Forecasting Stock Prices Using Stock Correlation Graph: A Graph Convolutional Network Approach

Xingkun Yin
Faculty of Arts & Science
University of Toronto
troy.yin@mail.utoronto.ca

Da Yan
Department of Computer Science
University of Alabama at Birmingham
yanda@uab.edu

Abdullateef Almudaifer
Department of Computer Science
University of Alabama at Birmingham
lateef11@uab.edu

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

Yang Zhou
Department of Computer Science and Software Engineering
Auburn University
yangzhou@auburn.edu

2021 International Joint Conference on Neural Networks (IJCNN) | 978-1-6654-3900-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/IJCNN52387.2021.9533510

***Abstract*—Accurate forecasting of stock prices plays an important role in stock investment. With the advancement of AI in Fintech applications, various deep learning models have recently been developed for stock price forecasting. However, these models focus on designing sequence models to capture the temporal dependence from a stock's historical prices (and other information such as technical indicators and news), leaving the information from similar stocks underexplored.**

To fill this gap, we propose a novel deep learning approach for stock price forecasting, which builds and uses a stock correlation graph G where nodes are stocks and edges connect highly price-correlated stocks. Our model combines the graph convolutional network (GCN) and gated recurrent unit (GRU). Specifically, the GCN is used to extract features from the price of each stock and the prices of those highly similar stocks in G. For each stock, a sequence of these extracted features are then fed into a GRU model to capture temporal dependence. The model training follows the idea of multi-task learning, where each task learns its unique RNN-based sequence predictor for one stock, but all stocks share a common GCN module to improve GCN training to more effectively propagate correlation-related market signals. Our extensive experiments on real stock price data demonstrate that our approach consistently outperforms a GRU baseline that does not consider similar stocks during prediction, which verifies the effectiveness of using a stock correlation graph.

I. Introduction

AI has gained significant momentum in adoption in the area of Fintech thanks to the advancement of deep learning, which allows simple end-to-end model training with much higher accuracy than traditional machine learning models. Predictive models are being actively 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 machine learning approaches. In Section II-B, we will review a series of works applying RNN for stock prediction.

However, these works do not consider the similarity of those stocks with highly-correlated price trends to improve their prediction, such as stocks in the same industry sectors or targeting similar markets. After examining the historical price curves of various stocks, we find that such similarity is prevalent and thus valuable in boosting the prediction accuracy. For example, Figure 1 shows an example of three stocks where the Pearson's correlation coefficient of any two price sequences is at least 0.9. The time period of the stock price sequences in Figure 1 is 2010-11-22 to 2018-07-05, which accounts for the first 80% of the total period that we consider (till 2020-06-04, only including trading days), and are the data used for the construction of a stock correlation graph, and for model training. While such strong correlations are prevalent among stocks (c.f. Section III-B), we only include three stocks in Figure 1 for ease of visualization. We can see that such correlations can provide a very strong market signal: for example, if the prices of most other highly-correlated stocks are rising, the target stock's price is very likely to rise.

The recent advancement of graph neural network models [26] provides a unique opportunity to utilize the stock correlations to improve prediction of sequence models (i.e., RNNs). In this paper, we build an undirected stock correlation graph $G = (V, E)$ from the historical price sequences of the stocks, where each node $v \in V$ corresponds to a stock, and two stocks v_1 and v_2 are connected by an edge (v_1, v_2) if their price sequences are highly correlated. We only keep high-quality connections to combat the over-smoothing problem [4], [17], [20], [27] of graph convolutional networks (GCNs) that might cause the performance to backfire.

At each time step, we then extract stock features from the price of each stock and the prices of those highly similar stocks in G, and for each stock, a sequence of these extracted features are then fed into an RNN model to capture temporal dependence for prediction. For training, each stock has its own RNN model but all stocks share a common GCN model. The goal of training is to minimize the overall prediction errors of all stocks, which collectively improve GCN parameters.

We choose to let each stock own its own RNN model since the time sequences can be quite different for different stocks, but we let all stocks share the same GCN model to improve the parameter training for knowledge propagation from similar

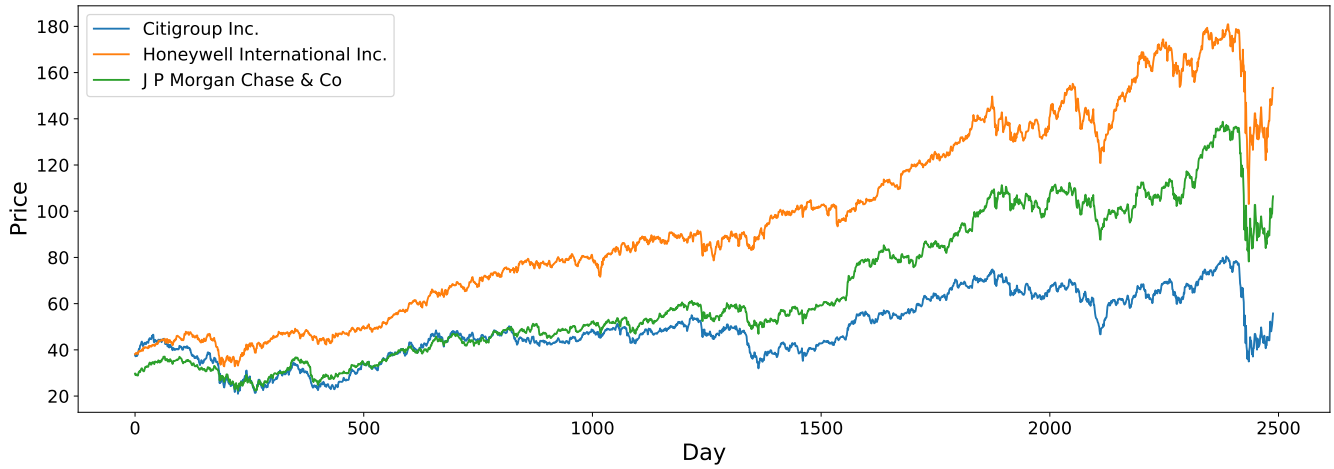

Fig. 1. Illustration of Strong Stock Price Correlations

stocks, following the idea of multi-task learning (where each task here is to predict the next price of one particular stock).

The major contributions of this paper are summarized as follows:

- We are the first to extract features from the stock correlations using GCN and to integrate the correlation-related features into an RNN-based sequence prediction model.
- Our model follows the idea of multi-task learning, where each task learns its unique RNN-based sequence predictor, but all stocks share a common GCN module to improve the GCN training to more effectively propagate correlation-related market signals.
- Extensive experiments were conducted to test the effect of correlation threshold (used for the construction of our stock correlation graph) on the prediction performance, and to compare our model with RNN baselines to verify the effectiveness of using a stock correlation graph.

The rest of this paper is organized as follows. Section II reviews the related work on graph neural networks, RNN models for stock forecasting, and recent efforts in combining GCN and temporal predictive models in other application domains. Then, Section III describes our model. Finally, Section IV reports our experiments and Section V concludes this paper with a discussion on the significance and impact of this work.

II. Preliminaries and Related Work

In this section, we first review the recent progress in graph neural network research in Section II-A. Then, Section II-B reviews the recent works on stock price prediction using recurrent neural networks. Finally, Section II-C reviews the recent efforts in combining GCN and temporal models in other application domains.

A. Graph Neural Networks

Earlier works such as DeepWalk [22] and node2vec [11] propose to embed nodes in a graph into vectors so that graph data can become admissible to deep learning tasks such as node classification and graph classification. However, these methods do not allow nodes to have their own feature vectors. Recently, graph convolutional networks (GCNs) have gained popularity thanks to a series of seminal works with the representative one being [16]. In a GCN, we use layers such as graph convolutions and graph poolings that directly operate on graphs (in contrast to tensors as in ConvNets), and the ultimate goal is usually for node classification or graph classification. We next review works on designing these layers, and recent efforts in fighting GCN's over-smoothing problem.

The notion of graph neural networks was initially outlined in [10] and further elaborated by [23] as a form of RNN. They propagate neighbor information in an iterative manner until reaching a stable fixed point. This process is expensive and is recently improved by several studies such as [18]. Encouraged by the success of ConvNets in computer vision, recent models focus on adapting these layers to work directly on graphs.

Graph convolution layers can be divided into two categories, spectral graph convolution and localized graph convolution [28]. Early works mainly focus on spectral graph convolutions, as pioneered by [3]. In a follow-up work, ChebyNet [7] defines graph convolutions using Chebyshev polynomials to remove the expensive Laplacian eigendecomposition. Then comes the state-of-the-art GCN [16] which further simplifies graph convolution by a localized first-order approximation without compromising performance, leading to a simple form of

$$H^{(\ell)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{\mathcal{A}}\tilde{D}^{-\frac{1}{2}}H^{(\ell-1)}W^{(\ell)}), \tag{1}$$

where $\tilde{\mathcal{A}} = \mathcal{A}+I$ is the adjacency matrix with self-connections added, $\tilde{D}$ is a diagonal matrix with $\tilde{D}_{ii} = \sum_j \tilde{\mathcal{A}}_{ij}$, $W^{(\ell)}$ is the weight matrix to train for Layer ℓ, σ is an activation function such as ReLU, and $H^{(\ell)}$ is the state matrix where each row keeps the features of a node at Layer ℓ; $H^{(0)}$ is the input matrix where each row keeps the features of a node.

Since spectral methods require operating on the entire graph Laplacian during training, which is expensive, some follow-up works endeavor to mitigate the cost as in FastGCN [5] and SGC [25]. To further avoid operating on graph Laplacian, GraphSAGE [13] proposes to learn a function that generates node embeddings by aggregating features from a node v's

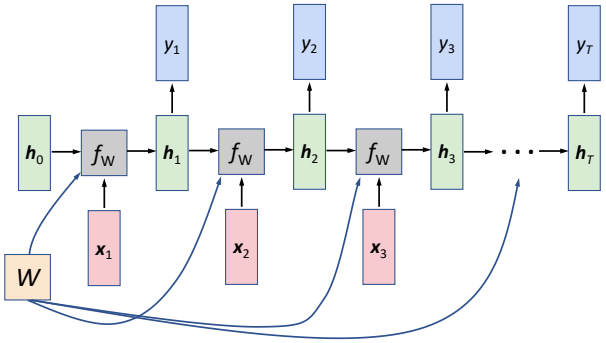

Fig. 2. A Many-to-Many Recurrent Neural Network

local neighborhood $\mathcal{N}(v)$. PinSage [28] further improves the performance of localized graph convolution by only incorporating top-k nodes that exert the most influence on node v into $\mathcal{N}(v)$ rather than using the full neighborhood, which is selected based on node visit frequencies of short random walks starting from v. The idea is similar to our binarization of A, i.e., to eliminate loosely correlated edges that introduce noise and cause over-smoothing.

GCN over-smoothing with many convolutional layers have recently gained a lot of attention. In fact, most existing graph convolution models as represented by [16] achieve their best performance with only 2 layers. This is also the case even when self-attention mechanism is used to weigh important neighbor-nodes higher as in GAT [24]. Several recent solutions have been proposed to tackle this problem. PairNorm [32] is a new normalization layer that prevents node embeddings from becoming too similar. Chen et al. [4] find that over-smoothness is caused by the low information-to-noise ratio of the message received by the nodes, propose two methods to alleviate the over-smoothing issue from the topological view. Chen et al. [20] and Yang et al. [27] further propose techniques to train deep GCN without over-smoothing, while Li et al. [17] propose both co-training and self-training approaches to train shallow GCNs to achieve better performance. We remark that our stock correlation graph only keeps the most correlated connections and is thus an effective over-smoothing reduction technique per se.

B. Stock Prediction with RNNs

Figure 2 shows a **recurrent neural network (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 the price of a stock (and some covariates at time t such as technical indicators). 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. 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 is computed as the average value of $\mathcal{L}$ in the set.

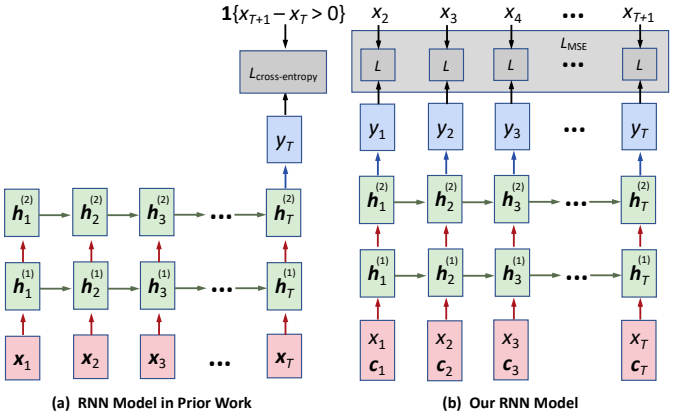

Fig. 3. RNN Models with Two Stacked RNN Layers

Since vanilla RNN suffers from the vanishing gradient problem during back-propagation, other recurrent units are proposed to mitigate this problem, such as long short-term memory (**LSTM**) [14] and gated recurrent unit (**GRU**) [6] which utilize internal mechanisms called gates to regulate the flow of information, so that the RNN model can process longer sequences.

The work by Nelson et al. [21] is probably the first work that applies LSTM network to price movement prediction. It considers 15-minute time blocks as time steps, 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, 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 15-minute block is less than that of the next block, and 0 otherwise. The work models the problem as a classification problem where $y_t = 1$ (resp. $y_t = 0$) means the price rises (resp. falls) in the next time step. A many-to-one LSTM network is adopted as illustrated in Figure 3, 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 sequence classification model.

Recently, [2] further improves upon this model by using an ensemble of LSTM networks. The difference is that [2] 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. Both works adopt a sequence classification model, and their performance is only slightly better than a random guess.

More recently, our prior works [12] conducted an extensive comparison among LSTM models and conclude that (i) a regression model should be adopted even for price movement classification since if we directly model the problem as a classification one, the training stops to close the gap between the actual price and the predicted price at time $(t+1)$ as long as the direction (rise or fall) is correct, and that (ii) a many-to-many model is superior to a many-to-one model, since the

gap between the actual and predicted prices at every step can be used to improve the training of LSTM parameter shared by all LSTM units (c.f. Figure 2). Figure 3(a) illustrates a many-to-one model for classification, while Figure 3(b) illustrates a many-to-many model for regression which is superior and thus adopted in this paper.

In [12], we used technical indicators as the covariates (besides prices) to input to LSTM, while in [8] we used stock news features extracted from tweets as the covariates. Other similar models have been proposed, but they use other kinds of covariates. Zhang et al. [31] extract features from limit order books (where more than 90% of orders end in cancellation rather than matching) using ConvNets for use by LSTM. Kim et al. [15] extract image features from candlestick stock charts using ConvNets to enhance the temporal features learned by LSTM. Bao et al. [1] decompose 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: we use features extracted from the prices of highly correlated stocks as covariates, but it is straightforward to expand our feature vector $\mathbf{x}_t$ to include other features.

C. Combining GCN and Temporal Predictive Models

The pioneering work of [16] has brought immense popularity of applying GCN to applications that involve graph data for predictive tasks. Probably the most actively studied application domain is geospatial data prediction where the underlying road network provides a natural backbone graph over which GCN can be applied. One typical problem is traffic prediction, the goal of which is to predict the future traffic volume/speed on a road segment or designated stations on the road network [19], [29], [33]. To our knowledge, no work has considered to apply GCN to capture stock correlations for stock price forecast, possibly due to the lack of an obvious backbone network such as a road network in traffic forecast. This paper makes a first attempt in this new application domain, and since the focus is on verifying whether our stock correlation graph and GCN integration to sequence models are effective, we adopt the state-of-the-art GCN (i.e., [16]) and GRU modules. However, we remark that our solution is orthogonal to how the GCN and sequence models are integrated, and the more integrated modules such as those for traffic prediction can be easily replaced into our approach as well.

III. Our Approach

This section presents our predictive models. Specifically, Section III-A first describes the datasets and data preprocessing. Then, Section III-B explains how we construct our stock correlation graph, and Section III-C presents our model built on top.

A. Datasets

Our stock data are purchased from kibot.com containing the price and trade volume of every stock in every minute till the current moment.

We consider the duration of 2010-11-22 to 2020-06-04, with the first 80% of the total period, i.e., 2010-11-22 to 2018-07-05, for constructing our stock correlation graph (including correlation computation) and for model training, and the rest for test.

As the pioneering work of [30] indicates, market microstructure noises are a major hurdle towards the price analysis when the sampling frequency is high since the realized volatility will not be stable. As a well-established conclusion, [30] shows that a sampling frequency of 5 minutes or longer is a must for stable analysis. We, therefore, first aggregate the minute-by-minute price sequences into 5-minute time blocks.

Instead of subsampling, we adopt the approach by [9] which better captures average price information in each time block. Specifically, we use the *median share-price* as a representative price for each time block, which is the median price per share treating each share traded as a separate observation.

In our dataset, there are even trades happening not on trading days (e.g., weekends) due to the existence of pre-market trades. We cleanse data by removing those rest days without frequent trades. Specifically, we remove those days with at least 20 5-minute time blocks that contain no trades, which captures almost all rest days, though some trade days where trades are not frequent are also eliminated.

Finally, the price sequences of 5-minute time blocks are integrated into a day-by-day time sequence. Without loss of generality, this paper studies the problem of predicting the stock median share-price of a future day from the price information of a sequence of k previous days. In other words, we use one day as the basic unit of a time block. This allows the price data to be more stable, ensuring the quality of price information propagation from correlated stocks, though our model can be easily used for a smaller time block size. In fact, most investors (excluding day traders) will not consider buying and selling their stocks on the same day or too frequently due to transaction fees, let alone some markets do not allow T+0 (e.g., in China). Our model provides predicted stock prices for a future day so that investors can get an intuitive feeling of how likely and how much the stock price will rise or fall to guide their investment decisions. Note that predicting median share-price means that the price is at least the predicted value for approximately 50% of the time in the day of prediction.

B. The Stock Correlation Graph

Two groups of investment products are studied. (1) **DOW**[1]: 43 stocks that are or were on the list of 30 companies that determine the Dow Jones Industrial Average (DJIA). (2) **ETF**[2]: the top-50 exchange-traded funds (ETFs).

While 43 stocks are listed for DOW, two stocks have their price sequences beginning from 2019 and are thus eliminated. Also, a stock PGP has 3,443 rest days after preprocessing, while all the others have only 1,090–1,109 rest days. This is

[1]http://www.kibot.com/historical_data/Dow_30_Historical_Intraday_Data.aspx

[2]http://www.kibot.com/Historical_Data/Top_50_ETFs_Historical_Intraday_Data.aspx

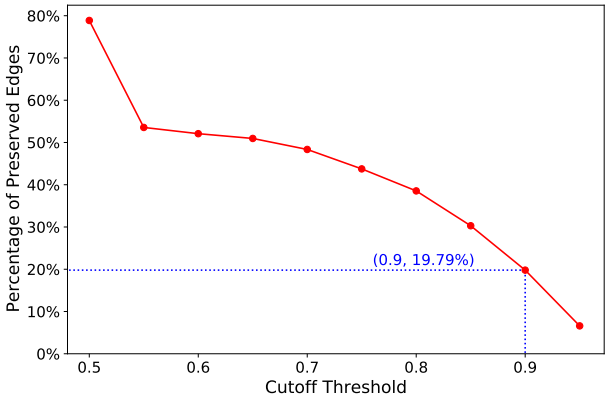

Fig. 4. Edge Percentage w.r.t. Cutoff on DOW

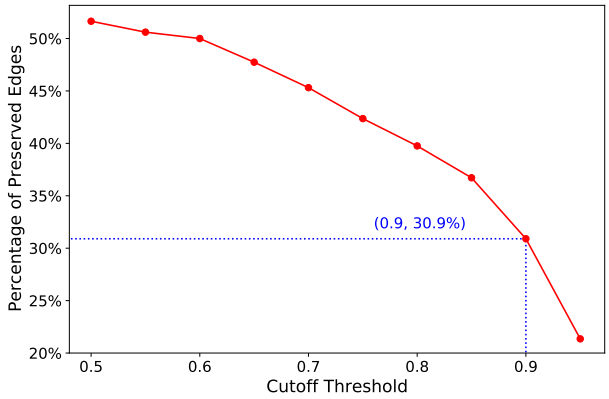

Fig. 5. Edge Percentage w.r.t. Cutoff on ETF

likely because PGP was not actively traded on some trade days which are judged as rest days by us, and therefore, we removed PGP as an outlier. This leaves only 40 stocks in **DOW** in total.

We similarly removed two outliers ERX and FXE from **ETF**, leaving 48 ETFs in total.

In each group, we compute the price-sequence correlation coefficient of every pair of stocks (or ETFs), forming a stock (or ETF) correlation matrix A where each element takes value within range $[-1, 1]$. To avoid over-smoothing, we binarize A with a correlation cutoff τ to keep only those elements with correlation $\geq \tau$, which gives the adjacency matrix of our correlation graph G. Recall that A in DOW has $40 \times 40 = 1,600$ elements (or edges, if we double-count (i, j) and (j, i) as the graph is undirected), while ETF has $48 \times 48 = 2,304$ elements.

Figure 4 (resp. Figure 5) shows the percentage of edges among the totally 1,600 (resp. 2,304) that are preserved by A after binarization using varying values of cutoff τ, where we can observe that the curve is initially flat but the slope increases with τ. We adopt $\tau = 0.9$ by default which is at the elbow of the curves, cutting off the flat part of the curve and keeping only high-correlation connections. Further increasing τ will cause a lot of high-correlation connections to be lost as the curve slope is steep. One exception is in DOW when the cutoff increases from 0.5 to 0.55, where we see a drastic drop in edge percentage; this is likely because of how these stocks are selected to determine DJIA (i.e., representative of the US stock market) where components bear a high base correlation (close to 0.5).

In Section IV-B we will also test our model with different values of τ to justify the choice of $\tau = 0.9$.

C. The Predictive Models

Our model follows the idea of multi-task learning, where each task learns a GRU-based sequence predictor for a particular stock, but all stocks share a common GCN module to improve the GCN training to more effectively propagate correlation-related market signals. The network architecture is shown in Figure 6.

Specifically, at each time step t, we extract stock features from the price of each stock and the prices of those highly similar stocks in G. The detailed process is illustrated in Figure 7 which assumes a total of 8 stocks: initially, the prices of these 8 stocks are fed to our backbone stock correlation graph, where each node corresponds to a stock and has a feature vector of size 1 which contains the stock's price at time t. We remark that we can expand the node feature vector with other features such as technical indicators to integrate richer information; this paper uses price alone to examine how GCN improves the performance, since we do not want to introduce other factors that interfere the performance.

Then, the standard spectral graph convolution operation as specified by Eq. (1) is used to extract a feature vector of length-ℓ for each stock (i.e., graph node). In Figure 7, we have $\ell = 4$. In our real experiments, we tested $\ell = 64$, 128, and 256, and we found that $\ell = 128$ delivers the best performance and is thus adopted.

Refer back to the Figure 6 again: we can see that this GCN module is shared by all time steps for feature extraction. In other words, the weight parameter matrix $W^{(\ell)}$ in Eq. (1) is shared across all time steps. In contrast, each stock has its own GRU network to learn its unique sequence predictor, where the sequence of extracted feature vectors for that stock is fed into the sequence of GRU units in order.

In Figure 6, the features extracted by the last GRU unit are fed into a dense layer to obtain the predicted stock price in time $(t + 1)$, but during training this shared dense layer is actually connected to the output of every GRU unit to predict the stock price in the subsequent time step. The sequence of predicted prices are compared to the actual price sequence to obtain the mean squared error for back-propagation to adjust the model parameters. In other words, the GRU network of every stock is a many-to-many network. We omit these details on loss in Figure 6 to keep the diagram simple.

IV. Empirical Evaluation

We now report our experiments. We have described our datasets in Section III-A, and our model is open-sourced at https://github.com/troyyxk/gcgru_stock_prediction.

We consider both the regression problem of predicting the actual stock price in day $(t+1)$, and the classification problem of predicting the price direction (up or down) in day $(t + 1)$. They are one-step ahead predictions. The prediction uses the

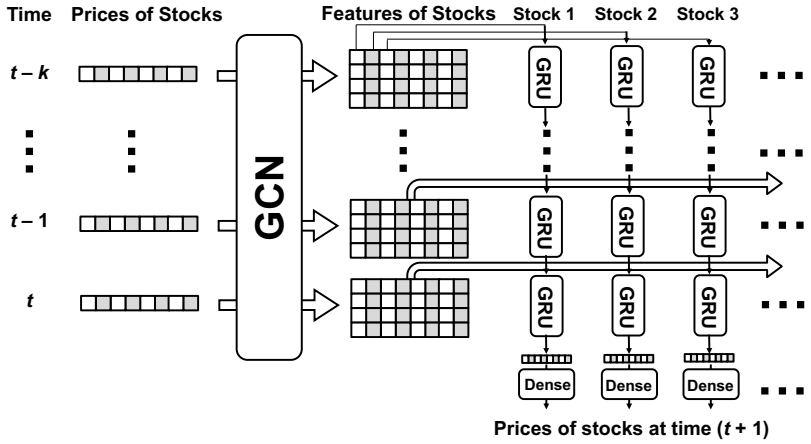

Fig. 6. Our Model Architecture

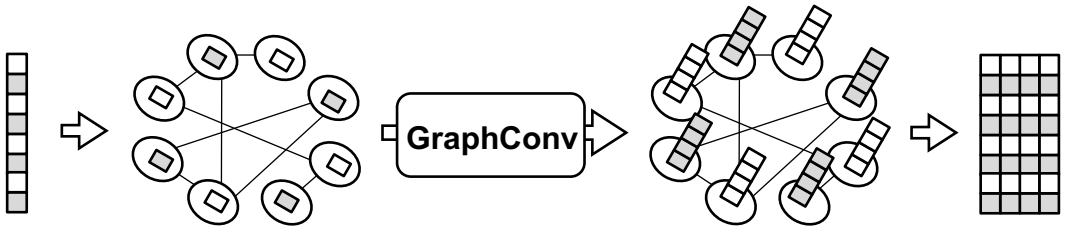

Fig. 7. Illustration of Graph Convolution

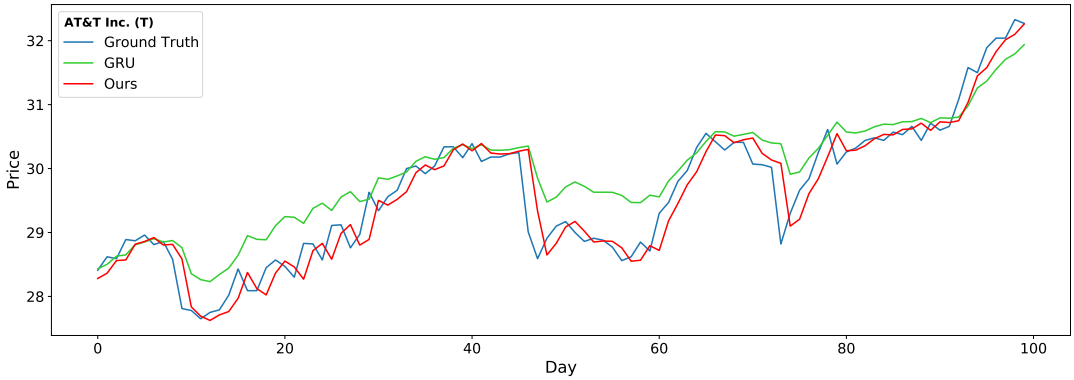

Fig. 8. Predicted Price Curve of AT&T

sequence of stock prices from the previous $k = 60$ days. We also tried $k = 30$ and the performance is similar, but further reducing k degrades the performance.

For the other model hyperparameters, we use 10^{-3} as the learning rate, Adam as the optimizer, 128 as the length of GCN output feature vector, 128 as the batch size. Our models for DOW and ETF are both trained for 100 epochs and they are able to reach loss convergence.

A. *Effectiveness of Applying GCN*

To demonstrate the effectiveness of applying a shared GCN to integrate the price information of highly-correlated stocks using a stock correlation graph, we compare our model (denoted by **Ours**) with a baseline **GRU** model trained for each stock using its price sequence.

As an illustration, Figures 8 and 9 show the predicted price curves of two stocks from **DOW** on the test time duration for the first 100 days, where we can see that the predicted curves of our model is much closer to the actual price curves than those predicted by GRU. We, however, observe that the predictions are slightly delayed in response compared with the actual curve, which is likely due to the lack of additional market signals (e.g., news) integrated into the model.

Due to the space limitation, we randomly sample 6 stocks from **DOW** and 4 ETFs from **ETF**, and show the model performance on the 10 investment products in Table I in order. For price direction classfication, we report the test accuracy and another ϵ-insensitive *test accuracy* (column "Accuracy*" in Table I) that does not consider a prediction to wrong if the predicted price $\hat{x}_{t+1}$ is within ϵ from the previous price x_t. We define this new accuracy metric since the day-price curve is relatively stable with much reduced volatility, so there are days where the median share-price almost does not change; in

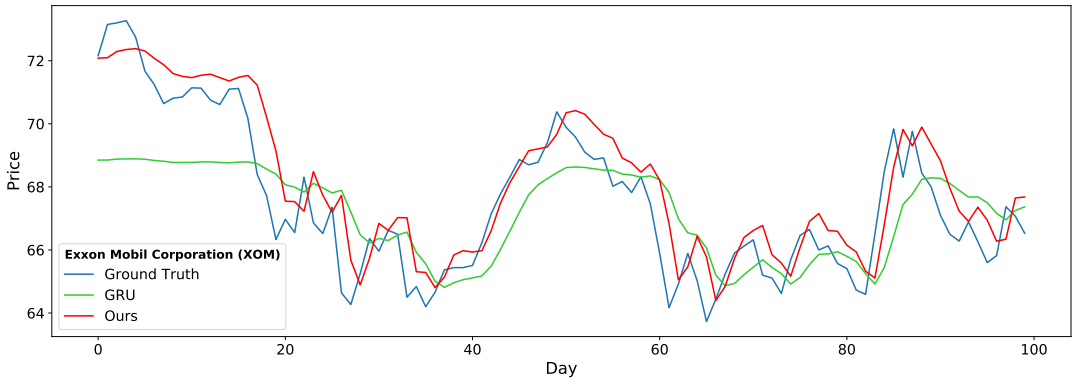

Fig. 9. Predicted Price Curve of Exxon Mobil Corporation

TABLE I
PERFORMANCE COMPARISON: OUR MODEL V.S. GRU

Stock	Model	Accuracy	Accuracy*	R2	RMSE	MAE	RE
AA	GRU	0.5400	0.6178	0.9885	0.8748	0.7403	0.0415
	Ours	0.5332	0.6545	0.9918	0.6228	0.5258	0.0282
AIG	GRU	0.5011	0.5606	0.9825	1.2375	0.8064	0.0219
	Ours	0.4897	0.5812	0.9875	1.0443	0.6980	0.0191
GE	GRU	0.4691	0.7140	0.9553	0.3751	0.2764	0.0318
	Ours	0.5057	0.7666	0.9630	0.3416	0.2484	0.0287
T	GRU	0.5423	0.6018	0.8290	1.5636	1.0971	0.0321
	Ours	0.5515	0.6888	0.8767	1.3278	0.8952	0.0259
WBA	GRU	0.4966	0.4989	0.9583	2.0606	1.3332	0.0228
	Ours	0.4966	0.5355	0.9756	1.5750	1.0629	0.0186
XOM	GRU	0.5217	0.5240	0.7459	5.5159	4.0886	0.0731
	Ours	0.5195	0.5584	0.8544	4.1763	2.3925	0.0496
EEM	GRU	0.5059	0.5216	-0.1335	2.8253	2.2704	0.0543
	Ours	0.5235	0.5431	0.1770	2.4073	1.8982	0.0454
EWZ	GRU	0.4706	0.5569	0.9790	0.9790	0.6599	0.0192
	Ours	0.4608	0.5765	0.9806	0.9102	0.6423	0.0187
XME	GRU	0.5431	0.5804	0.9848	0.5478	0.4156	0.0167
	Ours	0.5353	0.6824	0.9872	0.5031	0.3736	0.0149
XRT	GRU	0.5137	0.5471	0.8367	1.8077	1.2016	0.0277
	Ours	0.5157	0.6039	0.8885	1.4936	0.9630	0.0218

such cases, the predicted price direction is sensitive to small changes to the predicted price. For example, if $x_t = 60$ and x_{t+1} is almost still 60, then when we predict $\hat{x}_{t+1} = 59.9$ (resp. 60.1) the direction will be down (resp. up). Only one of the direction is correct when we evaluate the accuracy, but both are correct if we consider the ϵ-insensitive accuracy where we use $\epsilon = \$0.2$ by default. As shown in Table I where we highlight the better result of GRU and our model in red, we can see that our model is consistently higher than GRU in terms of ϵ-insensitive accuracy which demonstrates the effectiveness of using GCN, though GRU sometimes achieves a higher accuracy. In fact, the accuracy of both models are close to 50% due to the randomness of price direction prediction caused by small changes of the predicted price in cases where the prices almost remain the same; in contrast, ϵ-insensitive accuracy can be up to 76.66% and better reflects the actual model performance.

For regression, we report 4 metrics including (i) coefficient of determination (R^2) which measures the ability of the prediction result to represent the actual data: the larger the value is, the better the prediction is; (ii) root mean squared error (RMSE); (iii) mean absolute error (MAE); and (iv) (mean) relative error (RE). In Table I, we can see that our model consistently outperforms GRU in all four metrics.

B. *Effect of Binarization Threshold τ*

Recall that we use $\tau = 0.9$ in our experiments. Setting τ too low will cause GCN to over-smooth, while setting τ too high will prevent some highly correlated stocks from propagating

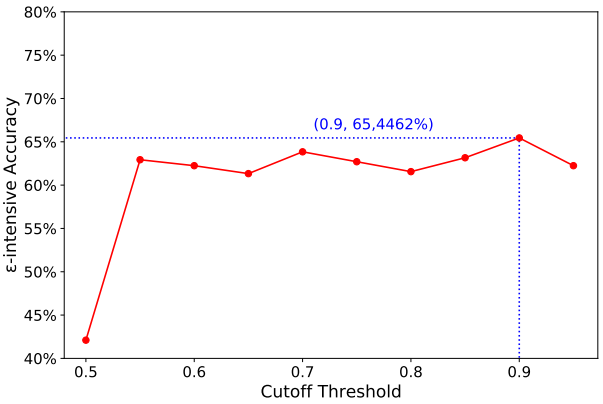

Fig. 10. Effect of τ on American Airline (AA)

their price information. We have conducted extensive experiments to find a good τ, and $\tau = 0.9$ delivers consistently the best accuracy for almost all stocks. Figure 10 provides such an illustration for AA.

V. Conclusion

This work proposed to extract features from the stock correlations using GCN and to integrate the correlation-related features into an RNN-based sequence prediction model. Extensive experiments on real stock data have been conducted to verify the effectiveness of the proposed approach. Our model follows the idea of multi-task learning, where each task learns its unique RNN-based sequence predictor, but all stocks share a common GCN module to improve the GCN training to more effectively propagate correlation-related market signals. Also, our design is general and can be applicable to predict other investment products. In fact, we already tested our model on ETFs. Last but not the least, our model can be easily extended to incorporate other features such as technical indicators and news sentiments (e.g., extending price vector on the left of Figure 7) to further improve performance.

Acknowledgment

This work is partially supported by NSF OAC-1755464 (CRII) and NSF DGE-1723250.

References

[1] W. Bao, J. Yue, and Y. Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944, 2017.

[2] S. Borovkova and I. Tsiamas. An ensemble of lstm neural networks for high-frequency stock market classification. *Journal of Forecasting*, 2019.

[3] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *ICLR*, 2014.

[4] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *AAAI*, pages 3438–3445, 2020.

[5] J. Chen, T. Ma, and C. Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *ICLR*. OpenReview.net, 2018.

[6] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *SSST@EMNLP*, pages 103–111, 2014.

[7] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pages 3837–3845, 2016.

[8] Y. Dong, D. Yan, A. Almudaifer, S. Yan, Z. Jiang, and Y. Zhou. Belt: A pipeline for stock price prediction using news. In *BigData*. IEEE, 2020.

[9] B. Ellickson, M. Sun, D. Whang, and S. Yan. Estimating a local heston model. *SSRN 3108822*, 2018.

[10] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IJCNN*, volume 2, pages 729–734. IEEE, 2005.

[11] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864. ACM, 2016.

[12] Y. Gu, D. Yan, S. Yan, and Z. Jiang. Price forecast with high-frequency finance data: An autoregressive recurrent neural network model with technical indicators. In *CIKM*, 2020.

[13] W. L. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017.

[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[15] T. Kim and H. Y. Kim. Forecasting stock prices with a feature fusion lstm-cnn model using different representations of the same data. *PloS one*, 14(2):e0212320, 2019.

[16] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[17] Q. Li, Z. Han, and X. Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In S. A. McIlraith and K. Q. Weinberger, editors, *AAAI*, pages 3538–3545, 2018.

[18] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel. Gated graph sequence neural networks. In *ICLR*, 2016.

[19] Y. Li, R. Yu, C. Shahabi, and Y. Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *ICLR*. OpenReview.net, 2018.

[20] Z. W. Ming Chen, B. D. Zengfeng Huang, and Y. Li. Simple and deep graph convolutional networks. 2020.

[21] D. M. Q. Nelson, A. C. M. Pereira, and R. A. de Oliveira. Stock market's price movement prediction with LSTM neural networks. In *IJCNN*, pages 1419–1426, 2017.

[22] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: online learning of social representations. In *SIGKDD*, pages 701–710. ACM, 2014.

[23] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Trans. Neural Networks*, 20(1):61–80, 2009.

[24] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *ICLR*, 2018.

[25] F. Wu, A. H. S. Jr., T. Zhang, C. Fifty, T. Yu, and K. Q. Weinberger. Simplifying graph convolutional networks. In *ICML*, volume 97, pages 6861–6871. PMLR, 2019.

[26] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[27] C. Yang, R. Wang, S. Yao, S. Liu, and T. F. Abdelzaher. Revisiting "over-smoothing" in deep gcns. *CoRR*, abs/2003.13663, 2020.

[28] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *SIGKDD*, pages 974–983. ACM, 2018.

[29] B. Yu, H. Yin, and Z. Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *IJCAI*, pages 3634–3640, 2018.

[30] L. Zhang, P. A. Mykland, and Y. 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.

[31] Z. Zhang, S. Zohren, and S. J. Roberts. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Trans. Signal Processing*, 67(11):3001–3012, 2019.

[32] L. Zhao and L. Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *ICLR*, 2020.

[33] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li. T-GCN: A temporal graph convolutional network for traffic prediction. *IEEE Trans. Intell. Transp. Syst.*, 21(9):3848–3858, 2020.