Price Forecast with High-Frequency Finance Data: An Autoregressive Recurrent Neural Network Model with Technical Indicators

Yuechun Gu
University of Electronic Science and Technology of China
Chengdu, China
329352129@qq.com

Da Yan
University of Alabama at Birmingham
Birmingham, AL
yanda@uab.edu

Sibo Yan
Freddie Mac
Washington, D.C.
sibo_yan@freddiemac.com

Zhe Jiang
University of Alabama
Tuscaloosa, AL
zjiang@cs.ua.edu

ABSTRACT

The availability of high-frequency trade data has made it possible for the intraday forecast of price patterns. With the help of technical indicators, recent studies have shown that LSTM based deep learning models are able to predict price directions (a binary classification problem) with performance better than a random guess. However, only naïve recurrent networks were adopted, and these works did not compare with the tools used by finance practitioners. Our experiments show that GARCH beats their LSTM models by a large margin.

We propose to adopt an autoregressive recurrent network instead so that the loss of the prediction at every time step contributes to the model training; we also treat a rich set of technical indicators at each time step as covariates to enhance the model input. Finally, we treat the problem of price pattern forecast as a regression problem on the price itself; even for price direction prediction, we show that our performance is much better than if we model the problem as binary classification. We show that only when all these designs are adopted, an LSTM model can beat GARCH (and by a large margin).

This work corrects the poor use of LSTM networks in recent studies, and provides "the" baseline that is able to fully unleash the power of LSTM for future work to compare with. Moreover, since our model is a price regressor with very good prediction performance, it can serve as a valuable tool for designing trading strategies (including day trading). Our model has been used by quantitative analysts in Freddie Mac for over one quarter, and is found to be more effective than traditional GARCH variants in market prediction.

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3412738

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Supervised learning*; • **Applied computing** → **Forecasting**;

KEYWORDS

high-frequency; stock price; technical indicators; GARCH; recurrent neural network; autoregressive

ACM Reference Format:
Yuechun Gu, Da Yan, Sibo Yan, and Zhe Jiang. 2020. Price Forecast with High-Frequency Finance Data: An Autoregressive Recurrent Neural Network Model with Technical Indicators. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3340531.3412738

1 INTRODUCTION

AI and data science have gained significant momentum in adoption thanks to the advancement of deep learning, which allows simple end-to-end model training with much higher accuracy than traditional machine learning models. A key feature of data science is to apply statistics, data analysis and machine learning to find values from the data in a specific domain, and deep learning models have enabled effective applications in many domains such as computer vision, natural language processing, medicine and autonomous driving.

Using deep learning models as FinTech tools has recently gained attention where predictive models are built to forecast the stock price in the future. Recurrent neural network (RNN) supports effective prediction from a temporal sequence and becomes a natural and promising deep learning tool for stock price prediction that can beat conventional statistics and machine learning approaches. Surprisingly, our literature review shows that even through a number of works have been done in this direction, they are not configuring the RNN in the best manner to carry out its predictive potential. Moreover, statistical tools commonly used by finance practitioners are also not compared as baseline approaches in these works.

In this work, deep learning practitioners and a financial domain practitioner (i.e., Sibo Yan who is a PhD in Economics from UCLA) work together as a team to deliver an effective RNN model that is

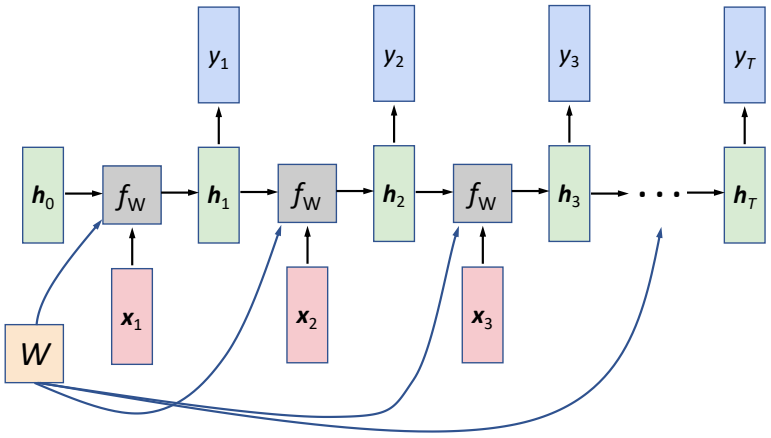

Figure 1: A Many-to-Many Recurrent Neural Network

properly configured to serve as "the" baseline for future work to compare against. We highlight the following features:

- The prediction problem should be modeled as a price **regression** even when we are considering the binary classification problem of predicting future price directions.
- The RNN model should be **autoregressve**, allowing the loss of the price prediction at every time step to contribute back to the model training.
- As an RNN model does not work well with long sequences, **technical indicators** computed over longer periods of time (with different scales) can be used to improve the prediction of an RNN model over short sequences at no additional cost and should be utilized.
- State-of-the-art tools in the domain of finance prediction, such as GARCH and ARIMA, should be compared to verify the superiority of a proposed RNN model.

Our model takes technical indicators as covariates to improve model prediction, and thus its design can easily utilize other data sources when available, by extracting features form them to be passed as covariates into our model (more discussion will be given in Section 5). For now, let us assume that we want to predict the future price of a stock, and all the data we have is a time series of the historical prices of that stock. Our model has been used by quantitative analysts in Freddie Mac for over one quarter, and is found to be more effective than traditional GARCH variants in market prediction.

The rest of this paper is organized as follows. We first review RNN concepts and trade data analysis preliminaries in Section 2. Section 3 describes our RNN model to predict stock prices and compares it agains prior RNN models, and Section 4 reports the experimental results. Finally, we review related work in Section 5 and conclude our paper in Section 6.

2 PRELIMINARIES

We assume readers are already familiar with RNN and deep learning concepts, and we now provide a brief review of some key concepts to help readers get familiar with our notations.

RNN. Figure 1 shows an RNN which takes an input sequence $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$ and predicts an output sequence $(y_1, y_2, \ldots, y_T)$. Here, each $\mathbf{x}_t$ is a vector of input features at time t, such as a word (in a sentence) that is embedded into a vector space, or the price of a stock at time t (the vector has only one element in this case). Each y_t is the predicted output at time t. For regression, y_t is a scalar like the predicted next stock price; let r_t be the actual next stock price, then we use a loss function $\ell_t = (y_t - r_t)^2$ to measure the error at time t. For classification, $\mathbf{y}_t$ is a score vector $(s_1, s_2, \ldots, s_k)$ where s_i is the score of the i-th class; let the true class be i, then loss ℓ_t is computed as the cross-entropy loss between $\mathbf{y}_t$ and the one-hot encoding of Class i, which equals $\ell_t = -\log s_i$. The overall loss of the training/validation data sequence $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$ and $(y_1, y_2, \ldots, y_T)$ is computed as the average loss $\mathcal{L} = \frac{1}{T}\sum_t \ell_t$, and the loss of a set of training mini-batch or a validation set (i.e., a set of labeled sequences) is computed as the average value of $\mathcal{L}$ in the set.

In Figure 1, a recurrence formula $\mathbf{h}_t = f_W(\mathbf{h}_{t-1}, \mathbf{x}_t)$ is applied at every time t, where $\mathbf{h}_t$ is the hidden state at time t which captures all the information up to time t as it sees $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_t)$. Bidirectional RNN is also applicable and is popular in NLP, but we do not consider it here since we are studying the price prediction problem and the time is one-way (unlike, for example, sentiment analysis for an entire sentence). In a vanilla RNN, we have $f_W = \tanh(W_{hh}\mathbf{h}_{t-1} + W_{xh}\mathbf{x}_t)$, and $\mathbf{y}_t$ is read out from the hidden state $\mathbf{h}_t$ as $\mathbf{y}_t = W_{hy}\mathbf{h}_t$. Note that $W = (W_{hh}, W_{xh})$, since $\mathbf{h}_t = f_W = \tanh(W\mathbf{v}_t)$ where $\mathbf{v}_t = (\mathbf{h}_t, \mathbf{x}_t)^T$.

We call the RNN in Figure 1 as a many-to-many RNN, since it emits an output $\mathbf{y}_t$ at every time t. Since all the recurrence steps share the same recurrence parameter W and the same read-out parameter W_{hy}, the loss ℓ_t at every time t will adjust the values of W and W_{hy} during back-propagation.

Such a strong supervision may not be possible in some applications like sentiment analysis of a sentence, where only one output (positive/negative) is available after reading the entire sequence of words in a sentence. In such a case, a many-to-one RNN is used where the output is only y_T and the loss function is defined only over y_T.

For ease of presentation, we will simplify the RNN in Figure 1 into the RNN diagram in Figure 2(a), where $\mathbf{h}_0$ is omitted since it is usually initialized as a zero state.

RNN Variants. If we treat the output $(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T)$ as intermediate features denoted as $(\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T)$, we can stack another layer of RNN on top that takes $(\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T)$ as the input sequence, which gives the stacked RNN in Figure 2(b) with 2 hidden layers that can be depicted by:

$$\begin{aligned} \mathbf{h}_t^{(1)} &= f_{W^{(1)}}(\mathbf{h}_{t-1}^{(1)}, \mathbf{x}_t), \quad & \mathbf{o}_t &= W_{h^{(1)}o}\mathbf{h}_t^{(1)}, \\ \mathbf{h}_t^{(2)} &= f_{W^{(2)}}(\mathbf{h}_{t-1}^{(2)}, \mathbf{o}_t), \quad & \mathbf{y}_t &= W_{h^{(2)}y}\mathbf{h}_t^{(2)}. \end{aligned}$$

More LSTM layers can be stacked on top to extract higher-level features before we use them to generate the output.

Vanilla RNN suffers from the vanishing gradient problem during back-propagation: the gradients shrink as they back-propagate through time, to the extent that the value is too small to contribute enough to the stochastic gradient descent based model training. To overcome this problem, the vanilla RNN unit can be replaced with other units including LSTM [10] and GRU [5] which utilize internal mechanisms called gates to regulate the flow of information, so that the RNN model can process longer sequences. As a result, in the RNN models presented in rest of this paper, LSTM is used instead of vanilla RNN. Since LSTM and GRU exhibit the same external interface as vanilla RNN, we skip their introduction.

Convergence of training can be further improved by performing batch normalization [11] over the mini-batch input before ReLU activation along the vertical (i.e., stacking) direction, and by performing layer normalization [1] over the sequence input before tanh activation along the sequence (i.e., time) direction.

High-frequency trade data are now readily available for intraday market research. For example, the Trade and Quote (TAQ) database available at the Wharton Research Data Services (WRDS) website provides stock data including price, number of shares, and time of each transaction (to the nearest second). On each weekday, the market opens at 9:30 AM ET and ends at 4:00 PM ET, and the market data in every second is recorded.

However, market microstructure noises are a major hurdle towards the price analysis when the sampling frequency is too high since the realized volatility will not be stable. As a well-established conclusion, [19] shows that a sampling frequency of 5 minutes or longer is a must for stable analysis, and thus most work on high-frequency financial data samples the data into a time series of 5-minute unit or longer.

Instead of subsampling at a frequency of 5 minutes, we adopt the approach by [7] which captures richer price information in each 5-minute block. Specifically, we first aggregate trade data from the TAQ database into 1-second blocks, remove trades that were canceled or otherwise flagged as illegitimate (using TAQ's condition codes) and eliminate bouncebacks using an influence statistic, following the exact steps described in Section 2 of [7]. Then, given all the cleansed (price, share) pairs in a 5-minute block, we use the *median share-price* as a representative price for that block, which is the median price per share treating each share traded as a separate observation. Besides the median share-price, we also extract additional features from the trades of each block, such as the lowest and highest prices. Section 3 will provide more details.

Statistical Method for Time Series Forecasting. AutoRegressive Integrated Moving Average (ARIMA) is a class of widely-used models that capture a suite of different standard temporal structures in time series data, and it has 3 hyperparameters. We denote a specific ARIMA model as ARIMA(p, d, q). However, in finance data, a change in volatility over time can cause problems when using ARIMA.

In contrast, the Autoregressive Conditional Heteroskedasticity (ARCH) method provides a way to model a change in variance in a time series that is time dependent, such as increasing or decreasing volatility. An extension of this approach named Generalized ARCH (GARCH) further allows the method to support changes in the time-dependent volatility, and is a popular tool used by finance practitioners [3]. GARCH has 2 hyperparameters, and we denote a specific model by GARCH(p, q). Since GARCH is the state of the art in finance data analysis, it is our major competitor.

Evaluating Price Direction Prediction. Many works in stock price prediction study the stock movement directions rather than predicting the actual future prices, possibly because of the need of data sparsification (into 5-minute blocks or more) due to market microstructure noises. In this binary classification context, we typically translate the score s returned by a machine learning model (such as logistic regression (LR) or neural networks) about price rising by a sigmoid function $P(s) = 1/(1 + e^{-s})$. A natural idea is to consider the price as rising if $P(s) \geq 0.5$, and as dropping otherwise.

However, it is well known that in a high-frequency setting, a large number of time intervals are characterized by zero returns, meaning that class "Drop" (or 0) is overpopulated in comparison to class "Rise" (or 1); the ratio is on average 54%:46% [4]. Due to this class imbalance phenomenon, prediction accuracy alone is a biased evaluation metric that favors class "Drop" (or 0). Thus, a receiver operating characteristic (ROC) curve is typically used to evaluate the price direction prediction quality where instead of using $\tau = 0.5$ to specify the classification decision boundary $P(s) \geq \tau$, we vary τ from 0 to 1 and compute the (true positive rate, false positive rate) pairs along the way to plot such an ROC curve (see Figure 3 for an illustration), and the area under the curve (or simply, AUC) measures how good the model prediction is.

3 APPROACH

Since high-frequency trade data need to be sparsified into 5-minute blocks or more to mitigate market microstructure noises, most existing works study the binary classification problem of price direction prediction as follows: given the historical prices $x_{t-k+1}, \ldots, x_{t-1}, x_t$ of k consecutive 5-minute blocks, predict whether the price will rise or drop in the next 5-minute block at time $(t + 1)$.

Prior LSTM models. [15] is probably the first work that applies LSTM network to price movement prediction. It considers 15-minute blocks rather than 5-minute blocks, and for each block, the features $\mathbf{x}_t$ include the opening/closing/highest/lowest prices and trade volume. On top of the price data, a set of 175 technical indicators is generated using the TA-Lib library (though details are not provided), which are used to expand $\mathbf{x}_t$ into a vector of 180 features. The class label y_t is defined as 1 if the closing price of the current

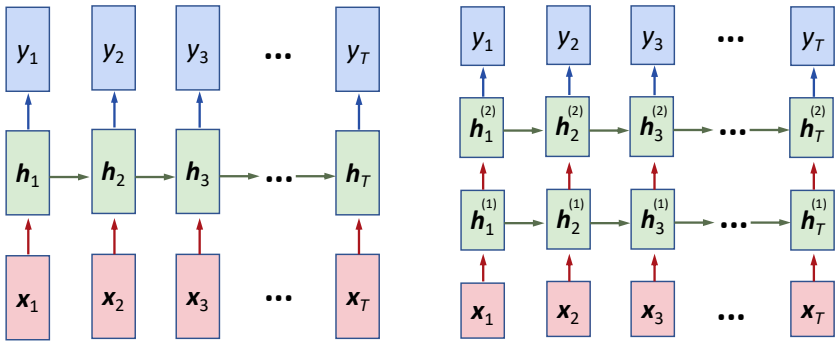

Figure 2: Simplified RNN Diagram

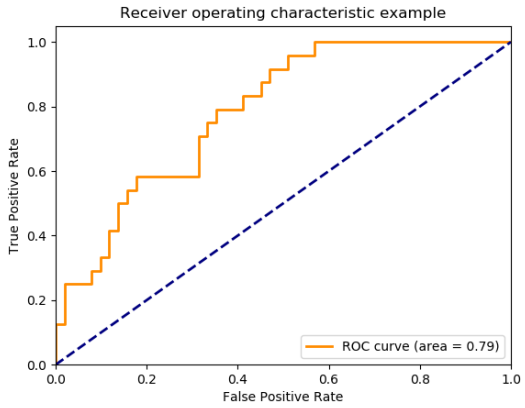

Figure 3: ROC Curve Illustration

15-minute block is less than that of the next block, and 0 otherwise. A many-to-one LSTM network is used which predicts y_t from $\mathbf{x}_{t-k+1}, \ldots, \mathbf{x}_{t-1}, \mathbf{x}_t$ where $k = 20$, and binary cross-entropy loss is used to train the model. However, the reported accuracy for next direction forecast is merely 53%-55% which is not much better than a random guess.

More recently, [4] further improves upon this model by using an ensemble of LSTM networks. The difference is that [4] works with 5-minute blocks, uses a stacked LSTM network with 2 hidden layers and input sequence of length 5, and trains 12 such LSTM networks with the final prediction being decided by a weighted sum where the weight of each individual model is proportional to its recent performance measured by AUC. The work compared with machine learning baselines such as lasso and ridge logistic regressions, as well as an equally weighted LSTM ensemble, and shows that their AUC-weighted LSTM ensembles achieves the highest AUC in the range of 0.513–0.536 which is not much better than the AUC of a random guess (i.e., 0.5).

On the other hand, the work lists the detailed features used in $\mathbf{x}_t$ including a comprehensive list of technical indicators and their parameters as shown in Table A3 and Appendix B of [4] which provide a valuable source for reference by later studies. They also use the technical indicators at the sector/index level as features in $\mathbf{x}_t$ to consider effect of similar stocks, as well as a list of basic features computed from the trades in each 5-minute block as shown in their Table A2. Other features used in $\mathbf{x}_t$ include price predictions from rolling regressions (computed as AR(1) over 1 month of observed opening/closing/highest/lowest prices) and the target class probabilities $P(y_t)$ and $P(y_t|y_{t-1})$.

Our Model. As a proof of concept, this work only considers the use of the historical prices of a stock to predict the future price of that stock, though other features such as sector-level technical indicators (which consider other similar stocks) can be straightforwardly used to expand $\mathbf{x}_t$ to improve prediction.

Figure 4(a) shows the many-to-one RNN model used by existing work, where each feature vector $\mathbf{x}_t$ contains features including the opening/closing/highest/lowest prices, trade volume, and technical indicators; and y_T predicts if the price will rise in time $(T + 1)$. There are two problems: (1) the label y_T is to predict the indicator function $\mathbf{1}\{x_{T+1} - x_T > 0\}$ which is a weak supervision: even though the error $(x_{T+1} - x_T)$ could be large, the training regards the current model parameter values as good as long as the direction is correct. For example, let $x_T = 100$ and even though $x_{T+1} = 108$, the training will consider the model parameters to be good even when they will lead to a prediction of $x_{T+1} = 101$, and another model parameter setting that leads to a prediction of $x_{T+1} = 106$ will not be considered much better. Moreover, the lack of the magnitude in prediction will render the model useless for making intraday trading decisions, as a big drop in the next 5 minutes is a strong signal to sell but a mild drop is not due to transaction fees and the possibility of price rising in subsequent 5-minute blocks. **We propose to model the problem as a regression one by using the mean squared error (MSE) as the loss function which directly tries to minimize the difference between the predicted price y_T and the actual price x_{T+1}.** Since we have the actual value of x_{T+1} in each training sequence, we should use it rather than the vague $\mathbf{1}\{x_{T+1} - x_T > 0\}$ to provide a strong supervision for model training. Our price regression model can still be used to predict the price direction as $\mathbf{1}\{y_T - x_T\}$, and so there is no need to make the model itself a classification one that admits only weaker supervision.

Another problem with the model in Figure 4(a) is that only the price x_{T+1} is used to adjust model parameters by back-propagation, while previous prices $x_2, \ldots, x_T$ are not utilized. This is probably because existing work model the price of a 5-minute block itself as a vector of opening/closing/highest/lowest prices rather than a single price.

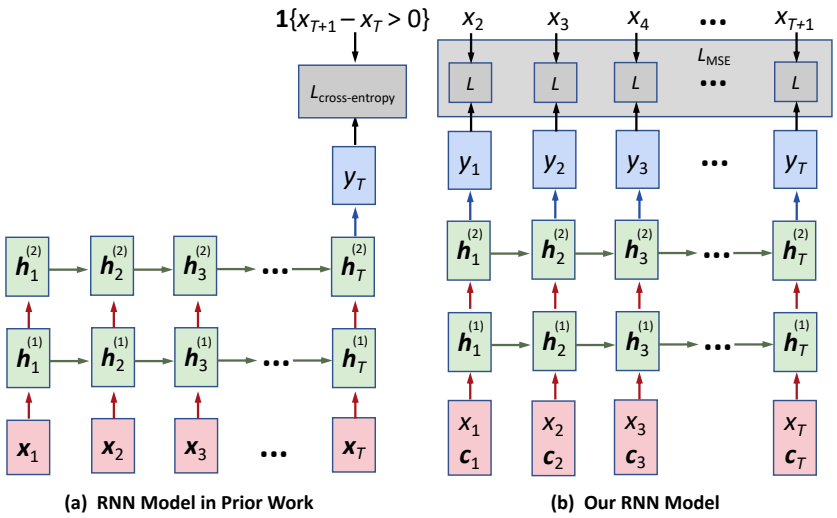

Figure 4: RNN Model Comparison

Recall that we use the **median share-price** of each 5-minute block t as its representative price, denoted by x_t. This design allows us to use an **autoregressive** many-to-many LSTM network as shown in Figure 4(b) such that the error at each 5-minute block in a price sequence contributes to the adjustment of the model parameters. In our case, y_t is predicting the value of x_{t+1} directly rather than $\mathbf{1}\{x_{t+1} - x_t\}$, and the errors in all the T time steps are used to compute the MSE loss. Each input vector $\mathbf{x}_t = (x_t, \mathbf{c}_t)$ where $\mathbf{c}_t$ is a vector of **covariates** such as technical indicators at time t which provides additional features than the median share-price x_t itself to improve prediction. This work only considers features that are derived from historical prices of a stock as covariates, but other features can also be integrated into $\mathbf{x}_t$ to further improve the prediction as we will describe in Section 5.

We use 15 **basic features** in $\mathbf{c}_t$ which are extracted from the trades within each 5-minute block t. These basic features are exactly those listed in Table A2 of [4] except the number of market sweep trades which we do not have access to. The basic features include (1) price-related features such as the opening/closing/highest/lowest prices, volume-weighted average price, maximum price difference, etc., and (2) volume-related features such as the total number of trades, the mean trade size, positive/negative volume (i.e., sum of volume of those trades with price rising/dropping in the next second), etc.

We also include all the **technical indicators** listed in Table A3 of [4] into our in covariate vector $\mathbf{c}_t$, which can be categorized into different categories such as those on return v.s. risk (e.g., Sharpe ratio), those on price and volume flow (e.g., accumulation/distribution index), those on volatility (e.g., Bollinger Band) and those on overbought/oversold signals (e.g., stochastic oscillator, relative strength index, commodity channel index, money flow index). We refer readers interested in these well-engineered indicators to Appendix B of [4] for a comprehensive introduction of these indicators, but we remark that they all can be computed from the historical prices within a sliding window up to the current time-block t. We follow [4] exactly to use different window sizes including $n = \{6, 12, 36, 78, 234\}$, where the unit of n is 5-minute block and thus 78 refers to an entire day (9:30 AM to 4:00 PM ET). This allows us to capture the signals in different time scales ranging from 30 minutes to 3 days.

We remark that even with LSTM to mitigate the vanishing gradient problem, the network still cannot process very long sequences. Our extensive tests show that a sequence length of 7 (5-minute blocks) delivers the best performance, which translates to 35 minutes of historical price data. Therefore, the basic features provide richer information inside each 5-minute block, while the technical indicators capture signals in longer terms up to 3 days. In other words, covariates in $\mathbf{c}_i$ are complementary to the median share-price of the 7 blocks that are inputted to the LSTM network, even though the temporal information of the latter are already accumulated by the LSTM network. Our experiments in Section 4 confirms that the covariates improves the AUC of our LSTM model.

We also remark that our LSTM model can be used to predict the median share-price into multiple future 5-minute blocks, since our autoregressive model is intrinsically generative. Let the current time-block be T, and our model predicts $y_T = \hat{x}_{T+1}$. Then, since we now have the prices all the way up to $(T+1)$, we can compute the various technical indicators for time-block $(T+1)$ to construct $\mathbf{c}_{T+1}$, so that $\mathbf{x}_{T+1} = (\hat{x}_{T+1}, \mathbf{c}_{T+1})$ can be used to predict $y_{T+1} = \hat{x}_{T+2}$, and the process may continue to generate more predictions into the future (though the accuracy may drop). One remaining issue is that the basic features of $\mathbf{c}_{T+1}$ cannot be computed as the trades in time-block $(T+1)$ have not been seen yet, but we can either exclude the basic features from model covariates, or set them all to be equal to the predicted $\hat{x}_{T+1}$.

Finally, we remark that our price regression model is not only a more accurate price direction predictor, but also a promising tool for intraday trading since we estimate the "median" price in the next 5 minutes: there is a good chance that the price will go beyond the predicted median in the next 5 minutes at which time one can sell the shares bought in the current 5 minutes (we buy them since we see there would be a big price rise as predicted by our model).

Table 1: AUC Scores of Algorithms Without Covariates

Stock Code	m1-sig	mm-sig	m1-mse	mm-mse	arima	garch
AA	0.511	0.524	0.58	0.611	0.602	0.629
C	0.507	0.519	0.579	0.609	0.605	0.613
DIS	0.511	0.513	0.576	0.613	0.599	0.614
IBM	0.513	0.526	0.566	0.58	0.597	0.61
KO	0.5	0.511	0.584	0.63	0.604	0.622
MSFT	0.504	0.512	0.569	0.599	0.587	0.614
UTX	0.513	0.521	0.571	0.601	0.601	0.611
CAT	0.505	0.522	0.576	0.599	0.601	0.617
DD	0.509	0.519	0.571	0.592	0.594	0.611
GE	0.511	0.522	0.564	0.589	0.591	0.609

Table 2: AUC Scores of Algorithms with Covariates

Stock Code	lasso	ridge	m1-sig-idx	mm-sig-idx	m1-mse-idx	mm-mse-idx
AA	0.52	0.518	0.559	0.577	0.621	**0.693***
C	0.518	0.517	0.554	0.599	0.611	**0.682***
DIS	0.515	0.52	0.549	0.602	0.596	**0.668***
IBM	0.521	0.518	0.559	0.611	0.605	**0.689***
KO	0.518	0.521	0.56	0.599	0.609	**0.691***
MSFT	0.511	0.517	0.562	0.611	0.611	**0.688***
UTX	0.519	0.514	0.551	0.609	0.618	**0.701***
CAT	0.516	0.516	0.549	0.608	0.601	**0.679***
DD	0.517	0.519	0.56	0.61	0.594	**0.688***
GE	0.514	0.513	0.552	0.608	0.597	**0.687***

4 EXPERIMENTS

To evaluate the performance of our model and compare it with other models, we use the TAQ database accessible through the Wharton Research Data Services (WRDS) interface. We evaluate the models over 10 large-cap US stocks as they have high liquidity. Their stock codes are AA, C, DIS, IBM, KO, MSFT, UTX, CAT, DD and GE, and the stocks are selected to cover all kinds of sectors such as food, entertainment, IT, finance, manufacturing, etc. We use the high-frequency trade data in Year 2014 and cleanse the data as described in Section 2. To avoid being affected by the seasonal effect, for each quarter, the first 2 months are selected into the training set, the next 2 weeks are selected into the validation set, and the last two weeks are selected into the test set.

After extensive experimentation, we find that the best-performing LSTM networks are achieved by 3 stacked layers with batch normalization and a sequence length of 7. The LSTM hidden state vectors have size 64-32-16 bottom up, i.e., $|\mathbf{h}_i^{(1)}| = 64$, $|\mathbf{h}_i^{(2)}| = 32$, and $|\mathbf{h}_i^{(3)}| = 16$. Each LSTM network is trained with Adam [13], where the batch size is 64, learning rate is 10^{-3}, and the training converges in 30 iterations. For the statistical tools in finance, we find that the best-performing ones are ARIMA(2, 0, 2) and GARCH(1, 1) whose results are reported in our experiments.

We evaluate the models using 2 problems: (1) predicting the price direction of the next 5-minute block, and (2) predicting the median share-price of the next 5-minute block.

Price Direction Prediction. This is a binary classification problem. Similar to [4], we include logistic regression (LR) as a "shallow learing" baseline classifier which takes $\mathbf{x}_t = (x_t, \mathbf{c}_t)$ and predicts whether y_t is 1 (rising) or 0 (dropping). Even though LR does not consider historical prices, the technical indicators in $\mathbf{c}_t$ are computed from historical prices. We use the same two LR variants as in [4], lasso LR (i.e., with ℓ_1-regularization on weight parameters) and ridge LR (i.e., with with ℓ_2-regularization on weight parameters).

Recall that we can also use regression model and then compute the class as $\mathbf{1}\{\hat{x}_{T+1} - x_T > 0\}$. Therefore, we also use the best statistical methods ARIMA(2, 0, 2) and GARCH(1, 1) which only uses historical prices rather than covariates. To compute an ROC curve, we apply a sigmoid function over $(\hat{x}_{T+1} - x_T)$ to calculate the probability that the price rises at time $(T + 1)$.

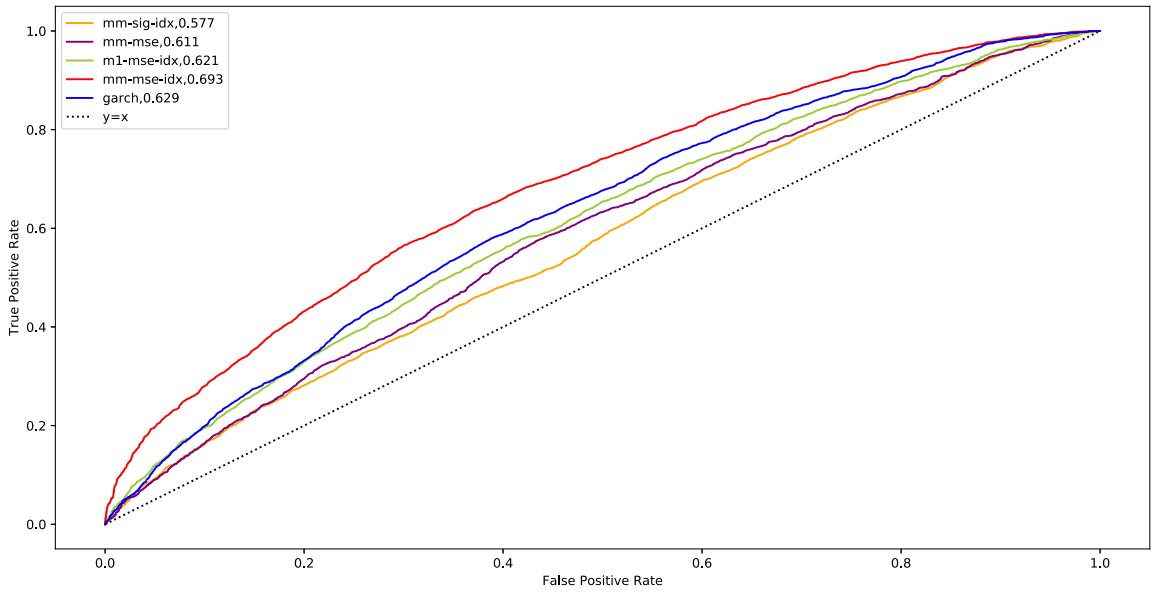

Figure 5: ROC Curves of Competitive Algorithms on AA

Table 3: Root Mean Square Error (RMSE) of Price Prediction

Stock Code	arima	garch	m1-mse	mm-mse	m1-mse-idx	mm-mse-idx
AA	0.06	0.05	0.05	0.05	0.03	**0.009***
C	0.06	0.06	0.07	0.05	0.04	**0.01***
DIS	0.07	0.05	0.06	0.05	0.03	**0.018***
IBM	0.05	0.06	0.06	0.04	0.03	**0.009***
KO	0.06	0.06	0.07	0.04	0.02	**0.009***
MSFT	0.05	0.04	0.05	0.05	0.03	**0.011***
UTX	0.06	0.04	0.07	0.04	0.02	**0.008***
CAT	0.08	0.07	0.08	0.05	0.03	**0.012***
DD	0.07	0.07	0.06	0.04	0.03	**0.01***
GE	0.07	0.06	0.06	0.04	0.04	**0.015***

For LSTM models, we have 3 dimensions to specify: (1) many-to-many (abbr. mm) or many-to-one (abbr. m1), (2) loss is sigmoid cross-entropy (abbr. sig) or mean squared error (abbr. mse), and (3) whether covariates are utilized (i.e., $\mathbf{x}_t = (x_t, \mathbf{c}_t)$ or only x_t) and if so, we append the algorithm name with "idx". We thus have $2 \times 2 \times 2 = 8$ LSTM variants. Our model is mm-mse-idx, while m1-sig-idx is used in prior works [4, 15].

Table 1 reports the AUC of all algorithms that take historical prices alone as the input without using covariates $\mathbf{c}_t$, and Table 2 reports the AUC of all algorithms that take $\mathbf{x}_t = (x_t, \mathbf{c}_t)$ as the input. Key observations are as follows:

- Our LSTM model mm-mse-idx has the best performance with AUC reaching up to 0.7, beating the second best which is GARCH by a large margin. This shows that a combination of many-to-many autoregressive design, MSE regression and the use of basic features and technical indicators as covariates together can deliver an excellent performance much better than prior solutions.
- Even with one of the 3 designs missing, an LSTM model cannot beat the state-of-the-art GARCH method already widely used by finance practitioners, even though GARCH does not look at covariates. This means that GARCH (AUC > 0.6) actually well beats prior works. Also, ARIMA is only slightly worse than GARCH.
- Using covariates improve the AUC of LSTM models.

To visualize the impact of each of the 3 designs in improving LSTM performance, we plot the ROC curves of competitive algorithms for the stock AA in Figure 5. The results of other stocks are similar and omitted due to space limitation. As we can see, our model mm-mse-idx is much better than GARCH, which is better than m1-mse-idx, followed by mm-mse, followed by mm-sig-idx. Thus, using binary cross-entropy loss is the biggest performance killer, followed by not using technical indicators (and other covariates), followed by not using an autoregressive architecture. This observation is quite consistent across our 10 stocks from diversified sectors.

Median Share-Price Prediction. Note that we can also directly consider price regression for predicting the median share-price in the next 5 minutes, which is useful for intraday trading. In this case, we compare all our 4 LSTM variants that use MSE as the regression loss, as well as the statistical methods GARCH and ARIMA. The RMSEs of these algorithms on our test data are reported in Table 3, where we see that our LSTM model mm-mse-idx has an RMSE one digit smaller than the others, which is pretty accurate. Also m1-mse-idx has a smaller error than mm-mse, which indicates that using covariates is more important than using a many-to-many model, which echos our observations in Figure 5.

Finally, GARCH and ARIMA are not as competitive here, which aligns with [6]: GARCH has the "data-stretch" problem that short-term fluctuations of volatility tend to disappear when sampling interval gets large. In contrast, the "leverage effect" for stocks says that price and volatility move in opposite directions empirically, so it is not surprising that GARCH predicts better on price movements.

5 RELATED WORK

Autoregressive LSTM networks have been used for inventory prediction by Amazon in their DeepAR model [8] where instead of directly predicting a price y_t, LSTM output predicts the parameters of a negative binomial distribution from which y_t is sampled as a positive (inventory) count. DeepAR also proposes to let LSTM output predict the parameters of a Gaussian distribution from which to sample a continuous value for y_t (e.g., a stock price), and the Gaussian distribution can account for the market microstructure noises. However, we do not adopt this model since the additional distribution layer before y_t slows down training and prediction, and when generating future price sequences for prediction, DeepAR has to sample a price at each time step leading to nondeterministic outputs so that many sampling passes are needed to estimate a stable prediction. Our solution to market microstructure noises is to cleanse and sparsify the high-frequency trade data using well-established finance practices before training our LSTM network. DeepAR has also been improved by replacing the distribution that generates y_t with spline quantile functions and trained with Continuous Ranked Probability Score (CRPS) as the loss [9]; and by fusing state space models (e.g., ARIMA) to enforce temporal smoothness [17].

Besides using covariates extracted from historical prices as we do, a number of works have explored the extraction of other features when other data sources are available. [4] also utilizes the technical indicators of other stocks in the same sector as covariates, while [16] constructs temporal stock correlation networks from historical stock quantitative data, over which 5 topological mesoscale indicators are computed to improve prediction. [18] further constructs two correlation-based networks from social media and financial data, respectively, which can be combined for prediction; while [14] extracts mood scores about DJIA by sentiment analysis over tweets to improve prediction. [20] extracts features from limit order books (where more than 90% of orders end in cancellation rather than matching) using convnets for use by LSTM. However, all these works consider the prediction as a binary classification problem rather than price regression.

Among other works, [12] extracts image features from candlestick stock charts using convnets to enhance the temporal features learned by LSTM, while [2] decomposes the stock price time series by wavelet transforms to eliminate noise, followed by extracting deep high-level features using stacked autoencoders, and the features are then fed into LSTM for price forecast. We remark that our model is compatible with these improvements, as the extracted features can be fed into our model as covariates.

6 CONCLUSION

This work corrects the poor use of LSTM networks in recent studies, and provides "the" baseline LSTM network (i.e., mm-mse-idx) for future work to compare with. Our model can be easily extended with additional features extracted from other data sources, and beats GARCH by a large margin.

Acknowledgments. This work was partially supported by NSF DGE-1723250 and OAC-1755464.

REFERENCES

[1] Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.
[2] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944, 2017.
[3] Tim Bollerslev. Financial econometrics: Past developments and future challenges. *Journal of econometrics*, 100(1):41–51, 2001.
[4] Svetlana Borovkova and Ioannis Tsiamas. An ensemble of lstm neural networks for high-frequency stock market classification. *Journal of Forecasting*, 2019.
[5] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *SSST@EMNLP*, pages 103–111, 2014.
[6] Feike C Drost and Theo E Nijman. Temporal aggregation of garch processes. *Econometrica: Journal of the Econometric Society*, pages 909–927, 1993.
[7] Bryan Ellickson, Miao Sun, Duke Whang, and Sibo Yan. Estimating a local heston model. *SSRN 3108822*, 2018.
[8] Valentin Flunkert, David Salinas, and Jan Gasthaus. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *CoRR*, abs/1704.04110, 2017.
[9] Jan Gasthaus, Konstantinos Benidis, Yuyang Wang, Syama Sundar Rangapuram, David Salinas, Valentin Flunkert, and Tim Januschowski. Probabilistic forecasting with spline quantile function rnns. In *AISTATS*, pages 1901–1910, 2019.
[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.
[12] Taewook Kim and Ha Young Kim. Forecasting stock prices with a feature fusion lstm-cnn model using different representations of the same data. *PloS one*, 14(2):e0212320, 2019.
[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
[14] Anshul Mittal and Arpit Goel. Stock prediction using twitter sentiment analysis. *Standford University, CS229*, 15, 2012.
[15] David M. Q. Nelson, Adriano C. M. Pereira, and Renato A. de Oliveira. Stock market's price movement prediction with LSTM neural networks. In *IJCNN*, pages 1419–1426, 2017.
[16] Zi Qi, Zhan Bu, Xi Xiong, Hongliang Sun, Jie Cao, and Chengcui Zhang. A stock index prediction framework: Integrating technical and topological mesoscale indicators. In *IRI*, pages 23–30, 2019.
[17] Syama Sundar Rangapuram, Matthias W. Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. In *NeurIPS*, pages 7796–7805, 2018.
[18] Thársis TP Souza and Tomaso Aste. Predicting future stock market structure by combining social and financial network information. *Physica A: Statistical Mechanics and its Applications*, 535:122343, 2019.
[19] Lan Zhang, Per A Mykland, and Yacine Aït-Sahalia. A tale of two time scales: Determining integrated volatility with noisy high-frequency data. *Journal of the American Statistical Association*, 100(472):1394–1411, 2005.
[20] Zihao Zhang, Stefan Zohren, and Stephen J. Roberts. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Trans. Signal Processing*, 67(11):3001–3012, 2019.