Abstract
The time series of stock prices are non-stationary and nonlinear, making the prediction of future price trends much challenging. To learn long-term dependencies of stock prices, we first perform unsupervised learning to extract and construct useful features, then build a deep Long Short-Term Memory (LSTM) network to generate the prediction. The experiments on real market dataset demonstrate that the proposed model outperforms other four baseline models in the mean square error.

1. Introduction
In the financial industry, stock price prediction has constantly been a popular field of research. According to many widely accepted studies, stock markets have been proved to be predictable in some scenarios. While different features are available for prediction, it’s interesting to focus solely on past trading patterns. In the last decade, there has been a huge increase in the application of deep neural network. As a result, applying deep learning on stock market prediction has become a field of interest.

1.1. Literature Review
Our research assumes minute-level price fluctuation pattern is independent of corporate fundamentals and macro economy. Thus, unlike the studies of (Chiang et al., 2016), (Chourmouziadis & Chatzoglou, 2016), and (Zhong & Enke, 2017) in which daily price data are used as input, we seek to develop a predictive model based on minute-level price data. The prediction of future stock price had also been understood as both classification and regression problems in previous studies. (liang Chen & yu Chen, 2016) and (Zhong & Enke, 2017) provided prediction of market direction as either up or down. In more complicated cases, (Chourmouziadis & Chatzoglou, 2016) specified cash and stock within the optimal portfolio composition. Our study intends to give a prediction of the stock return of the next minute.

There have been linear and nonlinear models to predict stock price movement with varying degrees of success. (Chong et al., 2017) noted a multilayer artificial neural network might be particularly suitable with such time-series data, due to its higher computational power and sophistication of algorithm. Such model selects features based on raw input price data automatically and does not require understanding or providing data from the side of fundamentals or macro economy, which fits our assumption about minute-level price fluctuation pattern. For performance measurement, previous studies have used trade simulation or various MSE methods (Chiang et al., 2016); (Chourmouziadis & Chatzoglou, 2016); (Zhong & Enke, 2017); (Chong et al., 2017).

1.2. Data
Due to certain limitation and just for preliminary testing of our strategy, we are currently using 30 days of minute-level price data of 50 stocks, in total of 11700 time steps. Our goal is to obtain data of 10 years for model training. Each stock, at each time step, has 5 features, open/high/low/close price and volume. Therefore, in total, each time step contains 250 features. The input will contain 60 lagged time step, and we aim to predict the close prices of the 50 stocks at next time step. We may adjust the number of lagged periods for better performance later. We use the first 28 days as training, the 29th day as validation for hyperparameter tuning, and the last day as test.

2. Method
We first transform the data into different feature representations, and then fit a LSTM neural network to the trans-
formed features.

2.1. Data Representation

First the price data is transformed to returns, which is very common in stock price analysis, i.e.

\[ r_{t+1} = \frac{x_{t+1} - x_t}{x_t} \]

We then experiment with three different data processing techniques:

- Normalization: \( v_{n,t+1} = \frac{v_{n,t+1} - m_n}{\sigma_n} \), where \( m_n \) and \( \sigma_n \) are the mean and standard deviation of the volumes of all time steps of stock \( n \).

- Principal component analysis: we keep the first 100 principal components, which account for 99.4% of the variance in the data.

- Auto encoder: we train a simple auto encoder that extracts 100 features from the 250 dimensional input. The auto encoder achieved reconstruction relative MSE error as low as 1.3%.

Experiments showed that normalization, PCA, and auto encoder actually all make the performance worse, and the best result is achieved with simply the returns.

2.2. LSTM Network

A Long short-term memory (LSTM) unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. Therefore, LSTM networks are well-suited for time series data.

2.3. Model

We propose two approaches to formulate the problem.

1. Many to one approach. (right figure) The output is explicitly dependent on the previous 60 time steps (i.e. 1 hr). We unroll the network for 60 time steps, and only take the prediction at the last time step. This way, when we perform each prediction, we first run the model for the previous 60 time steps and then only take the last output.

2. Many to many approach. (left figure). The prediction is made at each time step, and the model can deal with sequence of arbitrary length.

Experiments show that the many to many approach is roughly the same as many to one approach, but it’s also far less computationally expensive and more flexible. Therefore, we proceed with the many to many approach. After a grid search, we found that a network of 3 LSTM layers, each with 100 units achieves best performance. We also thought of using unsupervised feature extraction before the network, but it does not give any improvement.

After a grid search over possible architectures (1 layer to 15 layers, 50 to 250 cells each layer), we found 3 layers with 100 LSTM cells each layer achieves the best performance in terms of validation MSE error. We also tried other possible modifications to the network, including adding additional fully connected layers at the end of the network, but no significant improvement in performance was observed. Figure-1 shows the structure of our network.

2.4. Implementation and Training

We implement our network in Tensorflow with the dynamic rnn functionality. Xavier initialization is used to initialize the weights, a dropout probability of 0.5 is employed, and L2 regularization (on the weights only, not the biases) with \( \lambda = 0.01 \) is used to prevent overfitting. We use the Adam optimizer to optimize the MSE error between the predicted and the true price sequence, i.e.

\[ L(\Theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{50} (y_{t,i} - y_{t,i})^2 \]

The initial learning rate is set to \( 10^{-2} \), and is decreased by half for every 500 time steps. The training is also supervised by early stopping on validation error and stops at 24000 epochs.
3. Experimental Results

To evaluate our model performance on the test set, we employ the most popular and probably the standard validation strategy in time series analysis, the Walk Forward Validation strategy, where we do not retrain our model on each time step, but we make the last tested data point available to the model. In the case of our RNN, we do not reset the initial state, and simply let the model run forward to make prediction at each time step.

We define three metrics to quantify the error and evaluate the performance of the model:

- Mean Squared Error (MSE): \( \text{mean}((Y' - Y)^2) \)
- Mean Relative Error (MRE): \( \text{mean}(\text{abs}(\frac{Y' - Y}{Y})) \)
- Trend Error (TE): \( \text{mean}(I[Y' \times Y < 0]) \)

where the mean is over all dimensions, \( I \) is element-wise indicator variable, and the square and the division are element-wise.

3.1. Results

Overall, we could see from Figure-2 that our predictions of price at all time steps were not deviating by much from the actual price. Nevertheless, the comparison of predicted versus actual change in price in Figure-3 reveals more insight about how our model performs over time. We could observe that the variance of our predicted returns’ data was initially higher than the variance of actual data; we frequently predicted sudden jumps and falls when there were actually moderate price increases and decreases. This might in part be attributed to the fact that stock market indeed fluctuates more vehemently at open in the morning, when people make large transactions in reaction to the news released after yesterday’s close time. Specifically, the 50 minutes’ actual return data of all of AAPL, MSFT, and AMZN in Figure-3 were very volatile compared to later time steps, while our model just predicted the changes to be more significant. However, the variance of our predicted returns’ data converges to the actual variance at the end of the training set, which is also the end of the trading day.

The improvement of accuracy can also be reflected by trend error over time. Trend error is defined as incorrectly predicting positive price change when the actual change is negative, or predicting negative change when it is actually positive. Figure-4 shows that the percentage of trend error, initially over 25%, decreases rapidly in the first 50 time steps and stabilizes at around 4-5% after 250 time steps. The predicted value does not match actual value at initial steps because, with a few data processed, the model is not, in some sense, mature yet. RNN model improves when there

![Figure 2. Comparison between true and predicted price.](image)
Figure 3. Comparison between true and predicted price change.

Figure 4. Trend error averaged over all 50 stocks for each time step.

are more data, especially when more history data are added into the model. This improvement of accuracy shows that history data has considerable influence on later prediction. Therefore, to yield a better prediction accuracy in practical occasion, the model should be run with some data before serious prediction starts.

3.2. Comparison

We compare our method with two popular statistical methods ARIMA and GARCH, the method from (Chong et al., 2017), and a simple feed forward network of 6 layers, each with 200 neurons to make the total number parameters roughly equal to that of our LSTM network. Table-1 shows the results.

<table>
<thead>
<tr>
<th>Method</th>
<th>MSE</th>
<th>RE (10^-4)</th>
<th>TE (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROPOSED</td>
<td>0.511</td>
<td>3.32</td>
<td>12.48</td>
</tr>
<tr>
<td>ARIMA</td>
<td>7.24</td>
<td>47.13</td>
<td>33.54</td>
</tr>
<tr>
<td>GARCH</td>
<td>10.56</td>
<td>68.74</td>
<td>35.12</td>
</tr>
<tr>
<td>CHONG 2017</td>
<td>3.89</td>
<td>25.32</td>
<td>31.37</td>
</tr>
<tr>
<td>DEEP FNN</td>
<td>2.17</td>
<td>14.13</td>
<td>22.96</td>
</tr>
</tbody>
</table>

Clearly, our proposed model beats the other models on all three measurements. While TE shows that our improvement on the prediction of categories, i.e. upside or downside price change, is significant, MSE and MRE further indicate that our prediction of the scale of change improves dramatically. Among the other models, a simple fully-connected neural network model (FNN) works relatively well. The other models were intended for predicting daily stock price changes, and we find they are not very applicable to minute-level prediction. Thus, times-series data such as stock prices, especially at minute-level, might be particularly suitable for neural network models.

Although the proposed model made great improvement on predicting short-term stock price fluctuation, we recognize that the model still needs major modification before practical use. It is difficult to make the two buy and sell transactions within one minute, which is the time step we used for prediction. A potential direction of change would be to predict the fluctuation in the next 5 to 10 minutes based on minute-level input data.

4. Conclusions

The results show that we beat the other methods on all 3 metrics. We achieve a trend error rate of 12.48%, which means we can correctly predict whether the stock price is going up or down at next minute most of the times. The performance is consistent for all stocks after a short period of time.

On important possible future research direction is acquiring more data, which will enable us to build more deep and sophisticated network to detect the hidden patterns in the stock market. We could also include other features in our data, for example, more stocks or indexes like Standard & Poor and NASDAQ. We chose the 50 stocks based on their market size, but other criterion like basing on industry is also plausible.

In conclusion, by utilizing deep learning and high-frequency trading data, our model obtains better prediction in stock market fluctuations. We believe that this indicates a promising direction of research in the field of stock market prediction, and we hope to see additional human, hardware and data resources in this field of research in future.
Acknowledgments

We thank Guoyan Zheng and Eason Chan for their LSTM and tensorflow code templates (not publicly available).

References


