Proceedings of Machine Learning Research 95:454-469, 2018 ACML 2018

Stock Price Prediction Using Attention-based Multi-Input LSTM

Hao Li APPLEJACK@SJTU.EDU.CN
Yanyan Shen SHENYY@SJTU.EDU.CN
Yanmin Zhu YZHU@SJTU.EDU.CN
Department of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai, China

Editors: Jun Zhu and Ichiro Takeuchi

Abstract

Stock price prediction has always been a hot but challenging task due to the complexity and randomness in stock market. Investors and researchers usually derive a great number of factors from original data such as historical stock price, company profit, or textual data collected from social media. Normally these factors are then fed into models like linear regression, SVM or neural networks to make a prediction. Even though the number of factors are considerable, most of them have relatively weak correlations with future stock price. During training process, these factors not only result in additional computation but sometimes even be harmful to the performance of prediction. In this paper, we propose a novel multi-input LSTM model which is capable of extracting valuable information from low-correlated factors and discarding their harmful noise by employing extra input gates controlled by the convincing factors called *mainstream*. We also introduce several new factors including the prices of other related stocks to improve the prediction accuracy. The experimental results on the stock data from China stock market demonstrate the effectiveness of the proposed approach compared with the state-of-the-art methods.

1. Introduction

Using historical stock data to predict future stock price has been a hot topic for decades. Since stock price is typically influenced by various factors, it is common to derive a large number of factors from both historical stock price and other information such as financial statement and textual data from social media, etc. In our empirical experiments, we found an issue that including some of the factors by concatenation (dimensional expansion of input vector) not only makes no contribution to future stock price prediction but can even be harmful to the prediction accuracy. This usually happens when the correlation between the factor and prediction target is relatively weak. If this problem remains unsettled, the extra information from the additional factors can hardly offset the noise brought by them.

In general, we have two ways to solve this problem, either to identify and select useful factors, or to adaptively weaken the effects of useless factors. There are many works that tried to solve this problem by feature selection or feature extraction through various kinds of algorithms such as PCA Singh and Srivastava (2017), Restricted Boltzmann Machine Chong

Table 1: Correlation Coefficient between prediction target and other factors (10-day average).

	Self	Positive	Negative	Index	Noise
Correlation	0.9877	0.3522	-0.0939	0.2322	0.0061

et al. (2017), Genetic Algorithm Tsai and Hsiao (2010). However, these approaches treated every factor equally that they do not distinguish important factors from less important ones during feature selection. In particular, none of these works consider to use *mainstream* (i.e., important factors) to adaptively decide which other factors should be selected. In fact, *mainstream* has decisive influence on the prediction result hence it can usually be discriminated through a relatively simple process (e.g., by computing and comparing the correlation coefficient). Table 1 briefly summarizes the correlation coefficient between our prediction target (opening price of the target stock in the next day) and factors derived from historical opening prices of various stocks (e.g., target stock itself, positive related stocks, negative related stocks, stock index, etc.). These factors will be formally defined in Section 3. We can see that historical opening price of the target stock itself is much more related to the prediction target (0.9877) than opening price of other stocks. In this case, we define it as *mainstream*. More formally, we could easily define a threshold of correlation coefficient to separate mainstream with other factors. Using the *mainstream* to adaptively select other factors is a way to make better use of the most convincing information, which we believe would make the selection of secondary (or auxiliary) factors more comprehensively. As a result, the noise of the auxiliary factors can be significantly depressed. This inspires us to design additional structures (e.g., input gates) to let *mainstream* control auxiliary factors. Besides, to the best of our knowledge, little attention has been paid to leveraging information from other stocks in the same market for prediction. For example, from the stocks belong to the same industry, which would more likely to share the same tendency of fluctuation. As a prove we can see stock price of other stocks could be useful factors (2-4 columns) since they are still much better than Gaussian noise (0.0061) in terms of the correlation coefficient.

Stock price prediction is a special kind of time series prediction which is recently addressed by the recurrent neural networks (RNNs). However, the currently state-of-the-art long short-term memory (LSTM) Hochreiter and Schmidhuber (1997) also suffers from the aforementioned problem: it may be harmful when useless factors are simply concatenated into the input vector of LSTM. In this paper, we propose a novel multi-input LSTM unit to distinguish *mainstream* and auxiliary factors. Specifically, we design input gates which are controlled by *mainstream* and previous hidden states for *mainstream* and auxiliary factors. By filtering the data from both *mainstream* and auxiliary factors, these input gates generate memory cell inputs which will be merged before updating cell states. Regarding the importance gap among these cell inputs, it is necessary to assign different weights on different cell inputs and merge them into one memory cell input via weighted sum. We apply the attention mechanism Xu et al. (2015a) to assign different weights. For example, prior work Choi et al. (2017) calculated the feature expression of medical codes using weighted sum and the weights are assigned through the attention module. Therefore, we can employ

the attention mechanism to compute weights for the combination of different cell inputs based on cell inputs and previous cell states. The attention weights are learned adaptively through the training process.

The major contributions of this paper are summarized as follows:

- We discover the potential usage of related stocks and employ the stock price of related stocks to predict the future price of target stock.
- We propose a novel MI-LSTM model which enables *mainstream* to decide the usage of other factors and employs a dual-stage attention mechanism on different memory cell inputs and hidden states of different time steps to improve the prediction accuracy.
- We compare our proposed model with various state-of-the-art models to evaluate the effectiveness of MI-LSTM on the stock data from Chinese stock market. MI-LSTM achieves an improvement of 9.96% compared with LSTM in terms of mean square error (**MSE**).

The rest of this paper is organized as follows. In Section 2, we introduce the related works. Section 3 presents the definition of the problem, traditional LSTM and the factors we use. Details of our attention-based MI-LSTM model are provided in Section 4. The experimental results are presented in Section 5. Section 6 draws the conclusion of this paper.

2. Related Work

2.1. Stock Price Prediction

Stock price prediction has always been a challenging task because of the volatility in stock-market according to ADAM et al. (2016). Various attempts have been made using different kinds of traditional machine learning algorithms. For example, Chen and Hao (2017); Luo et al. (2017) applied WSVM, which is a kind of variant of the commonly used support vector machine but is able to assign weights on different features or samples. In the meantime, autoregressive (AR) model is another widely used method for time series prediction. Adebiyi et al. (2014); Xiao et al. (2014) used both autoregressive integrated moving average (ARIMA) and NN model to predict stock market in order to compare the performance of different models. In Chang and Lee (2017), Markov decision process and genetic algorithms are implemented to design stock market strategies. The aforementioned works tend to focus on deriving factors from original numerical data. In order to obtain extra information, Jin et al. (2017); Nguyen et al. (2015); Bordino et al. (2014); Ming et al. (2014) turned textual data collected from social media into vectors and use them as addition features. Apart from the efforts on traditional machine learning methods, neural networks have played more and more important roles in recent years.

2.2. Neural Networks

Neural networks, known by their complicated and non-linear nature, have achieved great success in various domains. To tackle time series problems, recurrent neural networks

(RNNs) which receive the output of hidden layer of the previous time step along with current input have been widely used. Because of their recurrent structure, RNNs use a special backpropagation through time (BPTT) algorithm Werbos (1990) to update cell weights. In financial domain, Rather et al. (2015) used RNN and genetic algorithms to calculate stock returns. However, Bengio et al. (1994) pointed out that traditional RNNs have great difficulty to capture long-term dependency because of vanishing gradients. Thus the performance of RNNs is restricted until Hochreiter and Schmidhuber (1997) first proposed long short-term memory (LSTM) units which store long-term information in additional cell states and use gates to control the information flow in or flow out. Since then, traditional RNNs are commonly replaced by LSTM or gated recurrent unit (GRU), which is another approch to deal with long-term dependency. Based on LSTM, Zhang et al. (2017) used discrete fourier transform (DFT) to decompose the output of hidden units to capture multi-frequency patterns. Besides RNNs, Ding et al. (2015) used convolutional neural networks (CNNs) to model both short-term and long-term dependencies.

2.3. Attention Mechanism

Assigning attention weights on neural networks has achieved great success in various machine learning tasks. In machine translation, the goal is to translate a given sentence to a new sentence in another language. It it a matter of course that different words in original sentence should have different importance while generating different words in target sentence. Since the great success that Bahdanau et al. (2014) used attention-based RNNs to assign attention weights on different hidden outputs corresponding to different words, employing attention weights has become epidemic. In electronic health records (EHRs) domain, Ma et al. (2017) introduced three kinds of temporal attention on different time steps. As for recommendation systems, Wang et al. (2017) assigned attention among multiple neural nets and Chen et al. (2017a) used knowledge-based attention to utilize field knowledge. Besides, attention mechanism has also made progress in QA systems Chen et al. (2017b) and image caption generation Xu et al. (2015b). Finally, a recent work in stock price prediction Qin et al. (2017) incorporated both encoder-decoder and attention mechanism to propose a kind of dual-stage attention-based RNN namely DA-RNN. We would also employ DA-RNN as one of our comparative methods.

3. Preliminaries

Problem Definition: The goal of this work is to predict the opening price of the next day given historical data. We define the historical series of the target stock as $\mathbf{Y} = (y_1, y_2, ..., y_T)^\top \in \mathbb{R}^T$ where T represents time window size and y_t the stock price at time t. Similarly related stock series (auxiliary factors) would be represented by $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_T)^\top \in \mathbb{R}^{T \times D}$ where D specifies the number of related stocks. $\mathbf{X}_t \in \mathbb{R}^D$ is the stock prices of all D related stocks at time t and $\mathbf{X}^d \in \mathbb{R}^T$ is the stock prices of the d^{th} stock in time window T. Thus the prediction target y_{T+1} could be defined as follows:

$$y_{T+1} = F(y_1, y_2, ..., y_T, \mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_T) \tag{1}$$

$F(\cdot)$ is the function we are aiming to learn.

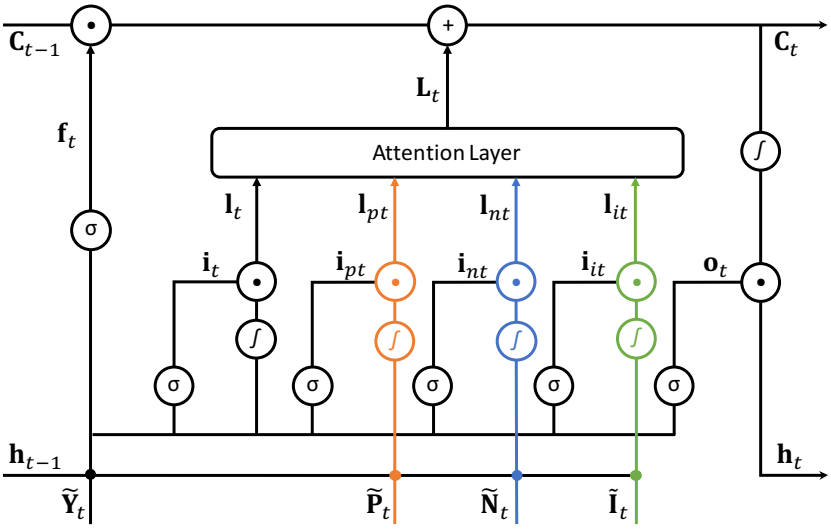

Figure 1: MI-LSTM. The intersection between $\mathbf{h}_{t-1}$ and $\tilde{\mathbf{Y}}_t, \tilde{\mathbf{P}}_t, \tilde{\mathbf{N}}_t, \tilde{\mathbf{I}}_t$ represents concatenate operation. Tokens like "$\cdot$", "σ", "$\int$", "$+$" represent element-wise operations of multiplication, sigmoid, tanh, addition, respectively.

Long Short Term Memory (LSTM) model: The LSTM has been one of the most popular models for time series prediction in recent years. Its output at time t depends on the input at time t and its previous hidden states. Formally, given a time series $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$ with $\mathbf{x}_t \in \mathbb{R}^m$, a LSTM unit updates as follows:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_f) \tag{2}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_i) \tag{3}$$
$$\tilde{\mathbf{C}}_t = tanh(\mathbf{W}_c[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_c) \tag{4}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}; \mathbf{x}_t] + \mathbf{b}_o) \tag{5}$$
$$\mathbf{C}_t = \mathbf{C}_{t-1} * \mathbf{f}_t + \tilde{\mathbf{C}}_t * \mathbf{i}_t \tag{6}$$
$$\mathbf{h}_t = tanh(\mathbf{C}_t) * \mathbf{o}_t \tag{7}$$

where $\mathbf{h}_t, \mathbf{h}_{t-1} \in \mathbb{R}^q$ are the hidden states at time t and $t-1$, q is the dimension of the hidden state. $\mathbf{W}_f, \mathbf{W}_i, \mathbf{W}_c, \mathbf{W}_o \in \mathbb{R}^{q \times (q+m)}$ are the weight matrices and $\mathbf{b}_f, \mathbf{b}_i, \mathbf{b}_c, \mathbf{b}_o \in \mathbb{R}^q$ are bias vectors. σ represents sigmoid function and operator $*$ is element-wise multiplication. For convenience, we would like to use a single non-linear function f_1 to represent a LSTM layer described using Eqn. (2) - (7):

$$\mathbf{h}_t = f_1(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{8}$$

Given input $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_T) \in \mathbb{R}^{T \times m}$ where $\mathbf{a}_t \in \mathbb{R}^m$, we define an q-dimensional LSTM layer:

$$\mathbf{A}' = LSTM(\mathbf{A}) \tag{9}$$

where $\mathbf{A}' = (\mathbf{a}'_1, \mathbf{a}'_2, ..., \mathbf{a}'_T) \in \mathbb{R}^{T \times q}$, $\mathbf{a}'_t \in \mathbb{R}^q$ is the output and:

$$\mathbf{a}'_t = f_1(\mathbf{a}'_{t-1}, \mathbf{a}_t) \tag{10}$$

Factors: Let $\mathbb{S}$ be the universe of all stocks in the same stock market. Our factors are derived from both target stock price series $\mathbf{Y} \in \mathbb{R}^T$ and related stock series $\mathbf{X} \in \mathbb{R}^{T \times D}$ as mentioned. The matrix $\mathbf{X}$ contains three kinds of stock series: positive correlation series $\mathbf{X_p} \in \mathbb{R}^{T \times P}$, negative correlation series $\mathbf{X_n} \in \mathbb{R}^{T \times N}$ and index series $\mathbf{X_i} \in \mathbb{R}^T$ where $D = P + N + 1$.

It is a common scene in the stock market that a group of stocks sometimes share an identical pattern of tendency. If one of the stocks takes the lead, the rest in the group are likely to follow. This inspires us to define the series $\mathbf{X_p}$ to capture those positive related stocks. As for a target stock $i \in \mathbb{S}$, we first trace back T_r time steps from $\mathbf{Y}$ and generates an extended series $\mathbf{S}^i = (y_{-T_r}, y_{-T_r+1}, ..., y_{-1}, y_1, y_2, ..., y_T)^\top \in \mathbb{R}^{T_r+T}$. For any other stock $s \in \mathbb{S}$ and $s \neq i$, we also trace back T_r time steps and obtain $\mathbf{S}^s = (x^s_{-T_r}, x^s_{-T_r+1}, ..., x^s_{-1}, x^s_1, x^s_2, ..., x^s_T)^\top \in \mathbb{R}^{T_r+T}$ by cutting out the series in the same time period as $\mathbf{S}^i$. In this paper, we use Pearson correlation coefficient (PCC) to measure the correlation between different stock price series. Given two variables $\mathbf{A}, \mathbf{B} \in \mathbb{R}^T$, PCC is computed by:

$$r(\mathbf{A}, \mathbf{B}) = \frac{Cov(\mathbf{A}, \mathbf{B})}{\sqrt{Var(\mathbf{A})Var(\mathbf{B})}} \tag{11}$$

where $Cov(\cdot)$ and $Var(\cdot)$ are the covariance and variance, respectively.

The correlation between target stock series and $\mathbf{S}^s$, $r(\mathbf{S}^i, \mathbf{S}^s)$ can be calculated using Eqn. (11). We consider the stocks with P-largest $r(\mathbf{S}^i, \mathbf{S}^s)$ as positive correlation stocks. The last T time steps $(x^s_1, x^s_2, ..., x^s_T)^\top$ of the P positive correlation stocks constitute $\mathbf{X_p}$. On the contrary, we generate negative correlation series $\mathbf{X_n}$ by selecting N-smallest $r(\mathbf{S}^i, \mathbf{S}^s)$. And $\mathbf{X_i}$ would be the stock index series in time window T. Note that in our experiments, the largest value of P or N is 20. And in our dataset, we always have enough number (more than 20) of positive or negative stocks which have PCC larger or smaller than 0 with target stock, respectively.

We use the first LSTM layer with hidden units sized q to extract useful factors from the original series. Table 1 shows the 4 kinds of factors we use. We define "Self" and "Index" as follows:

- Self (*mainstream*):
$$\tilde{\mathbf{Y}} = LSTM(\mathbf{Y}) \tag{12}$$
- Index:
$$\tilde{\mathbf{I}} = LSTM(\mathbf{X_i}) \tag{13}$$

where $\tilde{\mathbf{Y}}, \tilde{\mathbf{I}} \in \mathbb{R}^{T \times q}$.

Each row $\mathbf{X^p_p} \in \mathbb{R}^T$, $\mathbf{X^n_n} \in \mathbb{R}^T$ will go through the same LSTM layer:

$$\tilde{\mathbf{X}}^p_\mathbf{p} = LSTM(\mathbf{X}^p_\mathbf{p}) \tag{14}$$
$$\tilde{\mathbf{X}}^n_\mathbf{n} = LSTM(\mathbf{X}^n_\mathbf{n}) \tag{15}$$

where $\tilde{\mathbf{X}}^p_\mathbf{p}, \tilde{\mathbf{X}}^n_\mathbf{n} \in \mathbb{R}^{T \times q}$, $p = 1, 2, ..., P$ and $n = 1, 2, ..., N$.

In order to decrease input dimension we implement average strategy on the extracted features $\tilde{\mathbf{X}}^p_\mathbf{p}$ and $\tilde{\mathbf{X}}^n_\mathbf{n}$. We define "Positive" and "Negative" as follows:

- Positive:

$$\tilde{\mathbf{P}} = \frac{1}{P}\sum_{p=1}^{P}\tilde{\mathbf{X}}_{\mathbf{p}}^{p} \tag{16}$$

- Negative:

$$\tilde{\mathbf{N}} = \frac{1}{N}\sum_{n=1}^{N}\tilde{\mathbf{X}}_{\mathbf{n}}^{n} \tag{17}$$

where $\tilde{\mathbf{P}}, \tilde{\mathbf{N}} \in \mathbb{R}^{T\times q}$.

4. Proposed Model

4.1. Multi-Input LSTM Model

In our model, we employ "Self" factor as *mainstream* and 3 auxiliary factors namely "Positive", "Negative" and "Index". We believe that although these factors play secondary roles compared with *mainstream*, they are still useful for predicting future stock price.

The original LSTM receives only one input vector at each time step but we can always concatenate these 4 factors into a single input through dimensional expansion. However unlike most other prediction problems, stock price prediction can be an extreme challenge since the correlation coefficient between the 3 auxiliary factors and the prediction target are rather small. As we mentioned in Section 1, if we simply expand the input dimension by concatenation, the extended dimensions might become a serious distraction rather than supplementary. Therefore, we propose the multi-input LSTM to adapt multiple input series and emphasize the *mainstream*. We employ $\tilde{\mathbf{Y}}, \tilde{\mathbf{P}}, \tilde{\mathbf{N}}, \tilde{\mathbf{I}} \in \mathbb{R}^{T\times q}$ as the input of the 4 kinds of factors and $\tilde{\mathbf{Y}}_t, \tilde{\mathbf{P}}_t, \tilde{\mathbf{N}}_t, \tilde{\mathbf{I}}_t \in \mathbb{R}^q$ as their corresponding input vector at time t where $t = 1, 2, ..., T$.

Figure 1 shows the structure of our proposed MI-LSTM. We can see the cell input of auxiliary factors are generated by both auxiliary factors and previous hidden states $\mathbf{h}_{t-1}$. However, they have no impact on their corresponding input gates.

The forget gate and output gate of LSTM remain the same compared with the original LSTM:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}; \tilde{\mathbf{Y}}_t] + \mathbf{b}_f) \tag{18}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}; \tilde{\mathbf{Y}}_t] + \mathbf{b}_o) \tag{19}$$

where $\mathbf{h}_{t-1} \in \mathbb{R}^p$ is the hidden states of the previous time step and p is the number of the MI-LSTM hidden units. $\mathbf{W}_f, \mathbf{W}_o \in \mathbb{R}^{p\times(p+q)}$ are the weight matrices of forget gate and output gate respectively. Their corresponding biases are $\mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^p$. The forget gate and output gate are $\mathbf{f}_t, \mathbf{o}_t \in \mathbb{R}^p$ respectively.

Since there are 4 input factors, the cell state inputs should match the same number:

$$\tilde{\mathbf{C}}_t = tanh(\mathbf{W}_c[\mathbf{h}_{t-1}; \tilde{\mathbf{Y}}_t] + \mathbf{b}_c) \tag{20}$$

$$\tilde{\mathbf{C}}_{pt} = tanh(\mathbf{W}_{cp}[\mathbf{h}_{t-1}; \tilde{\mathbf{P}}_t] + \mathbf{b}_{cp}) \tag{21}$$

$$\tilde{\mathbf{C}}_{nt} = tanh(\mathbf{W}_{cn}[\mathbf{h}_{t-1}; \tilde{\mathbf{N}}_t] + \mathbf{b}_{cn}) \tag{22}$$

$$\tilde{\mathbf{C}}_{it} = tanh(\mathbf{W}_{ci}[\mathbf{h}_{t-1}; \tilde{\mathbf{I}}_t] + \mathbf{b}_{ci}) \tag{23}$$

where $\tilde{\mathbf{C}}_t, \tilde{\mathbf{C}}_{pt}, \tilde{\mathbf{C}}_{nt}, \tilde{\mathbf{C}}_{it} \in \mathbb{R}^p$ are the cell state inputs of *mainstream*, "Positive", "Negative" and "Index" factors. $\mathbf{W}_c, \mathbf{W}_{cp}, \mathbf{W}_{cn}, \mathbf{W}_{ci} \in \mathbb{R}^{p\times(p+q)}$ are the weight matrices and $\mathbf{b}_c, \mathbf{b}_{cp}, \mathbf{b}_{cn}, \mathbf{b}_{ci} \in \mathbb{R}^p$ are the biases.

To control auxiliary factors all input gates are decided by *mainstream* and previous hidden states.

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}; \tilde{\mathbf{Y}}_t] + \mathbf{b}_i) \tag{24}$$

$$\mathbf{i}_{pt} = \sigma(\mathbf{W}_{ip}[\mathbf{h}_{t-1}; \tilde{\mathbf{Y}}_t] + \mathbf{b}_{ip}) \tag{25}$$

$$\mathbf{i}_{nt} = \sigma(\mathbf{W}_{in}[\mathbf{h}_{t-1}; \tilde{\mathbf{Y}}_t] + \mathbf{b}_{in}) \tag{26}$$

$$\mathbf{i}_{it} = \sigma(\mathbf{W}_{ii}[\mathbf{h}_{t-1}; \tilde{\mathbf{Y}}_t] + \mathbf{b}_{ii}) \tag{27}$$

where $\mathbf{i}_t, \mathbf{i}_{pt}, \mathbf{i}_{nt}, \mathbf{i}_{it} \in \mathbb{R}^p$ are the input gates of the 4 kinds of factors. Notice that they are not related to auxiliary factors. $\mathbf{W}_i, \mathbf{W}_{ip}, \mathbf{W}_{in}, \mathbf{W}_{ii} \in \mathbb{R}^{p\times(p+q)}$ are the weight matrices and $\mathbf{b}_i, b_{ip}, \mathbf{b}_{in}, \mathbf{b}_{ii} \in \mathbb{R}^p$ are their corresponding biases.

In LSTM, the element-wise multiply product of $\tilde{\mathbf{C}}_t$ and $\mathbf{i}_t$ is the final cell state input at time t. However when there are 4 cell inputs $\tilde{\mathbf{C}}_t, \tilde{\mathbf{C}}_{pt}, \tilde{\mathbf{C}}_{nt}, \tilde{\mathbf{C}}_{it}$, we have to combine them first. Here we assign attention weights to different cell inputs. First we use the input gates to filter the cell inputs by element-wise multiplication:

$$\mathbf{l}_t = \tilde{\mathbf{C}}_t * \mathbf{i}_t \tag{28}$$

$$\mathbf{l}_{pt} = \tilde{\mathbf{C}}_{pt} * \mathbf{i}_{pt} \tag{29}$$

$$\mathbf{l}_{nt} = \tilde{\mathbf{C}}_{nt} * \mathbf{i}_{nt} \tag{30}$$

$$\mathbf{l}_{it} = \tilde{\mathbf{C}}_{it} * \mathbf{i}_{it} \tag{31}$$

where $\mathbf{l}_t, \mathbf{l}_{pt}, \mathbf{l}_{nt}, \mathbf{l}_{it} \in \mathbb{R}^p$ are the cell inputs of different factors. Then, weighted sum is performed on these cell inputs:

$$\mathbf{L}_t = \alpha_t \mathbf{l}_t + \alpha_{pt} \mathbf{l}_{pt} + \alpha_{nt} \mathbf{l}_{nt} + \alpha_{it} \mathbf{l}_{it} \tag{32}$$

where $\alpha_t, \alpha_{pt}, \alpha_{nt}, \alpha_{it} \in \mathbb{R}$ are the attention weights and $\mathbf{L}_t \in \mathbb{R}^p$ is the final cell state input at time t. The attention weights are decided by the cell inputs themselves and the cell state at previous time step:

$$u_t = tanh({\mathbf{l}_t}^\top \mathbf{W}_a \mathbf{C}_{t-1} + b_a) \tag{33}$$

$$u_{pt} = tanh({\mathbf{l}_{pt}}^\top \mathbf{W}_a \mathbf{C}_{t-1} + b_{ap}) \tag{34}$$

$$u_{nt} = tanh({\mathbf{l}_{nt}}^\top \mathbf{W}_a \mathbf{C}_{t-1} + b_{an}) \tag{35}$$

$$u_{it} = tanh({\mathbf{l}_{it}}^\top \mathbf{W}_a \mathbf{C}_{t-1} + b_{ai}) \tag{36}$$

$$[\alpha_t, \alpha_{pt}, \alpha_{nt}, \alpha_{it}]^\top = Softmax([u_t, u_{pt}, u_{nt}, u_{it}]^\top) \tag{37}$$

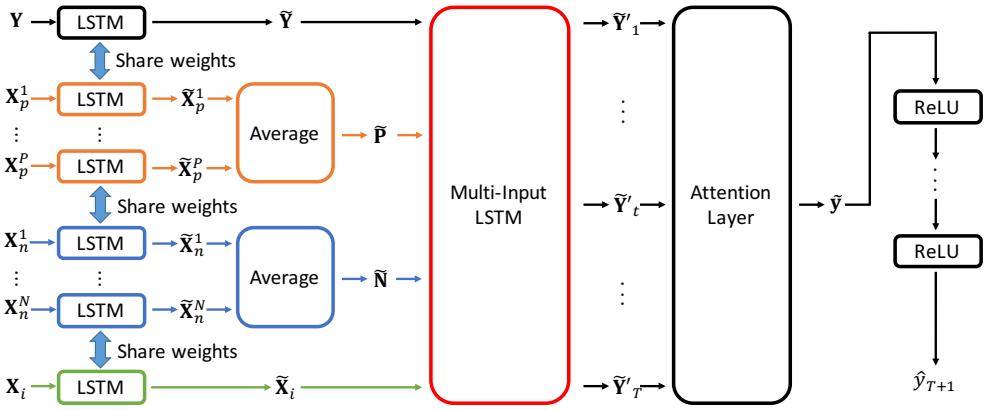

Figure 2: Illustration of our entire model.

where $u_t, u_{pt}, u_{nt}, u_{it} \in \mathbb{R}$ are the intermediate products of attention weights. $\mathbf{W}_a \in \mathbb{R}^{p \times p}$, $b_a \in \mathbb{R}$ are the parameters to be learned. $\mathbf{C}_{t-1} \in \mathbb{R}^p$ is the cell state at time $t-1$. The intermediate products are fed through a softmax layer as a vector to generate the attention weights. Note that the attention vector is able to change at every time step. When the update vector of cell state $\mathbf{L}_t$ is settled, the rest of our MI-LSTM unit work the same way compare with original LSTM:

$$\mathbf{C}_t = \mathbf{C}_{t-1} * \mathbf{f}_t + \mathbf{L}_t \tag{38}$$

$$\mathbf{h}_t = tanh(\mathbf{C}_t) * \mathbf{o}_t \tag{39}$$

where $\mathbf{C}_t, \mathbf{h}_t \in \mathbb{R}^p$ are the cell state and output at time t. The whole process of MI-LSTM are Eqn. (18) - (39) which we use function f_2 to summarize:

$$\mathbf{h}_t = f_2(\mathbf{h}_{t-1}, \tilde{\mathbf{Y}}_t, \tilde{\mathbf{P}}_t, \tilde{\mathbf{N}}_t, \tilde{\mathbf{I}}_t) \tag{40}$$

Similar to Eqn. (9), let $\tilde{\mathbf{Y}}' = (\tilde{\mathbf{Y}}'_1, \tilde{\mathbf{Y}}'_2, ..., \tilde{\mathbf{Y}}'_T) \in \mathbb{R}^{T \times p}$ be the output we can also use a non-linear function to represent a MI-LSTM layer:

$$\tilde{\mathbf{Y}}' = MILSTM(\tilde{\mathbf{Y}}, \tilde{\mathbf{P}}, \tilde{\mathbf{N}}, \tilde{\mathbf{I}}) \tag{41}$$

where

$$\tilde{\mathbf{Y}}'_t = f_2(\tilde{\mathbf{Y}}'_{t-1}, \tilde{\mathbf{Y}}_t, \tilde{\mathbf{P}}_t, \tilde{\mathbf{N}}_t, \tilde{\mathbf{I}}_t) \tag{42}$$

As aforementioned, we hope that even if the additional factors are useless, they should never make things worse. Thus matrix $\mathbf{W}_{cp}, \mathbf{W}_{cn}, \mathbf{W}_{ci}$ and biases $\mathbf{b}_{cp}, \mathbf{b}_{cn}, \mathbf{b}_{ci}$ are all initialized to 0 which means that the auxiliary factors are ignored in the very beginning. Hopefully the information from auxiliary factors will gradually flow in with the training process under the control of *mainstream*.

Based on the MI-LSTM units, we construct our prediction model as shown in Figure 2. The factors are generated by the first LSTM layer, then they are fed into MI-LSTM.

Using Eqn. (41) we obtain a the hidden states at each time step $\tilde{\mathbf{Y}}' = (\tilde{\mathbf{Y}}'_1, \tilde{\mathbf{Y}}'_2, ..., \tilde{\mathbf{Y}}'_T) \in \mathbb{R}^{T \times p}$. In our model, we further employ temporal self-attention on $\tilde{\mathbf{Y}}'$ which can be calculated as follows:

$$j_t = \mathbf{v}_b{}^\top tanh(\mathbf{W}_b(\tilde{\mathbf{Y}}'_t)^\top + \mathbf{b}_b) \tag{43}$$

$$\beta = Softmax([j_1, j_2, ..., j_T]^\top) \tag{44}$$

where matrix $\mathbf{W}_b \in \mathbb{R}^{p\times p}$, bias $\mathbf{b}_b \in \mathbb{R}^p$ and vector $\mathbf{v}_b \in \mathbb{R}^p$ are the parameters to learn. The vector consists of the intermediate product $j_t \in \mathbb{R}$ is fed into a softmax layer to generate the temporal attention vector $\beta \in \mathbb{R}^T$.

The attention output will be:

$$\tilde{\mathbf{y}} = \beta^\top \tilde{\mathbf{Y}}' \tag{45}$$

where $\tilde{\mathbf{y}} \in \mathbb{R}^p$. Vector $\tilde{\mathbf{y}}$ is finally fed through several fully connected NN layers to make the prediction:

$$\hat{y}_{T+1} = Activation(\mathbf{W}\tilde{\mathbf{y}} + b) \tag{46}$$

where $\mathbf{W} \in \mathbb{R}^{1\times p}, b \in \mathbb{R}$ are NN parameters to learn. Note that for simplicity, Eqn. (46) only describes a one-layer fully connected NN since it directly outputs the prediction. Additional NN layers can be easily added between $\tilde{\mathbf{y}}$ and $\hat{y}_{T+1}$ through similar equation. The activation can be various non-linear functions such as sigmoid, ReLU, etc. In our experiments we select ReLU as our activation.

4.2. Training

We use RMSProp Hinton et al. (2012) as our training optimizer with a learning rate of 0.001. In terms of the data size, the minibatch size is 512. The samples are shuffled at the end of each iteration. The model is learned by minimizing the mean square error over all train samples:

$$\mathcal{L} = \frac{1}{K}\sum_{k=1}^{K}(y^k_{T+1} - \hat{y}^k_{T+1})^2 \tag{47}$$

where K is the number of training samples.

5. Experiments

5.1. Data

We choose the major stocks under CSI-300 index as our dataset. Among the 300 stocks we remove 40 stocks whose historical record is less than 4 years. We also treat CSI-300 index as an individual stock thus our dataset contains 261 stocks' opening price from April 25, 2013 to May 15, 2017. Validation set is derived from the latest 300-101 days and testing set is derived from the latest 100 days, while the rest is used for training. With fixed time window size $T = 10$ and stride 1 we generate 249,516 samples from 261 stocks (including index) totally, including 171,216 train samples, 52,200 validate samples and 26,100 test samples.

5.2. Experimental Settings

The parameters in our model include LSTM dimension q, MI-LSTM dimension p, time window size T related timesteps T_r, positive stock number P and negative stock number N.

Table 2: Model performance.

Models	Min. **MSE** ($\times 10^{-3}$)	Avg. **MSE** ($\times 10^{-3}$)
LSTM	1.017	1.042
LSTM-C	1.050	1.124
DA-RNN	1.511	1.706
MI-LSTM	**0.996**	**1.012**
LSTM-CN	1.072	1.105
MI-LSTM-N	1.018	1.045

For convenience the following parameters is settled: $q = p = 64$, $T = 10$, $T_r = 20$. P and N are equal and chosen from $\{4, 6, 8, 10, 15, 20\}$. As for DA-RNN, the LSTM dimension of encoder and decoder are both 64 ($m = p = 64$ as defined in original paper). When one stock is treated as the target series, the rest 260 stocks would be the driving series (defined in original paper). Time window are also set to 10.

We choose the mean square error **MSE** of all samples as our metric. It can be calculated through Eqn. (47) using test set. Notice that the **MSE** we calculated are derived from normalized data. That's because there exists huge value gap among different stocks. If we use original stock price to evaluate error, the error of high price stocks would probably be much more larger than low price ones, which implies models perform better on high price stocks would very likely to have better overall performance. Thus the performance on low price stocks would become dispensable. To avoid the bias caused by the aforementioned problem we evaluate the error with normalized stock price ranged from -1 to 1.

5.3. Comparison Results

In Table 2, we compare our model with LSTM and DA-RNN. The first row is original LSTM which does not consider auxiliary series. LSTM-C is also original LSTM but concatenates both *mainstream* and auxiliary series at each time step. The fourth row is the results of our proposed MI-LSTM, which considers both series while emphasizing the *mainstream*. Compare the results of LSTM with LSTM-C, we could see that the average **MSE** increases from 1.042 to 1.124 while minimum **MSE** deteriorates from 1.017 to 1.050. That means when the auxiliary factors are added through expanded dimensions, the extra factors are actually harmful to the prediction. However, it does not mean that these factors are useless. As we can see, our proposed model achieves 0.996 in minimum **MSE** and 1.012 in average **MSE**. This result proves our proposed MI-LSTM is able to capture some valuable information which might be ignored by original LSTM. Besides, the DA-RNN model does not perform well in our experiments. This might be caused by 2 reasons:

- DA-RNN is used to predict only the NASDAQ100 index value given historical value and 81 stock series under NASDAQ100 as driving series. Besides, the time stamp of the prediction target is the same as the latest time stamp of driving series unlike the data we use which has a one-step gap. Hence DA-RNN would face a much more noisy situation while using our data set.

Table 3: Related Stock Number selection.

P	4	6	8	10	15	20
Min. **MSE** ($\times 10^{-3}$)	1.036	1.026	1.024	**0.996**	1.035	1.012
Avg. **MSE** ($\times 10^{-3}$)	1.066	1.050	1.048	**1.012**	1.056	1.054

- DA-RNN implements attention among 81 stocks, but there are 260 stocks in our data set, which makes the learning of attention weights more difficult.

As for the last 2 rows of Table 2, we want to know what will happen when the auxiliary seires are totally useless. The model and data shape of LSTM-CN and MI-LSTM-N are completely the same as LSTM-C and MI-LSTM respectively. But we replace the auxiliary series with Gaussian noise. As we expected, MI-LSTM achieves better performance since the noise inputs are filtered by *mainstream*. Notice that MI-LSTM-N (1.045) is just very slightly worse than LSTM (1.042) which indicates that the Gaussian noise inputs can hardly effect the result while using MI-LSTM. Another interesting thing is that if we compare LSTM-CN with LSTM-C, it turns out the average **MSE** of LSTM-C (1.124) is even larger than LSTM-CN (1.105). This phenomenon not only reveals the shortage of original LSTM but also further explains the potential consequence of improper factor selection.

5.4. Effect of Related Stock Number

Since we employ the stock price of related stocks to generate factors, we want to know how many related stocks should be considered. Table 3 shows how related stock number $P = N$ influence the performance of our model. From the results we can see that the performance increases along with P when the number of related stocks is rather small. At $P = 10$ the mean square error performance come to a peak which is 1.012×10^{-3} in average and the minimum reaches 0.996×10^{-3}. But it does not mean that larger P gets better result. We can see the performance deteriorates when $P = 15$ and $P = 20$.

5.5. Effect of Attention Module

Figure 3 shows the trend of attention weights inside MI-LSTM during training. The weights are calculated by Eqn. (37), consider only the last time step T and perform average among all samples. We can see the *mainstream* is gradually assigned the largest attention weights while the attention weights of "Positive" and "Negative" factors drop rather quickly. But the "Index" weights become larger. The average attention weights of *mainstream* and CSI-300 index converge to about 0.32 and 0.29 when the model parameters are stable. In economics we know stock market index is an important and comprehensive symbol which can largely reflect recent market environment. Thus our result is another evidence to prove its significance. On the other hand, the attention weights of "Positive" or "Negative" factors are generally less than 0.2. These kind of factors might be harmful if they are not treated properly.

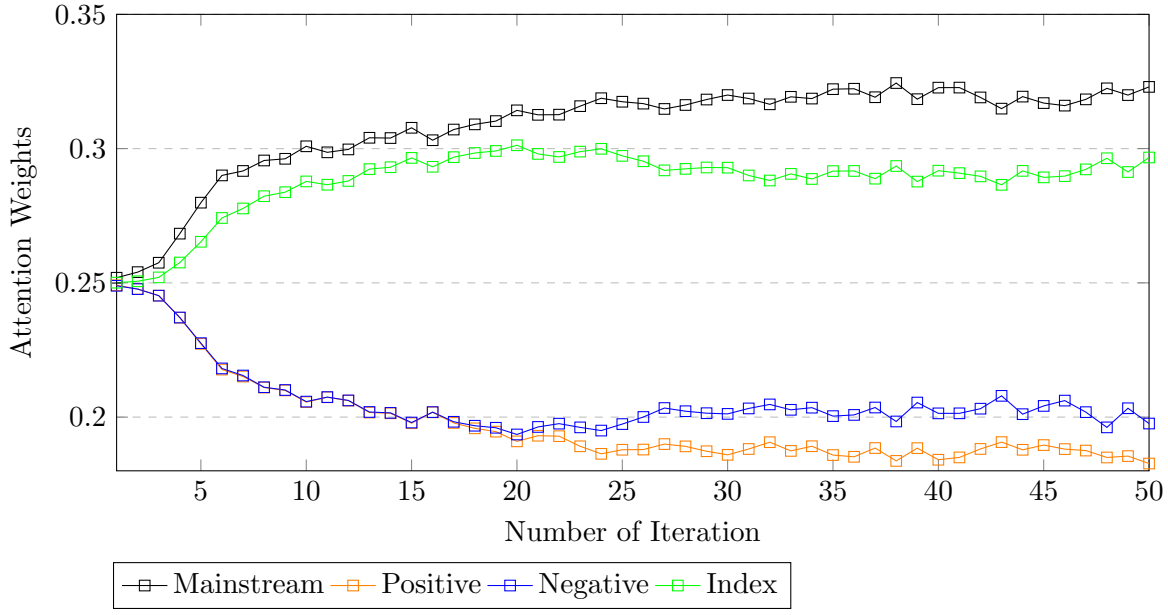

Figure 3: The visualization of attention weights of all cell input candidates in MI-LSTM. It illustrates how attention weights change during train process.

Table 4: Profit Comparison of Different Models.

Date	20161214	20170116	20170215	20170315	20170414	20170515
MI-LSTM	**100**	**97.88**	**102.89**	**106.18**	**113.19**	**104.69**
LSTM-C	100	96.69	99.76	102.24	108.67	97.93
CSI-300	100	97.27	100.27	101.50	102.17	99.61

5.6. Profit of Portfolio

We empirically show which stocks should be invested everyday. Specifically, we predict the stock price of the next day and calculates the return. Given a portfolio sized 20, we select the 20 stocks with largest return into the portfolio. Portfolio changes everyday and the transaction cost is ignored. Originally we have 100 units of property, the property is allocated equally to each stock in the portfolio. We use CSI-300 as benchmark. From Table 4 we can see that MI-LSTM outperforms LSTM significantly.

5.7. Discussion

For convenience we only use 3 additional factors. But that does not mean our model can only cope with limited factors. In fact, the first LSTM layer is used for feature extraction on arbitrary dimension inputs. Thus the input data will be handled into a fixed shape before they are fed into MI-LSTM.

6. Conclusion

In this paper, we propose an improved MI-LSTM based on LSTM and attention mechanism, which achieves better performance in extracting potential information and filtering noise. MI-LSTM units assign different weights to different input series to keep the dominant status of *mainstream*, while absorbing information from leaky input gates. The output of MI-LSTM is further processed using temporal self-attention. Based on these stage attentions, our model can not only focus on the most important factors but also adaptively capture the most relevant time steps. By employing 3 additional factors and Gaussian noise, we design experiments to prove our improvements over both original LSTM and DA-RNN. Additionally, we successfully enable positive related stocks and negative related stocks to facilitate the prediction task and discover that an appropriate number of related stocks help achieve better prediction performance.

Acknowledgments

This research is supported by the National Key Research and Development Program of China (No. 2018YFC0831604), NSFC (no. 61602297, 61772341, 61472254, 61572324, 61170238, and 61472241), Singapore NRF (CREATE E2S2). Yanmin Zhu is also supported by the Program for Changjiang Young Scholars in University of China, and the Program for Shanghai Top Young Talents. Yanyan Shen is the corresponding author.

References

KLAUS ADAM, ALBERT MARCET, and JUAN PABLO NICOLINI. Stock market volatility and learning. *The Journal of Finance*, 71(1):33–82, 2016.

Ayodele Ariyo Adebiyi, Aderemi Oluyinka Adewumi, and Charles Korede Ayo. Comparison of arima and artificial neural networks models for stock price prediction. *Journal of Applied Mathematics*, 2014, 2014.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, Mar 1994.

I. Bordino, N. Kourtellis, N. Laptev, and Y. Billawala. Stock trade volume prediction with yahoo finance user browsing behavior. In *2014 IEEE 30th International Conference on Data Engineering*, pages 1168–1173, March 2014.

Ying-Hua Chang and Ming-Sheng Lee. Incorporating markov decision process on genetic algorithms to formulate trading strategies for stock markets. *Applied Soft Computing*, 52:1143 – 1153, 2017.

Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. Attentive collaborative filtering: Multimedia recommendation with item- and component-level attention. In *Proceedings of the 40th International ACM SIGIR Conference on*

Research and Development in Information Retrieval, SIGIR '17, pages 335–344. ACM, 2017a.

Qin Chen, Qinmin Hu, Jimmy Xiangji Huang, Liang He, and Weijie An. Enhancing recurrent neural networks with positional attention for question answering. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 993–996. ACM, 2017b.

Yingjun Chen and Yongtao Hao. A feature weighted support vector machine and k-nearest neighbor algorithm for stock market indices prediction. *Expert Systems with Applications*, 80:340 – 355, 2017.

Edward Choi, Mohammad Taha Bahadori, Le Song, Walter F. Stewart, and Jimeng Sun. Gram: Graph-based attention model for healthcare representation learning. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 787–795. ACM, 2017.

Eunsuk Chong, Chulwoo Han, and Frank C. Park. Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies. *Expert Systems with Applications*, 83:187 – 205, 2017.

Xiao Ding, Yue Zhang, Ting Liu, and Junwen Duan. Deep learning for event-driven stock prediction. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2327–2333, 2015.

G Hinton, Nitish Srivastava, and Kevin Swersky. Rmsprop: Divide the gradient by a running average of its recent magnitude. *Neural networks for machine learning, Coursera lecture 6e*, 2012.

Sepp Hochreiter and Jrgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Fang Jin, Wei Wang, Prithwish Chakraborty, Nathan Self, Feng Chen, and Naren Ramakrishnan. Tracking multiple social media for stock market event prediction. In Petra Perner, editor, *Advances in Data Mining. Applications and Theoretical Aspects*, pages 16–30. Springer International Publishing, 2017.

Linkai Luo, Shiyang You, Yanru Xu, and Hong Peng. Improving the integration of piece wise linear representation and weighted support vector machine for stock trading signal prediction. *Applied Soft Computing*, 56:199 – 216, 2017.

Fenglong Ma, Radha Chitta, Jing Zhou, Quanzeng You, Tong Sun, and Jing Gao. Dipole: Diagnosis prediction in healthcare via attention-based bidirectional recurrent neural networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 1903–1911. ACM, 2017.

F. Ming, F. Wong, Z. Liu, and M. Chiang. Stock market prediction from wsj: Text mining via sparse matrix factorization. In *2014 IEEE International Conference on Data Mining*, pages 430–439, Dec 2014.

Thien Hai Nguyen, Kiyoaki Shirai, and Julien Velcin. Sentiment analysis on social media for stock movement prediction. *Expert Systems with Applications*, 42(24):9603 – 9611, 2015.

Yao Qin, Dongjin Song, Haifeng Cheng, Wei Cheng, Guofei Jiang, and Garrison W. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *CoRR*, abs/1704.02971, 2017.

Akhter Mohiuddin Rather, Arun Agarwal, and V.N. Sastry. Recurrent neural network and a hybrid model for prediction of stock returns. *Expert Systems with Applications*, 42(6): 3234 – 3241, 2015.

Ritika Singh and Shashi Srivastava. Stock prediction using deep learning. *Multimedia Tools Appl.*, 76(18):18569–18584, 2017.

Chih-Fong Tsai and Yu-Chieh Hsiao. Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches. *Decision Support Systems*, 50(1):258–269, 2010.

Xuejian Wang, Lantao Yu, Kan Ren, Guanyu Tao, Weinan Zhang, Yong Yu, and Jun Wang. Dynamic attention deep model for article recommendation by learning human editors' demonstration. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 2051–2059. ACM, 2017.

P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, Oct 1990.

Yi Xiao, Jin Xiao, John Liu, and Shouyang Wang. A multiscale modeling approach incorporating arima and anns for financial market volatility forecasting. *Journal of Systems Science and Complexity*, 27(1):225–236, Feb 2014.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015a.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 2048–2057, 2015b.

Liheng Zhang, Charu Aggarwal, and Guo-Jun Qi. Stock price prediction via discovering multi-frequency trading patterns. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 2141–2149. ACM, 2017.