Predicting Future Asset Returns with GCN and LSTM

Wesley Yuan
Department of Statistics
Columbia University
New York, NY, 10027
wy237@columbia.edu

Gurmeha Makker
Department of Statistics
Columbia University
New York, NY, 10027
gm2946@columbia.edu

Sierra Vo
Department of Statistics
Columbia University
New York, NY, 10027
tdv2104@columbia.edu

Aiden Kenny
Department of Statistics
Columbia University
New York, NY, 10027
apk2152@columbia.edu

Abstract

Placeholder

1 Introduction

The problem of predicting future returns given historical data for tradable assets has been extensively studied with many approaches having been explored. Traditional methods used time-series models such as ARIMA and GARCH to predict future price movements. Similarly, deep-learning models that can take advantage of temporal relations such as Long Short-Term Memory (LSTM) models have been applied to this problem with promising results. However, these methods fail to take into account the propagation of information through the market and the correlations of assets. In this aspect, Graph Convolutional Networks (GCN) has demonstrated good performance in regression problems. Combining these should allow for the capture and use of both intra-asset temporal and cross-asset relations to provide superior prediction performance.

1.1 Related Works

1.2 Dataset

The dataset used for this project is the G-Research Crypto Dataset which contains minute-level price data for 14 commonly traded crypto assets. For each minute, the open, high, low, and close prices for the past minute are provided as well as the total volume, number of trades, and the volume weighted average price (VWAP). The target variable is the 15 minute return of the given asset. Due to computational constraints, we utilize the latest 100k timestamps available for our study.

In total, about 3% of the Target values and a near zero number of VWAP values are missing. These missing values are largely the result of insufficient data on assets earlier in the time window from which the data was collected while the last 15 values are all missing (most likely to prevent information leaking). These missing values do not affect our study due to us not utilizing earlier data where the majority of missing values are. The last 15 are forward-filled.

We build other

Preprint. Under review.

1.3 Background

1.3.1 LSTM

LSTM models are a special kind of Recurrent Neural Network (RNN) model which feature evolving hidden states that capture time-dependencies in sequential inputs. As such, these models have been widely popular in processing sequential data such as speech, text, and video. They solve the main drawback of other RNN models of not being able to caputre long-term dependencies by introducing a "memroy gate" and a "forget gate" to better persist relevant information and discard irrelevant information, respectively.

A standard LSTM model will have the following components

1. Some input at every time step $x_t \in \mathbb{R}^{D_f}$ where D_f is the embedding dimension of the features
2. A memory state $c_t \in \mathbb{R}^{D_h}$ and a hidden state $h_t \in \mathbb{R}^{D_h}$ where D_h is the number of units in the hidden dimension (typically user-defined)
3. Input cell $i_t \in \mathbb{R}^{D_h}$ that controls what relevant information from previous states and new input is passed forward
4. Some intermediate states $z_t, c_t \in \mathbb{R}^{D_h}$
5. Output cell $o_t \in \mathbb{R}^{D_h}$ that controls what relevant information from intermediate steps makes it to the next time step
6. Forget cell $f_t \in \mathbb{R}^{D_h}$ that controls what information is discarded going forward

The typical operations in a single LSTM step are as follows

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + Q_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + Q_i h_{t-1} + b_i) \\
z_t &= \tanh(W_c x_t + Q_c h_{t-1} + b_c) \\
c_t &= f_t * C_{t-1} + i_t * z_t \\
o_t &= \sigma(W_o x_t + Q_o h_{t-1} + b_o) \\
h_t &= o_t * \tanh(c_t)
\end{aligned}
$$

where $W_* \in \mathbb{R}^{D_h \times D_f}$, $Q_* \in \mathbb{R}^{D_h \times D_h}$, and $b_* \in R^{D_h}$ are learnable parameters and σ denotes some user-specified activation function.

1.3.2 Graph Neural Networks

Graph neural networks (GNN) is the application of neural networks to learning graph problems such as link prediction, node classification, etc. These models have achieved state-of-the-art performance on graph problems as well as regression/classification on data with natural graph structure (such as review data, word nets, social networks, etc.). GNN are typically classified according to the scope of their learning, either node-level or graph-level where graph-level GNN add more sophisticated pooling to node-level GNN to make global predictions.

GNN's main improvement over standard deep learning methods is the focus on learning embeddings of nodes, edges, or subgraphs that preserve structure such as permutation invariances. This can be done by having separate models for each component of a graph (nodes, edges, global) that are able to better capture information than using a single model on a simpler representations (such as adjacency matrices/lists). A convenient viewpoint is that of the encoder-decoder model taken in Hamilton et al. [2017] where any given task has an encoder that learns the embeddings of the graph components and a decoder that learns the mapping from the embeddings to the targets. Thus, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $v_i, v_j \in \mathcal{V}$, and $s_\mathcal{G}$ being some structure of the graph, the model is given by

$$s_\mathcal{G}(v_i, v_j) \approx \text{Decode}(\text{Encode}(v_i), \text{Encode}(v_j)) \tag{1}$$

For example, setting $s_{\mathcal{G}}(v_i, v_j) = A_{i,j}$ would give the problem of predicting the connections in the graph.

Graph connectivity is utilized in pooling the outputs of these separate models between layers. For node prediction tasks this would entail pooling edge information at the connected node whereas for edge prediction tasks the information is pooled from nodes that the edge connects. This pooling between neighboring components also performs an implicit message passing whereby component embeddings affect the updates of those they are connected to. Let $\boldsymbol{x}_i^l$ be the encoding after the l-th message passing layer, then the update to the embedding is given by

$$\boldsymbol{x}_i^{l+1} = \boldsymbol{W}^l x_i^l + \text{Aggregate}(\boldsymbol{x}_j^l),\ j \in Ne(v_i) \tag{2}$$

where Aggregate is some aggregation function, $\boldsymbol{W}^l$ is a learned weights matrix, and $Ne(v_i)$ denotes the neighbors of node v_i.

The level of message passing is largely determined by the number of GNN layers, with more layers allowing for greater aggregation) and the aggregation function used in pooling the embeddings (with more complicated aggregation schemes allowing for greater information flow).

Graph convolutional networks (GCN) add an additional layer of complexity as it takes in a node feature vector and an adjacency matrix at every step to provide improved context for the message-passing steps. Given the adjacency matrix $A \in \mathbb{R}^{n \times n}$, we add self-connections to get $\tilde{A} = A + I_n$ where I_n is the $n \times n$ identity matrix. We can then construct the diagonal degree matrix $\tilde{D}$ where $\tilde{D}_{ii} = \sum_i \tilde{A}$. Then the propagation rule introduced in Kipf and Welling [2017] is

$$f(H^{(l)}, A) = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} H^{(l)} W^{(l)}) \tag{3}$$

where $H^{(l)}$ is the activations of layer l and $H^{(0)} = X$ i.e. the node feature vector and $W^{(l)}$ is a learnable weights matrix.

2 Methods

Traditional asset return predictions utilized time-series models that take into account the temporal nature of trading as sequential decision making. As such, most deep-learning applications in this area have leveraged the temporal aspect of RNN models, particularly LSTM, in making predictions (Shen and Shafiq [2020], Li et al. [2018], Selvin et al. [2017]). Therefore, we use a vanilla LSTM model as the baseline model against which we compare our combined methods.

To improve upon the baseline, we introduce two additional techniques to better capture information lost in training an LSTM model. The first technique we introduce is the graph neural network (GNN) that we use to learn relational information from an implicit graph structure derived from the assets. The connectivity of the assets is evident in the log-returns correlation of the assets in our dataset as seen in figure 1. This phenomenon can largely attributed to the fact that agents acting in markets tend to trade many assets at once to get better risk-adjusted returns through diversification. The interactions of these market agents with the assets determines the structure of our market graph representation. This approach is also used in Matsunaga et al. [2019], Feng et al. [2019], Sun et al. [2020], Hou et al. [2021], and Peng [2021].

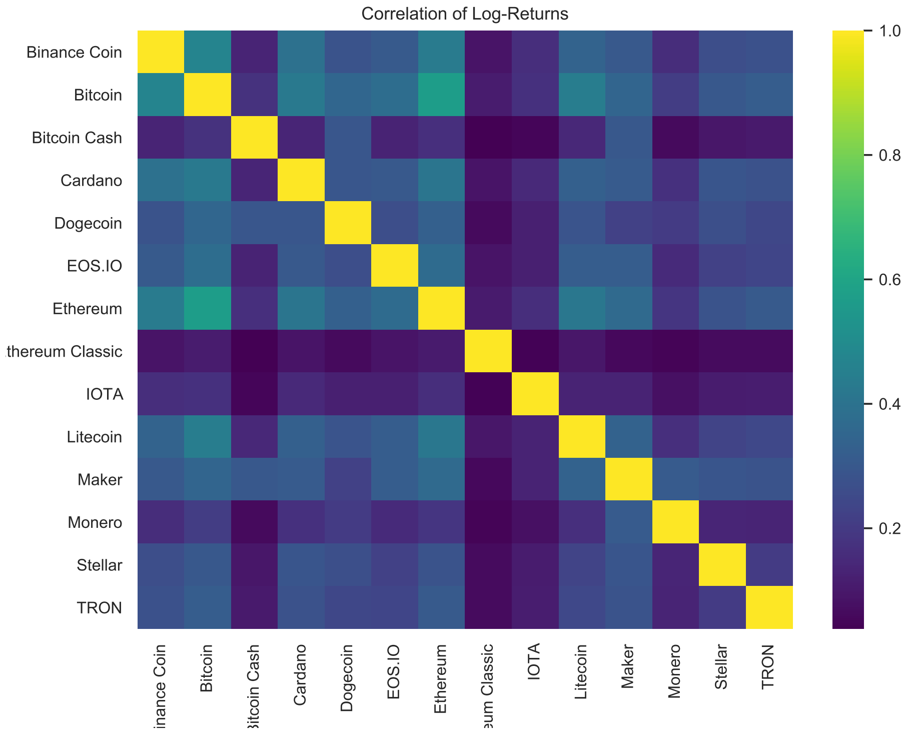

Figure 1: Correlation of log returns calculated on minute close prices of assets. Note the high correlation among the most commonly traded assets like Bitcoin/Ethereum and Bitcoin/Binance Coin.

Peng [2021] show empirically that using the correlation matrix as an adjacency matrix as input into GCN layers provided best performance. Therefore, we use the same, imposing a lookback window equal to the length of the sequence input to the LSTM model. Restricting this lookback window provides a basis for direct comparison as the GCN-augmented model will not have access to more information than the LSTM model and any performance boost would be a result of capturing relational information from the correlation matrix. For our specific GCN model, we implement the propogation rule taken from Kipf and Welling [2017].

3 Results

4 Discussion

References

F. Feng, X. He, X. Wang, C. Luo, Y. Liu, and T.-S. Chua. Temporal relational ranking for stock prediction. 37(2), 2019. doi: 10.1145/3309547.

W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584, 2017. URL `http://arxiv.org/abs/1709.05584`.

X. Hou, K. Wang, C. Zhong, and Z. Wei. St-trader: A spatial-temporal deep neural network for modeling stock market movement. *IEEE/CAA Journal of Automatica Sinica*, 8(5):1015–1024, 2021.

T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

H. Li, Y. Shen, and Y. Zhu. Stock price prediction using attention-based multi-input lstm. In J. Zhu and I. Takeuchi, editors, *Proceedings of The 10th Asian Conference on Machine Learning*, volume 95 of *Proceedings of Machine Learning Research*, pages 454–469. PMLR, 14–16 Nov 2018.

D. Matsunaga, T. Suzumura, and T. Takahashi. Exploring graph neural networks for stock market predictions with rolling window analysis. *CoRR*, abs/1909.10660, 2019.

S. Peng. Stock forecasting using neural network with graphs. Master's thesis, University of York, York, England, May 2021.

S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V. K. Menon, and K. P. Soman. Stock price prediction using lstm, rnn and cnn-sliding window model. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1643–1647, 2017. doi: 10.1109/ICACCI.2017.8126078.

J. Shen and M. O. Shafiq. Short-term stock market price trend prediction using a comprehensive deep learning system. *Journal of Big Data*, 7(66), 2020.

J. Sun, J. Lin, and Y. Zhou. Multi-channel temporal graph convolutional network for stock return prediction. In *2020 IEEE 18th International Conference on Industrial Informatics (INDIN)*, volume 1, pages 423–428, 2020. doi: 10.1109/INDIN45582.2020.9442196.

A Appendix