Time Series Prediction: A Combination of Deep Learning and Statistical Models

Draft version

Quoc Luu
Quantitative and Computational Finance
John von Neumann Institute
Ho Chi Minh, Vietnam
quoc.luu2015@qcf.jvn.edu.vn

Uyen Pham
Quantitative and Computational Finance
John von Neumann Institute
Ho Chi Minh, Vietnam
Economic Mathematics
University of Economics and Law
Ho Chi Minh, Vietnam
uyenph@uel.edu.vn

***Abstract*—Stock market is an important capital mobilization channel for economy. However, the market has potential loss. There are many approaches to reduce risk such as portfolio construction and optimization, hedging strategies. Hence, it is critical to leverage time series prediction techniques to achieve higher performance in stock market. In this work, we propose an incorporated model that combines Sequence to Sequence with Long-Short Term Memory model of deep learning and structural models time series. We choose 21 most traded stocks with over 500 trading days from VN-Index of Ho Chi Minh Stock Exchange and HNX-Index of Hanoi Stock Exchange (Vietnam) to perform the proposed model and compare their performance with pure structural models and Sequence to Sequence. Results suggest that the Sequence to Sequence with LSTM model of deep learning and structural models time series achieve higher performance with lower prediction errors in terms of mean absolute error than existing models. This work significantly contribute to literature of time series prediction as our approach can relax heavy assumptions of existing methodologies such as Auto-regressivemoving-average model, Generalized Auto-regressive Conditional Heteroskedasticity. In practical, investors from Vietnam stock market can use the proposed model to develop trading strategies.**

***Index Terms*—LSTM; Seq2Seq; Structural models; Hybrid model**

I. Introduction

Describing the behavior of the observed time series plays a critical role to understand the past and predict the future in many disciplines. In quantitative finance, time series prediction is a very important task for risk management to measure of the uncertainty of investment return [49], portfolio construction for hedge fund [15], high-frequency trading [51].

However, creating high accuracy prediction of time series with low error is not an easy job, due to high fluctuations of stock market. From this perspective, there have been many methods that were proposed to study historical patterns of time series data to crate high quality of stock price prediction. To measure quality of a prediction, forecast error indicators were compute from actual price and predicted price. Mean Square Error (MSE) or Root-Mean-Square Error are populate indicators for the measurement works.

In terms of time series prediction, there are many approaches to address the matter. However, there are two methods which have been widely adopted. The first one is univariate analysis to capture volatility. They include autoregressive models ($AR(p)$), moving-average models ($MA(q)$), combination of autoregressive and moving-average models ($ARMA(p,q)$) for linear processes, and generalized autoregressive conditional heteroscedastic ($GARCH(p,q)$) for nonlinear processes to model return of stocks [4]. By differencing, a transformation from price to return of a stock poses a problem. Unobserved components (e.g. seasonal component, trend) of raw series were eliminated. Furthermore, differencing is hard to interpret and select adequate model. Hence, the second approach have been proposed to fill the gap [24]. It is called Structural Time Series Models which comprises trend component, seasonal component, and a random irregular component to model a time series without differentiation.

With revolution of computational power, beside statistical models, machine learning and deep learning models have been widely adopted to solved many problems from academic to industry. In financial time series prediction, we can leverage these models to achieve higher accuracy. In deep learning, models are considered black-box with billions of parameters. However, feature engineering is still an important work to improve model accuracy [13]. Furthermore, neural network like Long-Short Term Memory of deep learning can link current event to previous events while Structural Time Series Models only depends on previous event. Hence, it is critical to combine these two approaches to address the limitations.

In this work, we step-by-step describe procedures to perform fitting data with Structural Time Series Models, Sequence to Sequence model, and the combination of these two models. We report the results of the fitting process to

21 stocks listed on Ho Chi Minh Stock Exchange, then we compare the prediction results.

II. Literature Review

A. Structural Time Series Models

Decomposition of time series is an important procedure. Traditionally, regarded as a functional depended on time and deterministic, non-stationary time series are often detrended by applying difference to construct models from the processed-data. It is suggested that this procedure may lead to misleading results if trend is not deterministic [37]. A structural time series models are a decomposable time series in terms of three components of trend, seasonality and cycle [23] [30]. It is defined as following equation:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t, t = 1, , T \tag{1}$$

where $g(t)$ is stochastic and non-periodic changes trend, $s(t)$ is a seasonal stationary linear process with periodic changes (e.g. quarterly, yearly seasonality), and $h(t)$ is a cyclical frequency of time occurring on potentially irregular schedules over one or more days [44].

Many researches strongly support the model in practice have been carried out. For instance, [24] [22] shown that class of structural models have several advantages over the seasonal ARIMA models adopted and are applicable to model cycles in macroeconomic time series. [33] decomposed time series into trend, seasonal, globally stationary autoregressive and observation error components with state space Kalman filter and used Akaike minimum AIC procedure to select the best of the alternative state space models. [44] use structural models for forecasting of business time series.

1) Trend Models: The local linear trend is a process can be regarded as a local approximation to a linear trend. The stochastic linear process can be described as:

$$y(t) = g(t) + \epsilon_t \tag{2}$$
$$g(t) = g(t-1) + \beta(t-1) + \eta_t \tag{3}$$
$$\beta(t) = \beta(t-1) + \zeta_t \tag{4}$$

where the $\epsilon_t \sim \boldsymbol{NID}(0, \sigma_\epsilon^2), t = 1, \ldots, T$, $\eta_t \sim \boldsymbol{NID}(0, \sigma_\eta^2)$, and $\zeta t \sim \boldsymbol{NID}(0, \sigma_\zeta^2)$ are distributed independent of one another and white noise disturbance terms with mean zero and variances σ_ϵ^2, σ_η^2, and σ_ζ^2 respectively [21]. [34] proposed trend with stationary drift process to extend local linear trend process by adding a stationary stochastic drift component:

$$g(t) = g(t-1) + \beta(t-1) + \eta_t \tag{5}$$
$$\beta_t = (1 - \varphi_\beta)\bar{\beta} + \varphi_\beta \beta_t + \zeta_t \tag{6}$$

with auto-regressive coefficient $0 < \varphi_\beta \leq 1$. However, there is a drawback with this approach that make such drift processes are difficult to identified. It requires very large data samples.

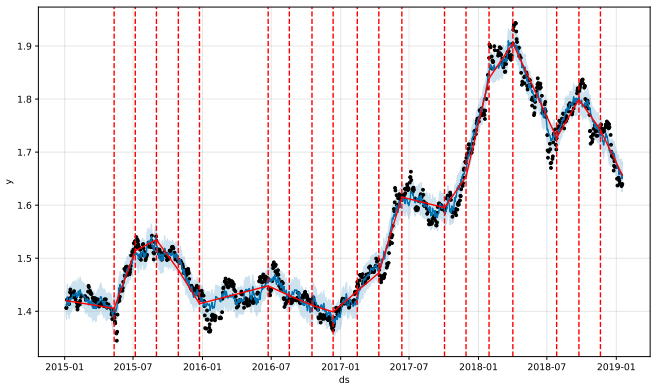

Fig. 1. Fitted daily stock price of Ho Chi Minh City Securities (HCM) stock with structural time series models from January 3rd, 2017 to February 26th, 2019 in log-scale

[44] developed new type of trend models. Accordingly, they suggested that there are two types of trend models: a saturating growth model, and a piecewise linear model. Saturating growth model is characterized by growth rate and limitation of population growth. By applying nonlinear logistic function:

$$g(t) = \frac{C}{1 + e^{-k(t-m)}} \tag{7}$$

with e is the natural logarithm base, m is the value of sigmoid middle point, C is the maximum capacity value, k is growth rate of the curve. From (1-4) point of view, it cannot be captured movement in dynamic world due to nonconstant growth of maximum capacity value and rate of the curve. Hence, to overcome the issues, [44] defined a time-varying of maximum capacity C and growth rate k. Suppose that we explicitly define S changepoints at times $s_j, j = 1, \ldots, S$ and a vector of rate adjustments $\delta \in \mathbb{R}^S$ with δ_j is the change in rate that occurs at time s_j. The saturating growth model is defined as:

$$g(t) = \frac{C(t)}{1 + e^{-(k+a(t)^\intercal \delta)(t-(m+a(t)^\intercal \gamma)))}} \tag{8}$$
$$\tag{9}$$

where

$$\gamma_j = (s_j - m - \sum_{l<j} \gamma_l)(1 - \frac{k + \sum_{l<j} \delta_l}{\sum_{l \leq j} \delta_l}) \tag{10}$$

Maximum capacity $C(t)$ is adopted from external data source. From saturating growth model, we can define piecewise linear model without exhibit saturating growth:

$$g(t) = (k + a(t)^\intercal \delta)t + (m + a(t)^\intercal \gamma) \tag{11}$$

like saturating growth model, k is the growth rate, δ has the rate adjustments, m is offset parameter, and γ_j is set to $-s_j \delta_j$ to make the function continuous.

B. Deep Neural Network

1) Recurrent Neural Network: Despite powerfulness of deep neural networks, traditional neural networks have two drawbacks [35]. Firstly, main assumption of standard neural networks is independence among the samples (data points). On the other words, traditional neural networks can not link current event to previous events to inform later ones due to it stateless preservation. In time series analysis, it is widely accepted that current value depends on past values [4]. It is unacceptable because the independence assumption fails. Secondly, traditional neural networks require fixed-length vector of each sample. Hence, it is critical to develop a new generation of deep neural networks. [41] (p.533) introduced a new learning procedure for neuron networks with back-propagation which can capture internal hidden state to "represent important features of the task domain". On the other words, we feedforward neural networks is lack of cyclical connections. Furthermore, with current development, recurrent neural network can model sequential data with varying length and time dependences. A simple feed forward recurrent neural network is defined [10]:

$$h^{(t)} = \sigma(W^{hx}x^{(t)} + W^{hh}h^{t-1} + b_h) \tag{12}$$

$$\hat{y}^{(t)} = softmax(W^{yh}h^{t} + b_y) \tag{13}$$

where $h^{(t)}$ is hidden state of input data point $^{(t)}$ at time t. Clearly, $h^{(t)}$ is influenced by $h^{(t-1)}$ in the networks previous state. The output $\hat{y}^{(t)}$ at each time t is calculated given the hidden node values $h^{(t)}$ at time t. W^{yh} is weight matrix of input-hidden layer and W^{hh} is the matrix of hidden-to-hidden transition. In most context, $h^{(0)}$ is initialized to zero. [25] suggested that RNN can achieve stability and higher performance by nonzero initialization. By comparison to traditional fully connected feedforward network, a recurrent neural network takes advantage of sharing parameters across the model that helps it learns without separately at each position of sentence or series [16]. Earlier, [31] proposed an almost like [10]. However, context nodes are fed from the output layer instead of from hidden layers. It means that Jordans neural network can take previous predicted output into account to predict current output.

$$h^{(t)} = \sigma(W^{hx}x^{(t)} + W^{yh}\hat{y}^{t-1} + b_h) \tag{14}$$

$$\hat{y}^{(t)} = softmax(W^{yh}h^{t} + b_y) \tag{15}$$

In term of training, there are two steps to train a recurrent neural network. First, the forward propagation creates $\hat{y}$ outputs. After that, loss function value $\mathcal{L}(\hat{y}_k, y_k)$ of the network of each output node k are compute in backpropagation stage. There are many types of loss function to measure distance between the output $\hat{y}$ and the actual value y of classification problems. To minimize the distance, we need to update each of the weights iteratively by applying back-propagation algorithm [41].

The algorithm applies derivative chain rule to calculate the derivative of the loss function $\mathcal{L}$ for each parameter in the network. In addition, weights of neural network are updated by gradient descent algorithm [35]. Hence, gradient of error of a neuron is calculated as:

$$\delta_k = \frac{\partial \mathcal{L}(\hat{y}_k, y_k)}{\partial \hat{y}_k} g'_k(a_k) \tag{16}$$

where $a_k = w\widetilde{a}_k + b$ is input to node k and $\widetilde{a}_k$ is incomming activation ouput of a_k, $g'_k(a_k)$ is activation function for node k. The first term $\frac{\partial \mathcal{L}(\hat{y}_k, y_k)}{\partial \hat{y}_k}$ expresses how fast the cost is changing as a function of estimated output. The second term $g_k(a_k)$ suggests rate of change of g_k activation function at a_k. In vectorized form, we generalize equation (2-10) for any layer l^{th}:

$$\delta^l = \nabla_{\hat{y}} C \odot g\prime\left(a^l\right) \tag{17}$$

In addition, from the δ^l, we can compute the error of the next layer δ^{l+1} as:

$$\delta^l = ((w^{l+1})^T \delta^{t+1}) \odot g\prime(a^l) \tag{18}$$

and error with respect to any weight, bias in the neural network:

$$\frac{\partial C}{\partial w^l} = \hat{y}^{l-1}\delta^l \tag{19}$$

$$\frac{\partial C}{\partial b^l} = \delta^l \tag{20}$$

From the final layer to first hidden layer, for each layer of the neural network, we can apply the back-propagation and compute the error vector δ^l with the chain rule repeatedly to update weight and bias vectors. In term of local minimum optimization, gradient descent is utilized for finding the minimum of cost function by updating weight and bias vectors. It is computed as:

$$w^l \rightarrow \ w^l - \frac{\eta}{m}\sum x\delta^{x,l}(\hat{y}^{x,l-1})^{\mathsf{T}} \tag{21}$$

$$b^l \rightarrow \ b^l - \frac{\eta}{m}\sum_x \delta^{x,l} \tag{22}$$

where m is number of training examples in a given mini-batch with each training example x, η is a step size. In practical, there are many optimizers developed to improve mini-batch gradient descent limitations [40]. For instance, Momentum [39] and Nesterov accelerated gradient [38] were developed to relax navigating ravines problem of stochastic gradient decent. Recurrent neural network is a breakthrough in temporal sequence by adding internal state (memory) in each cell to process sequences of inputs. In term of training, recurrent neural network parameters can be computed and optimized by feed forward propagation and back-propagation. For shallow network with a few hidden layers, the algorithm can be trained effectively. However, with many hidden layers, it is hard to train the network due to vanishing and exploding gradient problem as derivatives become too small (e.g. 0 to 1 for sigmoid activation function) or too large. It only allows the

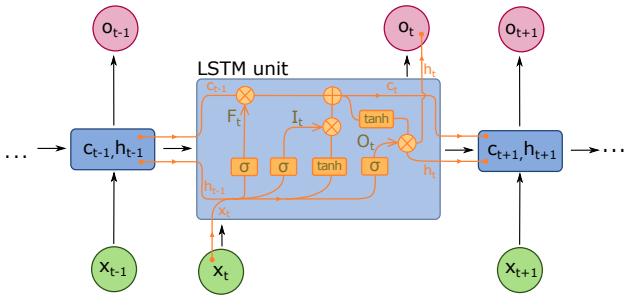

Fig. 2. Long-Short term memory network architecture. Adopted from [20]

network to learn in short-range dependencies and prevents from learning long-range dependencies. As a result, long-short term memory network architecture [29], rectified linear units activation function [36], residual learning framework [26] were introduced to overcome the limitation.

2) Long-Short Term Memory Network: Formally identified by [28] in both theoretical and experimental approaches, with involvement of long-term dependencies data, back propagation algorithm of recurrent neural network is showed that it suffers from insufficient that tends to explode or vanish through time may lead to oscillating weights or unusable model. Not just recurrent neural network, [3] also pointed out that any deep feed-forward neural network with shared weights may have vanishing gradient problem. [29] (p.6) developed a new approach called Long Short-Term Memory (LSTM) to fill these gaps by introducing "input gate unit", "output gate unit", and "memory cell". Accordingly, the purpose of multiplicative input gate unit is to protect memory contents from irrelevant inputs, and multiplicative output gate unit is to protect other units from perturbation by currently irrelevant stored memory contents. On the other words, with the new LSTM architecture, each cell can maintain its state over time, and adjust input or output information. Hence, the new type of neural network architecture is able to capture very long-term temporal dependencies effectively, handle noise and continuous values with unlimited state numbers in principle.

Since introduction, with revolution of computational power, LSTM has been widely adopted and applied for many difficult problems of many fields in practice and academic. This includes language modeling [3]), text classification [50]), language translation [50], speech recognition [18].
From original LSTM proposed by [29]), a significant improvement had been developed by introducing forget gates to reset out-of-dated contents of LSTM memory cells [14]. In addition, in order to achieve higher capability of learning timings, peephole connections that allows gates to look at cell state were added to LSTM neural network. A forward pass LSTM architecture with forget gate and peephole connections is described as [19]:

$$\bar{z}^t = W_z x^t + R_z y^{t-1} + b_z \tag{23}$$
$$z^t = g(\bar{z}^t) \tag{24}$$
$$\bar{i}^t = W_i x^t + R_i y^{t-1} + p_i \odot c^{t-1} + b_i \tag{25}$$
$$i^t = \sigma(\bar{i}^t) \tag{26}$$
$$\bar{f}^t = W_f x^t + R_f y^{t-1} + p_f \odot c^{t-1} + b_f \tag{27}$$
$$f^t = \sigma(\bar{f}^t) \tag{28}$$
$$c^t = z^t \odot \bar{i}^t + c^{t-1} \odot f^t \tag{29}$$
$$\bar{o}^t = W_o x^t + R_o y^{t-1} + p_o \odot c^{t-1} + b_o \tag{30}$$
$$o^t = \sigma(\bar{o}^t) \tag{31}$$
$$y^t = h(c^t) \odot o^t \tag{32}$$

where z^t is block input, i^t is input gate, f^t is forget gate, c^t is memory cell, o^t is output gate, y^t is block output. W_z, W_i, W_f, $W_o \in \Re^{N \times M}$ are input weights; R_z, R_i, R_f, $R_o \in \Re^{N \times M}$ are recurrent weights; $p_i, p_f, p_o \in \Re^N$ are peephole weights; b_z, b_i, b_f, b_o are bias weights; g, σ, h are activation functions.
Like RNN, LSTM is trained with gradient descent as it is a differentiable function estimator [17]. Backpropagation equations of LSTM are detailed:

$$\delta y^t = \Delta^t + R_z^T \delta z^{t+1} + R_i^T \delta i^{t+1} + R_f^T \delta f^{t+1} + R_o^T \delta o^{t+1} \tag{33}$$
$$\delta \bar{o}^t = \delta y^t \odot \ h(c^t) \odot \sigma\prime(\bar{o}^t) \tag{34}$$
$$\delta c^t = \delta y^t \odot o^t \odot h'(c^t) + p_o \odot \delta \bar{o}^t + p_i \odot \delta \bar{i}^{t+1} + p_f \odot \delta \bar{f}^{t+1} + \delta c^{t+1} + f^{t+1} \tag{35}$$
$$\delta \bar{f}^t = \delta c^t \odot c^{t-1} \odot \sigma\prime(\bar{f}^t) \tag{36}$$
$$\delta \bar{i}^t = \ \delta c^t \odot z^t \odot \sigma\prime(\bar{i}^t) \tag{37}$$
$$\delta \bar{z}^t = \delta c^t \odot i^t \odot g\prime(\bar{z}^t) \tag{38}$$
$$\delta x^t = W_z^T \delta \bar{z}^t + W_i^T \delta \bar{i}^t + W_f^T \delta \bar{f}^t + W_o^T \delta \bar{o}^t \tag{39}$$
$$\delta W_* = \sum_{t=0}^{T} \langle \delta *^t, X^t \rangle \tag{40}$$
$$\delta R_* = \sum_{t=0}^{T-1} \langle \delta *^{t+1}, X^t \rangle \tag{41}$$
$$\delta b_* = \sum_{t=0}^{T} \langle \delta *^t \rangle \tag{42}$$
$$\delta p_i = \sum_{t=0}^{T-1} c^t \odot \delta \bar{i}^{t+1} \tag{43}$$
$$\delta p_f = \sum_{t=0}^{T-1} c^t \odot \delta \bar{f}^{t+1} \tag{44}$$
$$\delta p_o = \sum_{t=0}^{T-1} c^t \odot \delta \bar{o}^{t+1} \tag{45}$$

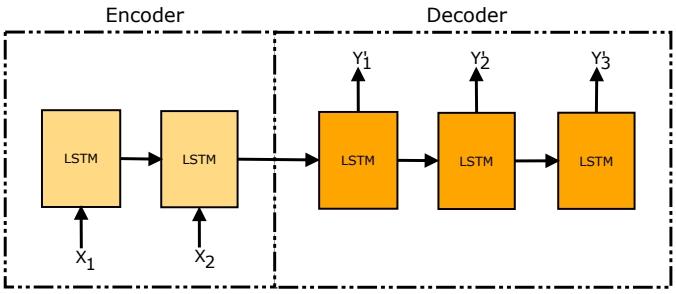

Fig. 3. Encoder-Decoder architecture

where $*$ can be one of $\bar{z}, \bar{i}, \bar{f}, \bar{o}$ and $\langle *1, *2 \rangle$ is outer product of two vectors.

It is worth to note that peephole is not always implemented as forget gate because it simplifies LSTM and reduce computational cost without significantly scarifying performance. For instance, Keras [7] does not support peephole, but CNTK, TensorFlow does support [7], [1]. There have been many variant versions of vanilla LSTM architecture with minor changes. [19] found that vanilla LSTM (with forget gate and peephole) achieve reasonably performance on various datesets. Despite effectiveness of LSTM, there are many efforts to simplify the architecture as LSTM requires huge computational power of hardware. Gated Recurrent Unit (GRU), a variant of LSTM with fewer parameters than LSTM by simplifying forget gate, which introduced by [6] has reasonable accuracy. However, [5] shows that LSTM still significantly outperforms GRU. Hence, [45] is another attempt to save computational costs and maintain performance of models by developing a forget-gate-only version of the LSTM with chrono-initialized biases that achieves lightly higher accuracy.

3) Sequence to sequence model: Sequence to Sequence is a learning model that maps an input sequence from a fixed-sized vector using a LSTM to another LSTM to extract an output sequence. Sequence to Sequence has been widely applied in machine translation [42], video captioning [47], time series classification for human activity recognition [43]. [2] used RNN Encoder-Decoder that contains two recurrent neural networks (or long short-term memory) to represent an input sequence into another sequence of symbols. One the other words, encoder-decoder architecture is used to encode a sequence, decode the encoded sequence, and recreate the sequence. The approach aims to maximize the conditional probability of output sequence given an input sequence.

Encoder neural network transforms an input sequence of variable length $X = x_1, x_2, \ldots, x_T$ into a fixed-length context variable with information of the sequence. RNN is mostly used as an encoder neural network. However, [42] found that LSTM significantly outperformed shallow LSTMs and RNN. As mentioned, RNN and LSTM use previous hidden states $h_1, h_2, \ldots, h_{t-2}, h_{t-1}$ to create current hidden state h_t. Hence, hidden state of an input sequence is defined as:

$$h_t = f(x_t,\ h_{t-1} \tag{46}$$

$$c = k(h_1, h_2, \ldots, h_T) \tag{47}$$

where h_t is hidden state at time t, c is summary hidden state of the whole input sequence, function $f()$ can be RNN, LSTM, GRU network, or an activation function.

With summary hidden state c, given a target output $Y = y_1,\ y_2, \ldots,\ y_{T\prime}$, instead of computing $\mathbb{P}(Y|X)$ directly, decoder neural network computes conditional probability of y_t using previous information and summary hidden state. It is formally described as:

$$\mathbb{P}\left(y_1, \ldots,\ y_{T\prime}|x_1, \ldots,\ x_T\right) = \prod_{t\prime=1}^{T\prime} \mathbb{P}(y_{t\prime}|c,\ y_1, \ldots,\ y_{t\prime-1}) \tag{48}$$

The trained sequence to sequence model can be used to generate a sequence give an input sequence. In machine translation, reverse the order of the words of the input sequence is necessary because it is easy for optimizer (e.g stochastic gradient decent) to "establish communication between the input and the output" [42] (p.3). For the sake of nature, time series prediction problems always have desired order as input and output is straightforward sequence.

III. Empirical Results

A. Data

In this study, for liquidity and fairness of trading, we use daily price data of 21 most traded stocks that is listed on from VN-Index of Ho Chi Minh Stock Exchange and HNX-Index of Hanoi Stock Exchange (Vietnam) from 05 January 2015 to 19 January 2019. It total 1010 data points. We predict 9-type of window size. It varies from 5 to 45 with 5-step ahead.

B. Data Pre-processing

Beyond algorithm improvement and parameter tuning, an approach to improve the accuracy of machine learning model is apply data pre-processing techniques. For instances, these techniques are impute missing values, encode categorical data, detect outliers, transform data, and scaling data. In this work, we perform logarithm and Box-Cox transform to transform the input dataset. Rationally, the idea behind the logarithm transformation is to turn probabilistic distribution of raw input data from skewed data into approximately normal. Hence, prediction performance is improved dramatically [48]. However, in some circumstances, the logarithm technique does not generate new data with less variable or more normal. In contrast, it may lead to be more variable and more skewed [12]. Thus, it is recommended that transformation techniques must be applied very cautiously.

Output data of the transform stage is passed to data scaler to be normalized. There are many types of scaling method (e.g. maximum absolute value, given range of feature). We use min-max scaler by scaling the input feature to a range of $[0, 1]$. It ensures the large input value do not overwhelm smaller value inputs, then helps to stablize neural networks [32].

C. Detail Results

1) Structural Time Series: The aim of this step is to create baseline models for evaluating prediction quality of structural time series and sequence to sequence models with our proposed model. Mean square error was calculated to measure performance of each out-of-sample forecast.

Input : List of price of V stocks in logarithmic form LP, Box-Cox form BC, steps ahead W
Output: Trend prediction T, log-price prediction $\hat{Y}$, combination result dataset D of stocks
for v *in* V **do**
  if *Length of* $LP > 500$ **then**
    Initialize Prophet model P;
    if $W > 0$ **then**
      $T \leftarrow P(LP, BC)$;
      $D \leftarrow [LP, BC, T]$;
      return D;
    else
      `// out-sample prediction w = 0;`
      $\hat{Y} \leftarrow P(LP)$;
      return $\hat{Y}$;
    end
  end
end

Algorithm 1: Algorithm of structural time series analysis with Facebook Prophet library

We develop structural time series models as a baseline model. For this task, we choose Prophet package which is developed by Facebook for Python programming language [44]. In this model, data input is a transformed price of stocks in logarithm. In terms of parameter tuning, we almost use default settings except adding monthly, quarterly, and yearly with Fourier orders. We initialize 20, 30, 30 for Fourier orders of monthly, quarterly, and yearly respectively. As it is required future data would have to be known to perform prediction if we use Box-Cox transformation as an extra regressor in structural time series models, we omit the transformation procedure [11]. Without extra regressor, the model can generate prediction of 21 selected tickers from 5 to 45 with 5-step incrementation window size.

2) Sequence to Sequence Model: Regarding to baseline models, we also develop a Sequence to Sequence with LSTM architecture. We use Keras with Tensorflow backend to create Encoder-Decoder model to solve the sequence to sequence problem [7], [1]. To benefit from the efficiency of parallel computation for training deep learning neural network, we train the model on virtual machine with GPU on Google Cloud Platform.

Input : Arrays of train data X^*, steps ahead W
Output: Log-price prediction O
for x *in* X^* **do**
  Pre-train encoder prediction network En and decoder prediction network De with x;
  Last element k of array x;
  States $H, C \leftarrow En(k)$;
  for $w = 1$ *to* W **do**
    Output $O \leftarrow De([H, c])$
  end
end

Algorithm 2: Algorithm of Sequence to Sequence

The implementation is straightforward. First, we use scaled data of Box-Cox and logarithm transforms as input data. Furthermore, we use logarithm transform data as target data. A main advantage of Sequence to Sequence with LSTM over structural time series models is that it can dynamically perform prediction multiple time steps without requiring extra data. In terms of accuracy, we found that result of deeper LSTM model does not outperform shallow one. Hence We used LSTM with single hidden layer, with 64 cells and rectified linear unit activation function. To prevent over-fitting, we apply both L2 regularization and dropout. We use 0.0001 for regularization parameter lambda, rate of dropout is 0.001 as recommended [27].

3) Sequence to Sequence with Structural Time Series Models: In this step, we combine both sequence to sequence model and structural time series models. Specifically, we use output dataset D (with $W = 0$) from Algorithm 1 as train data for Algorithm 2. On the other words, we combine trend component of structural time series models with price of stock in Box-Cox and logarithm forms. Parameters of these models are defined exactly same as aforementioned baseline models. We found that results are improved dramatically.

4) Results Analysis: Structural time series models was used to generate a set of out-of-sample forecast in multiple window time steps in log-scale (see Table I). In terms of prediction error, the result show that $MSE = 0.087787$ (CTG at 45 time steps ahead) is highest, $MSE = 0.003501$ (SSI at 5 time steps ahead) is lowest. Likewise, results from Sequence to Sequence model (see Table II) and Sequence to Sequence with Structural Time Series Models (see Table III) show that $MSE = 0.231800$ (PNJ at 45 time steps ahead) and $MSE = 0.046146$ (ACB at 45 time steps ahead) are highest, $MSE = 0.000068$ (CII at 5 time steps ahead) and $MSE = 0.000006$ (CII at 5 time steps ahead) are lowest.

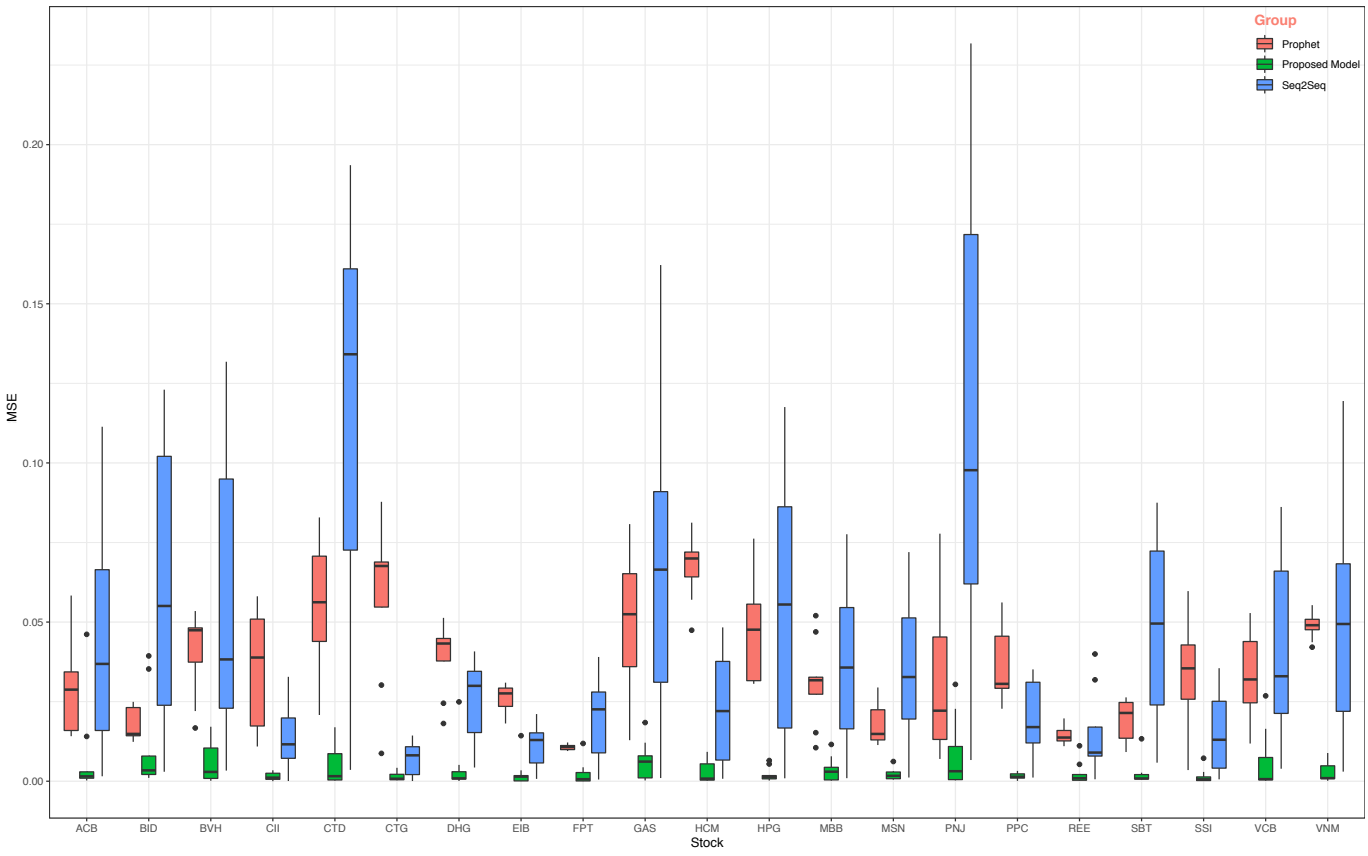

Fig. 4. Error of out-of-sample forecasts in log-scale

From univariate time series analysis perspective, We found that structural time series models of Facebook Prophet generate stable and high quality out-of-sample prediction without requiring advanced techniques or data assumptions. In addition, we also found that it even achieve higher accuracy in-sample fitted data when we add an extra regressor to structural time series models. Unfortunately, we cannot create out-sample prediction with extra regressor. In contrast to structural time series models, Sequence to Sequence model with LSTM neural network cannot create stable out-of-sample prediction. As Figure 4 point out, in some cases, Sequence to Sequence model captures movement of stocks to generate high accuracy prediction with lower error than structural time series models. However, the model cannot constantly capture movement of stocks in some other cases. In terms of computational performance, Sequence to Sequence model also takes more time for training and predicting than structural time series models. It leads to a gap to leveraging the state-of-the-art technique for time series prediction. Fortunately, results from Table III suggest that we can fill gaps of structural time series models and Sequence to Sequence model by adding output from structural time series models to Sequence to Sequence model. Figure 4 show that the model is stable and prediction error of proposed model is almost always lowest among in three models.

In terms of benchmark limitation, there are some drawbacks in this benchmark. On the one hand, it is lack of residual analysis for each prediction. We only compute Mean Square Error (MSE) for performance comparison. The evaluated results are not consistent enough to be fully accurate as their some outlier points as figure 4 point out. On the other hand, although the results are clear and useful when we use MSE as a indicator for forecasting accuracy evaluation, these forecasting evaluation criteria cannot be discriminated between forecasting models when errors of the forecast data are very close to each other. Thus, the Chong and Hendry

TABLE I
MEAN SQUARE ERROR OF STRUCTURAL TIME SERIES MODEL FORECAST FROM 5 TO 45 WINDOW TIME STEPS AHEAD IN LOG-SCALE

	5	10	15	20	25	30	35	40	45
ACB	0.014721	0.014147	0.015933	0.025314	0.028773	0.029434	0.034342	0.048010	0.058335
BID	0.024936	0.023140	0.014819	0.012375	0.014865	0.013050	0.014286	0.021603	0.023775
BVH	0.016722	0.022044	0.037414	0.046589	0.047791	0.047427	0.048169	0.052609	0.053467
CII	0.015892	0.010909	0.017354	0.025683	0.038861	0.049060	0.050928	0.056750	0.058097
CTD	0.020805	0.038501	0.043919	0.048860	0.056231	0.062125	0.070704	0.081036	0.082875
CTG	0.008746	0.030208	0.054718	0.065659	0.067614	0.068867	0.068824	0.084200	***0.087787***
DHG	0.024497	0.018136	0.037780	0.043572	0.043265	0.042221	0.044850	0.047424	0.051320
EIB	0.027591	0.027747	0.023507	0.023673	0.029305	0.030964	0.029256	0.021934	0.018138
FPT	0.012170	0.009933	0.009485	0.009722	0.010321	0.011376	0.011212	0.010736	0.010837
GAS	0.012885	0.016930	0.035988	0.047679	0.052466	0.057323	0.065196	0.078481	0.080781
HCM	0.047436	0.057021	0.064199	0.071993	0.069983	0.069544	0.070894	0.080751	0.081232
HPG	0.031256	0.030599	0.031586	0.044547	0.047608	0.050945	0.055629	0.067218	0.076217
MBB	0.015255	0.010532	0.027337	0.032361	0.031718	0.031117	0.032678	0.046915	0.052013
MSN	0.029451	0.022273	0.014707	0.011364	0.012848	0.012974	0.014851	0.022448	0.026552
PNJ	0.006993	0.007920	0.013121	0.016369	0.022160	0.028659	0.045313	0.065240	0.077773
PPC	0.030586	0.030464	0.024987	0.022773	0.029191	0.037727	0.045555	0.054872	0.056164
REE	0.019733	0.016948	0.015939	0.014662	0.012662	0.013016	0.011561	0.011009	0.013720
SBT	0.009171	0.009984	0.020123	0.023605	0.026334	0.026340	0.024738	0.013508	0.021442
SSI	***0.003501***	0.010974	0.025759	0.033560	0.035462	0.038449	0.042812	0.058004	0.059730
VCB	0.043887	0.052832	0.048772	0.038443	0.031976	0.029193	0.024631	0.011838	0.017969
VNM	0.049289	0.055314	0.049015	0.043668	0.047589	0.050864	0.054577	0.048458	0.042112

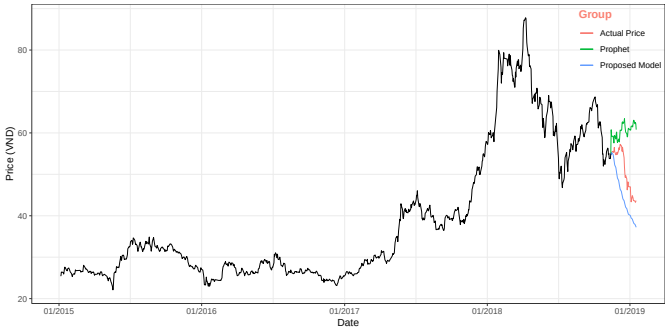

Fig. 5. 45-day forecast of HCM

encompassing test for nested models [8] should be carried out to evaluate the statistical significance of the forecasting models. However, there is no package in Python supporting the test at this time, the test was not carried out to conduct appropriated benchmark in terms of statistics. In addition, Diebold-Mariano (DM) test for comparing predictive accuracy (not for comparing models) cannot be applied as it only works for non-nested models [46], [9].

Overall, in same window size, the combination of structural time series models and Sequence to Sequence model are always achieve high performance than pure structural time series models and Sequence to Sequence model. However, in some cases, the hybrid model cannot capture movement of stock when market is highly volatile.

IV. CONCLUSION AND FUTURE WORK

In this work, we generally discussed a set of procedures to model and predict price of stocks in Vietnam stock market with structural time series models and Sequence to Sequence model and the combination of these models. We used output of models to compare accuracy performance of each model. We found that by using output from statistical method (i.e. structural time series models) as input features for deep learning method (i.e. Sequence to Sequence model), we can overcome limitations of each model and generate forecast with high accuracy. Furthermore, deep learning is a powerful approach to address time series problems. However, without feature engineering, deep learning generates prediction lower accuracy than structural time series models.

In future work, we will improve that model to achieve real-time prediction to apply for quantitative trading. In addition, we believe that Generative Adversarial Networks (GAN) is a promising approach to apply.

REFERENCES

[1] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A System for Large-Scale Machine Learning, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. sep 2014.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 1994.

TABLE II
MEAN SQUARE ERROR OF SEQUENCE TO SEQUENCE MODEL FORECAST FROM 5 TO 45 WINDOW TIME STEPS AHEAD IN LOG-SCALE

	5	10	15	20	25	30	35	40	45
ACB	0.001549	0.010649	0.015936	0.036856	0.066454	0.088031	0.065661	0.023986	0.111370
BID	0.002975	0.023858	0.022093	0.055056	0.045212	0.055175	0.123015	0.111880	0.102072
BVH	0.003339	0.007065	0.022930	0.037389	0.038270	0.048385	0.094943	0.131791	0.118588
CII	***0.000068***	0.000949	0.007197	0.011593	0.008407	0.015061	0.032799	0.024295	0.019871
CTD	0.003564	0.031081	0.093107	0.072614	0.134139	0.160979	0.193552	0.157608	0.180879
CTG	0.000067	0.001404	0.002087	0.008134	0.003509	0.010062	0.014351	0.011942	0.010831
DHG	0.004307	0.015290	0.027839	0.005491	0.031615	0.040781	0.029965	0.034531	0.036360
EIB	0.000734	0.004151	0.005739	0.015192	0.009920	0.018238	0.014715	0.021097	0.012947
FPT	0.000538	0.004052	0.008909	0.011313	0.022583	0.027503	0.039039	0.028022	0.028253
GAS	0.001004	0.021844	0.031094	0.073882	0.090982	0.066480	0.057572	0.106243	0.162174
HCM	0.000771	0.005351	0.012451	0.006651	0.022035	0.030805	0.048325	0.047193	0.037641
HPG	0.000912	0.016714	0.055040	0.013062	0.073548	0.086221	0.099066	0.055516	0.117556
MBB	0.000955	0.008257	0.016460	0.021182	0.035717	0.054560	0.038501	0.064305	0.077589
MSN	0.001165	0.008558	0.019539	0.023428	0.032733	0.051309	0.039536	0.061719	0.071979
PNJ	0.009910	0.006657	0.061987	0.097721	0.094561	0.124379	0.171758	0.173592	***0.231800***
PPC	0.001150	0.002104	0.014166	0.012028	0.017004	0.027315	0.031092	0.031779	0.035133
REE	0.000612	0.005939	0.008998	0.008640	0.012904	0.017005	0.007903	0.031858	0.039960
SBT	0.005828	0.017245	0.031245	0.023963	0.067473	0.087509	0.049534	0.072302	0.079048
SSI	0.000589	0.004103	0.003022	0.011251	0.014304	0.013024	0.026790	0.025110	0.035506
VCB	0.003926	0.009320	0.032977	0.021303	0.023619	0.066008	0.039739	0.086166	0.084254
VNM	0.003010	0.021958	0.021868	0.043893	0.065320	0.049399	0.068308	0.087976	0.119454

[4] GEP Box, GM Jenkins, GC Reinsel, and GM Ljung. Time series analysis: forecasting and control. 2015.

[5] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. Massive Exploration of Neural Machine Translation Architectures. mar 2017.

[6] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. jun 2014.

[7] François Chollet. Keras, 2015.

[8] Yock Y. Chong and David F. Hendry. Econometric Evaluation of Linear Macro-Economic Models. *The Review of Economic Studies*, 53(4):671, aug 1986.

[9] Francis X. Diebold. Comparing Predictive Accuracy, Twenty Years Later: A Personal Perspective on the Use and Abuse of DieboldMariano Tests. *Journal of Business & Economic Statistics*, 33(1):1–1, jan 2015.

[10] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 1990.

[11] Facebook. Prophet Notebook.

[12] C. Feng, H. Wang, N Lu, T. Chen, H. He, Ying Lu, and X. M. Tu. Log-transformation and its implications for data analysis. *Shanghai archives of psychiatry*, 2014.

[13] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. 2017.

[14] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 2000.

[15] Daniel Giamouridis and Ioannis D. Vrontos. Hedge fund portfolio construction: A comparison of static and dynamic approaches. *Journal of Banking & Finance*, 31(1):199–217, jan 2007.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. 2016.

[17] Alex Graves. Supervised Sequence Labelling. Springer, Berlin, Heidelberg, 2012.

[18] Alex Graves, Navdeep Jaitly, and Abdel-rahman Mohamed. Hybrid speech recognition with Deep Bidirectional LSTM. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 273–278. IEEE, dec 2013.

[19] Klaus Greff, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink, and Jurgen Schmidhuber. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 2017.

[20] Mark Harmon and Diego Klabjan. Dynamic Prediction Length for Time Series with Sequence to Sequence Networks. jul 2018.

[21] P. J. Harrison and C. F. Stevens. Bayesian Forecasting, 1976.

[22] A. C. Harvey. Trends and cycles in macroeconomic time series. *Journal of Business and Economic Statistics*, 1985.

[23] A. C. Harvey and S. Peters. Estimation procedures for structural time series models. *Journal of Forecasting*, 1990.

[24] A. C. Harvey and P. H.J. Todd. Forecasting economic time series with structural and box-jenkins models: A case study. *Journal of Business and Economic Statistics*, 1983.

[25] S Haykin, JC Principe, TJ Sejnowski, and J Mcwhirter. Modeling Large Dynamical Systems with Dynamical Consistent Neural Networks. *ieeexplore.ieee.org*.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. dec 2015.

[27] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. jul 2012.

[28] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. *Master's thesis, Institut für Informatik, Technische Universität, Munchen*, 1991.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, nov 1997.

[30] JT Jalles. Structural time series models and the Kalman Filter: a concise review. 2009.

[31] Michael I Jordan. Serial order: A parallel distributed processing approach. *University of California, San Diego. Institute for Cognitive Science*, 1986.

[32] Kyoung-jae Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1-2):307–319, sep 2003.

[33] Genshiro Kitagawa and Will Gersch. A smoothness priorsstate space modeling of time series with trend and seasonality. *Journal of the American Statistical Association*, 1984.

[34] S Koopman and M Ooms. Forecasting economic time series using unobserved components time series models. 2011.

[35] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A Critical Review of Recurrent Neural Networks for Sequence Learning. may 2015.

[36] Vinod Nair and Geoffrey Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[37] Charles R. Nelson and Heejoon Kang. Pitfalls in the Use of Time as an Explanatory Variable in Regression. *Journal of Business & Economic Statistics*, 2(1):73–82, jan 1984.

TABLE III
MEAN SQUARE ERROR OF PROPOSED MODEL FORECAST FROM 5 TO 45 WINDOW TIME STEPS AHEAD IN LOG-SCALE

	5	10	15	20	25	30	35	40	45
ACB	0.000243	0.001528	0.000953	0.002946	0.000294	0.001339	0.002150	0.014070	***0.046146***
BID	0.002119	0.001047	0.003420	0.002281	0.007414	0.007904	0.039367	0.035262	0.002000
BVH	0.000145	0.007243	0.000876	0.000427	0.011959	0.002539	0.010426	0.017116	0.002917
CII	***0.000006***	0.002513	0.000787	0.001118	0.001933	0.000599	0.000063	0.002505	0.003436
CTD	0.001580	0.000395	0.000173	0.000329	0.008648	0.000776	0.016954	0.015527	0.002938
CTG	0.001237	0.000776	0.000641	0.002322	0.000283	0.000287	0.000458	0.002171	0.004216
DHG	0.000081	0.002963	0.000692	0.000894	0.005131	0.002980	0.000729	0.000448	0.024931
EIB	0.000022	0.001645	0.000096	0.001451	0.000103	0.001309	0.003419	0.001766	0.014310
FPT	0.000018	0.000009	0.002727	0.000098	0.000629	0.000087	0.004376	0.011860	0.001626
GAS	0.001049	0.000242	0.006162	0.001701	0.007973	0.000451	0.012106	0.007322	0.018421
HCM	0.000120	0.000814	0.000365	0.006616	0.000347	0.000154	0.005450	0.009231	0.003637
HPG	0.001480	0.000214	0.000307	0.000747	0.000886	0.001769	0.005409	0.006507	0.001292
MBB	0.000151	0.000234	0.002842	0.000414	0.003575	0.004369	0.011552	0.002987	0.007829
MSN	0.002896	0.001696	0.000507	0.001222	0.003149	0.006190	0.000697	0.002198	0.000785
PNJ	0.000267	0.003331	0.000503	0.000599	0.010909	0.003138	0.030432	0.022743	0.000275
PPC	0.000106	0.001729	0.001398	0.002341	0.001320	0.000454	0.003241	0.002314	0.000987
REE	0.000349	0.001104	0.000185	0.000304	0.000936	0.000123	0.002099	0.005302	0.011126
SBT	0.000863	0.002671	0.001414	0.000694	0.000526	0.000598	0.000768	0.013324	0.002088
SSI	0.000214	0.000316	0.000056	0.000131	0.001353	0.002919	0.000708	0.000588	0.007224
VCB	0.000439	0.000612	0.000192	0.000652	0.016464	0.007455	0.000140	0.002109	0.026819
VNM	0.000237	0.000130	0.008860	0.005173	0.002549	0.000879	0.000835	0.004840	0.000963

[38] Yurii Nesterov. *Introductory Lectures on Convex Optimization*, volume 87 of *Applied Optimization*. Springer US, Boston, MA, 2004.

[39] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, jan 1999.

[40] Sebastian Ruder. An overview of gradient descent optimization algorithms. sep 2016.

[41] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 1986.

[42] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks, 2014.

[43] Yujin Tang, Jianfeng Xu, Kazunori Matsumoto, and Chihiro Ono. Sequence-To-Sequence Model with Attention for Time Series Classification. In *IEEE International Conference on Data Mining Workshops, ICDMW*, 2017.

[44] Sean J Taylor and Benjamin Letham. Forecasting at Scale. 2017.

[45] Jos van der Westhuizen and Joan Lasenby. The unreasonable effectiveness of the forget gate. apr 2018.

[46] Marián Vávra. On a Bootstrap-Based Diebold-Mariano Test for Forecast Evaluations. Technical report, 2015.

[47] Subhashini Venugopalan, Marcus Rohrbach, Jeff Donahue, Raymond Mooney, Trevor Darrell, and Kate Saenko. Sequence to Sequence – Video to Text. may 2015.

[48] Y.Y. Yang, D.A. Linkens, and M. Mahfouf. Nonlinear Data Transfonnation to Improve Flow Stress Prediction Using Neural Networks. *IFAC Proceedings Volumes*, 37(15):371–376, sep 2004.

[49] Yong Zhang, Xiaobin Tan, Hongsheng Xi, and Xin Zhao. Real-time risk management based on time series analysis. In *2008 7th World Congress on Intelligent Control and Automation*, pages 2518–2523. IEEE, 2008.

[50] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. A C-LSTM Neural Network for Text Classification. nov 2015.

[51] Xingyu Zhou, Zhisong Pan, Guyu Hu, Siqi Tang, and Cheng Zhao. Stock Market Prediction on High-Frequency Data Using Generative Adversarial Nets. *Mathematical Problems in Engineering*, 2018:1–11, apr 2018.