t. When we move to the $t + 1$ interval, only the oldest data point from previous training data set would be dropped off and replaced with the newest time $t + 1$ data. Hence, among all 1000 existing models, only those with the oldest data point need to be updated, which would further speed up our model implementation.

4.5. Case Studies

In this section, we examine the performance of each method with respect to individual stock or sector, particularly in the case of $p = 50\%$ (the 3rd column in Table II).

Table III summarizes the top 10 performers of each method with respect to the Sharpe ratio. It is clear that the top lists in these ARMA based models (ARMA and ARMAX) are quite different from those in the NN-based models (DNN, SNN, and RDNN), and in general, NN based models perform better than the ARMA based models. Among the top lists of DNN and RDNN, 5 of them are the same (JP Morgan, Johnson & Johnson, Coca-Cola, Google and Walt Disney). Pair-wise t-tests, which are based on the Sharpe ratios of the 100 stocks, between the DNN and RDNN or SNN shows Sharpe ratios in DNN is significantly higher those in RDNN or SNN, with p-values of 0.0002 and 0.015, respectively. These results demonstrate that although our DNN model outperforms the regular DNN and ARMA models in overall performance, it is not a "super" model that can predict better in every single stock, which highlights the complexity of the financial data.

Table IV summarizes the performance of the five models in ten business sectors. Medians and 25%, 75% quantiles of annualized Sharpe ratios are used as representa-

Table III. Sharpe Ratios of the Top-10 Annualized for Each of the Five Methods When $p = 50\%$

DNN		SNN		RDNN		ARMA		ARMAX	
Ticker	Sharpe Ratio	Ticker	Sharpe Ratio	Ticker	Sharpe Ratio	Ticker	Sharpe Ratio	Ticker	Sharpe Ratio
JPM	8.865	T	7.870	DIS	7.006	MA	13.339	COF	4.547
JNJ	8.718	JNJ	7.554	KO	6.966	MSFT	6.413	JNJ	3.547
T	8.572	MO	7.448	JPM	6.087	AMGN	5.262	RTN	3.529
KO	8.501	TGT	7.207	F	5.789	SPG	4.792	MON	3.357
BAX	8.047	JPM	7.193	JNJ	5.105	COP	4.427	TXN	3.337
XOM	7.868	GOOG	6.328	SLB	4.603	DVN	3.891	GD	3.324
COST	7.360	SO	6.113	GOOG	4.484	C	3.833	INTC	3.250
GOOG	7.293	DIS	5.934	PEP	4.458	WMT	3.390	GILD	3.227
MO	7.206	WMT	5.929	LLY	4.061	NSC	3.268	AAPL	2.868
DIS	6.954	AXP	5.902	EMC	3.793	BMY	3.137	WMT	2.780

Table IV. Performance of Five Methods within Ten Business Sectors When Executing 50% Transactions

Sectors	# of stocks	DNN	SNN	RDNN	ARMA	ARMAX
Information Technology	16	1.12 (−1.28, 3.82)	0.02 (−1.88, 3.33)	−0.89 (−2.44, 1.43)	0.54 (−1.94, 1.40)	0.02 (−1.83, 2.30)
Health Care	11	1.31 (−1.08, 2.39)	0.44 (−2.59, 2.98)	0.28 (−1.20, 2.56)	0.19 (−1.54, 2.14)	−0.34 (−2.71, 1.44)
Utilities	3	2.72 (1.81, 3.37)	2.45 (0.61, 5.16)	1.76 (1.12, 3.00)	−1.87 (−3.55, −0.17)	−2.36 (−3.49, −0.95)
Financials	14	1.86 (0.53, 3.90)	1.52 (−1.02, 3.00)	1.56 (0.68, 3.25)	0.72 (−0.76, 2.11)	−0.86 (−2.21, 0.34)
Consumer Discretionary	12	2.38 (−0.12, 4.68)	1.90 (0.09, 4.69)	1.87 (−0.03, 3.48)	−0.77 (−1.38, 0.08)	−1.97 (−2.74, −0.72)
Energy	12	1.60 (−0.87, 3.96)	0.14 (−1.61, 2.04)	0.70 (−0.04, 1.63)	0.38 (−1.62, 1.98)	−1.00 (−1.84, −0.03)
Industrials	14	1.89 (−0.19, 4.95)	1.65 (0.61, 3.40)	0.82 (−0.09, 2.14)	0.61 (−1.18, 2.38)	−0.03 (−2.22, 2.30)
Consumer Staples	12	3.38 (0.97, 6.15)	2.36 (−0.33, 5.16)	0.40 (−2.21, 2.76)	0.56 (−0.73, 1.57)	0.00 (−1.08, 1.01)
Materials	4	3.03 (1.76, 5.07)	1.43 (0.47, 2.45)	−1.02 (−1.52, −0.50)	0.22 (−0.15, 0.96)	1.65 (0.71, 2.89)
Telecommunications Services	2	5.07 (3.32, 6.82)	4.99 (3.55, 6.43)	1.54 (1.37, 1.71)	1.60 (1.45, 1.75)	−0.72 (−0.81, −0.62)

Notes: Each cell presents the median with 25%, 75% quantiles of annualized Sharpe ratios among stocks in corresponding sectors.

tive statistics. At the sector level, our proposed DNN model outperforms the other four methods in almost every sector and perform particularly well in the sectors of Utilities, Consumer Discretionary, Consumer Staples, Materials and Telecommunications Services (median annualized Sharpe ratio is larger than 2), indicating how to best apply the DNN in practice. However, why DNN perform well in these sectors is still unknown and is worth further investigation.

5. CONCLUDING REMARK AND DISCUSSION

In this article, we propose a novel double-layer neural network framework for market movements forecasting with high-frequency data. Our framework dynamically filters important information from the complex dependency structure among different stocks and business sectors. It also hierarchically explores momentum signals from selected technical indicators. We investigated and benchmarked the performance of the newly proposed model, together with two regular NN models and two classical econometric models for high-frequency forecasting. The results show that our approach outperforms these benchmark approaches in terms of adjusted returns as well as the prediction accuracies. Although a large amount of transaction costs might reduce the profits in

high-frequency trading, our model can still help in providing effective solutions. For example, focusing on predictions with large returns may be used to design useful trading strategies as it would not only reduce the number of transactions but also increase the prediction accuracy rates.

In our model design, we employed two hidden layers in the NN construction. Deeper networks (i.e., deep learning), which have been very successfully in other fields such as image analysis, are certainly a natural extension. However, as we have demonstrated that the RDNN model performs almost the same as, or a little worse than, the SNN, simply increasing the number of hidden layers might not improve the performance of NN models much in the high-frequency financial data. Nevertheless, it is an interesting question to examine if deeper networks with similar hierarchical structure in Figure 2 could be developed to further improve the power of forecasting.

REFERENCES

Torben G. Andersen, Tim Bollerslev, and Francis X. Diebold. 2007. Roughing it up: Including jump components in the measurement, modeling, and forecasting of return volatility. *Rev. Econom. Stat.* 89, 4, 701–720.

Torben G. Andersen, Tim Bollerslev, Francis X. Diebold, and Heiko Ebens. 2001. The distribution of realized stock return volatility. *J. Finan. Econ.* 61, 1, 43–76.

Andrés Arévalo, Jaime Niño, German Hernández, and Javier Sandoval. 2016. High-frequency trading strategy based on deep neural networks. In *Intelligent Computing Methodologies*. Springer International Publishing, 424–436.

Senthil Arasu Balasubramanian, Jeevananthan Manickavasagam, Thamaraiselvan Natarajan, and Janarthanan Balakrishnan. 2015. An experimental analysis of forecasting the high frequency data of matured and emerging economies stock index using data mining techniques. *Int. J. Operat. Res.* 23, 4, 406–426.

G. W. Brown and M. T. Cliff. 2004. Investor sentiment and the near-term stock market. *J. Empir. Finance.* 11, 1, 1–27.

Pei-Chann Chang, Di-di Wang, and Chang-le Zhou. 2012. A novel model by evolving partially connected neural network for stock price trend forecasting. *Expert Syst. Applic.* 39, 1, 611–620.

Gong-meng Chen, Michael Firth, and Oliver M. Rui. 2001. The dynamic relation between stock returns, trading volume, and volatility. *Finan. Rev.* 36, 3, 153–174.

Tai-Liang Chen, Ching-Hsue Cheng, and Hia Jong Teoh. 2007. Fuzzy time-series based on Fibonacci sequence for stock price forecasting. *Phys. A: Stat. Mech. Applic.* 380, 377–390.

Thomas C. Chiang, Hai-Chin Yu, and Ming-Chya Wu. 2009. Statistical properties, dynamic conditional correlation, scaling analysis of high-frequency intraday stock returns: Evidence from dow-jones and nasdaq indices. *Phys. A*. 388, 8, 1555–1570.

Taufiq Choudhry, Frank McGroarty, Ke Peng, and Shiyun Wang. 2012. High-frequency exchange-rate prediction with an artificial neural network. *Intell. Syst. Account., Finan. Manage.* 19, 3, 170–178.

Wensheng Dai, Jui-Yu Wu, and Chi-Jie Lu. 2012. Combining nonlinear independent component analysis and neural network for the prediction of Asian stock market indexes. *Expert Syst. Applic.* 39, 4, 4444–4452.

Philip Hans Franses and Dick Van Dijk. 1996. Forecasting stock market volatility using (nonlinear) GARCH models. *J. Forecast.* 229–235.

M. Ghiassi, J. Skinner, and D. Zimbra. 2013. Twitter brand sentiment analysis: A hybrid system using n-gram analysis and dynamic artificial neural network. *Expert Syst. Applic.: An Int. J.* 40, 16, 6266–6282.

Rosella Giacometti, Marida Bertocchi, Svetlozar T. Rachev, and Frank J. Fabozzi. 2012. A comparison of the Lee-Carter model and AR-ARCH model for forecasting mortality rates. *Insurance: Math. Econom.* 50, 1, 85–93

M. D. Rafiul Hassan, Baikunth Nath, and Michael Kirley. 2007. A fusion model of HMM, ANN and GA for stock market forecasting. *Expert Syst. Applic.* 33, 1, 171–180.

C. C. Holt. 2004. Author's retrospective on 'forecasting seasonals and trends by exponentially weighted moving averages. *Int. J. Forecast.* 20, 1, 11–13. DOI:10.1016/j.ijforecast.2003.09.017.

Sheng-Hsun Hsu, J. J. Po-An Hsieh, Ting-Chih Chih, and Kuei-Chu Hsu. 2009. A two-stage architecture for stock price forecasting by integrating self-organizing map and support vector regression. *Expert Syst. Applic.* 36, 4, 7947–7951.

Roger D. Huang and Hans R. Stoll. 1994. Market microstructure and stock return predictions. *Rev. Finan Stud.* 7, 1, 179–213.

Yakup Kara, Melek Acar Boyacioglu, and Ömer Kaan Baykan. 2011. Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the istanbul stock exchange. *Expert Syst. Applic.* 38, 5, 5311–5319.

Jonathan M. Karpoff. 1987. The relation between price changes and trading volume: A survey. *J. Finan. and Quantitat. Anal.* 22, 01, 109–126.

Bryan Kelly and Seth Pruitt. 2015. The three-pass regression filter: A new approach to forecasting using many predictors. *J. Econometrics.* 186, 2, 294–316.

Christopher G. Lamoureux and William D. Lastrapes. 1990. Heteroskedasticity in stock return data: volume versus GARCH effects. *J. Finance* 45, 1, 221–229.

Fajiang Liu and Jun Wang. 2012. Fluctuation prediction of stock market index by Legendre neural network with random time strength function. *Neurocomputing* 83, 12–21.

Ping-Feng Pai and Chih-Sheng Lin. 2005. A hybrid ARIMA and support vector machines model in stock price forecasting. *Omega* 33, 6. 497–505.

Jigar Patel, Sahil Shah, Priyank Thakkar, and K. Kotecha. 2015. Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. *Expert Syst. Applicat.* 42, 1, 259–268.

Alexis Akira Toda. 2012. The double power law in income distribution: Explanations and evidence. *J. Econom. Behav. Organ.* 84, 1, 364–381.

Alexis Akira Toda and Kieran James Walsh. 2014. The double power law in consumption and implications for testing euler equations. *Available at SSRN 2319454*.

Chih-Fong Tsai and Yu-Chieh Hsiao. 2010. Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches. *Decis. Supp. Syst.* 50, 1, 258–269.

Johannes Voit. 2013. *The Statistical Mechanics of Financial Markets*: Springer Science & Business Media.

Martin L. Weitzman. 2007. Subjective expectations and asset-return puzzles. *Amer. Econom. Rev.*:1102–1130.

J. Zakon and J. C. Pennypacker. 1968. An analysis of the advance-decline line as a stock market indicator. *J. Finan. Quant. Anals.* 3, 3, 299–314

Received March 2016; revised August 2016; accepted November 2016