Predicting Future Asset Returns with GCN and LSTM

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

Gurmehar 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 Background

1.1.1 Long Short-Term Memory (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 $\mathbf{x}_t \in \mathbb{R}^{D_f}$ where D_f is the embedding dimension of the features
2. A memory state $\mathbf{c}_t \in \mathbb{R}^{D_h}$ and a hidden state $\mathbf{h}_t \in \mathbb{R}^{D_h}$ where D_h is the number of units in the hidden dimension (typically user-defined)

Preprint. Under review.

3. Input cell $\mathbf{i}_t \in \mathbb{R}^{D_h}$ that controls what relevant information from previous states and new input is passed forward
4. Some intermediate states $\mathbf{z}_t, \mathbf{c}_t \in \mathbb{R}^{D_h}$
5. Output cell $\mathbf{o}_t \in \mathbb{R}^{D_h}$ that controls what relevant information from intermediate steps makes it to the next time step
6. Forget cell $\mathbf{f}_t \in \mathbb{R}^{D_h}$ that controls what information is discarded going forward

The typical operations in a single LSTM step are as follows

$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{Q}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{Q}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{z}_t &= \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{Q}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{c}_t &= \mathbf{f}_t * \mathbf{c}_{t-1} + \mathbf{i}_t * \mathbf{z}_t \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{Q}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t * \tanh(\mathbf{c}_t)
\end{aligned}
$$

where $\mathbf{W}_* \in \mathbb{R}^{D_h \times D_f}$, $\mathbf{Q}_* \in \mathbb{R}^{D_h \times D_h}$, and $\mathbf{b}_* \in R^{D_h}$ are learnable parameters and σ denotes some user-specified activation function.

1.1.2 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 **?** 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 $s_\mathcal{G}(v_i, v_j) = \mathbf{A}_{i,j}$ 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 $\mathbf{x}_i^l$ be the encoding after the l-th message passing layer, then the update to the embedding is given by

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

where Aggregate is some aggregation function, $\mathbf{W}^l$ 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 $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_n$ where I_n is the $n \times n$ identity matrix. We can then construct the diagonal degree matrix $\tilde{\mathbf{D}}$ where $\tilde{\mathbf{D}}_{ii} = \sum_i \tilde{\mathbf{A}}$. Then the propagation rule introduced in **?** is

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

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

1.2 Related Works

The application of deep learning in asset price predictions is a popular area of ongoing research with most works utilizing variations of RNN models which can learn temporal relationships.

The authors of **?** develop a custom deep learning pipeline from feature extraction to final prediction. They apply typical transformations such as scaling and calculating technical indicators on price data before combining with company financial data as features that are filtered through recursive feature elimination and PCA before being fed into an LSTM final prediction model. Their work focuses on select single-name stocks from the Chinese stock market and achieves 87% accuracy on directional predictions on their dataset.

Other works focus on specific parts of the prediction pipeline rather than developing it end-to-end. **?** expand on the vanilla LSTM model to allow for multiple inputs at each step for prediction. The proposed multi-input LSTM model combines low-correlated factors through additional "input gates" (for more detailed description of LSTM model see section **??**). The authors test their architecture on stocks in the CSI-300 index and achieve a 10% improvement on MSE compared to other state-of-the-art LSTM architectures. Their work shows that additional contextual information can be used to improve performance of stock prediction models.

? looked to compare the LSTM model with other temporal models such as other RNN architectures as well as sliding window Convolutional Neural Network (CNN) models. The authors tested each of the three models on select single-name stocks (Infosys, TCS, Cipla) and measured their error percentage. They show that deep learning models are able to effectively capture stock price dynamics and that temporal information is highly useful in making future predictions.

The combination of LSTM models and graphical models has been a growing topic of interest as more researchers look to combine the performance of LSTM models in sequential data prediction tasks and the ability of graph models to capture cross-asset dependenceis.

? explored the use of temporally-enabled graph models on the stock price prediction problem. In their work, temporal information is added using an LSTM model on raw inputs as an embedding layer that produces inputs for the GCN model. This combination has been named Temporal Graph Convolution (TGC) and aims to produce both sequential and relational embeddings. This embedding and an adjacency matrix calculated using contextual company relationship data are fed into a GCN that outputs some future price prediction. They test their model on the Nikkei 225 and show that it outperforms both the market and, more importantly, the baseline LSTM model by 29.5% and 6.3%, respectively.

Similarly, **?**, makes use of TGC on the related stock ranking problem. That is, rather than predict the raw returns/prices of stocks at some future time horizon, the authors use the TGC model to predict the relative future performance of a set of assets. They test the model on NYSE and NASDAQ stock data and shows it outperforms vanilla GCN and other state-of-the-art ranking algorithms. Their study shows that the superior performance of TGC comes from combining the relational and temporal information and not from either single embedding scheme. Further, their study suggests that TGC's use is not limited to any specific market or regression problem but is trainable on any given set of assets.

? and **?** further expand on the literature of TGC by testing different methods of constructing the adjacency matrix input. **?** used a Variational AutoEncoder (VAE) model to cluster stocks into a graph structure and produce some adjacency matrix based on company fundamentals. They tested their

adjacency matrix construction method using minute-level data from S&P100. The authors show superior performance compared to adjacency matrices constructed using raw company fundamentals and using company industry encodings. **?** also uses industry encodings as an adjacency matrix option but compares it to price-data derived alternatives such as correlation matrix and Dynamic Time Warping (DTW) induced distances. **?** demonstrated highest performance when using correlation matrix based on 10-day lookback window as the adjacency matrix.

1.3 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.

1.4 Technical Indicators

To create the feature vectors of the GCN for training, we make use of several popular technical indicators. The purpose of these technical indicators is to provide a better baseline of previous information in order to predict the current price with a smaller model.

The authors in **?** study the performance of several technical indicators to predict the daily closing price of Bitcoin. Of the five used in their paper, we choose to adopt three to use ourselves: the *Simple Moving Average* (SMA), *Exponential Moving Average* (EMA), and the *Relative Strength Index* (RSI). In addition, we have two other indicators, *Bollinger Bands* and a *Stochastic Oscillator*, which has been shown by **?** to be effective in predicting cryptocurrency prices.

We use several different sized windows (5, 20 ,50) for moving average indicators and keep RSI, Bollinger Bands and Stochastic Oscillator windows constant at 14, 20, and 14 time step look backs, respectively. This added an additional 9 features per asset that we use as input to our deep learning models.

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 (**?**, **?**, **?**). 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 a 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 **??**. 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 **?**, **?**, **?**, **?**, and **?**.

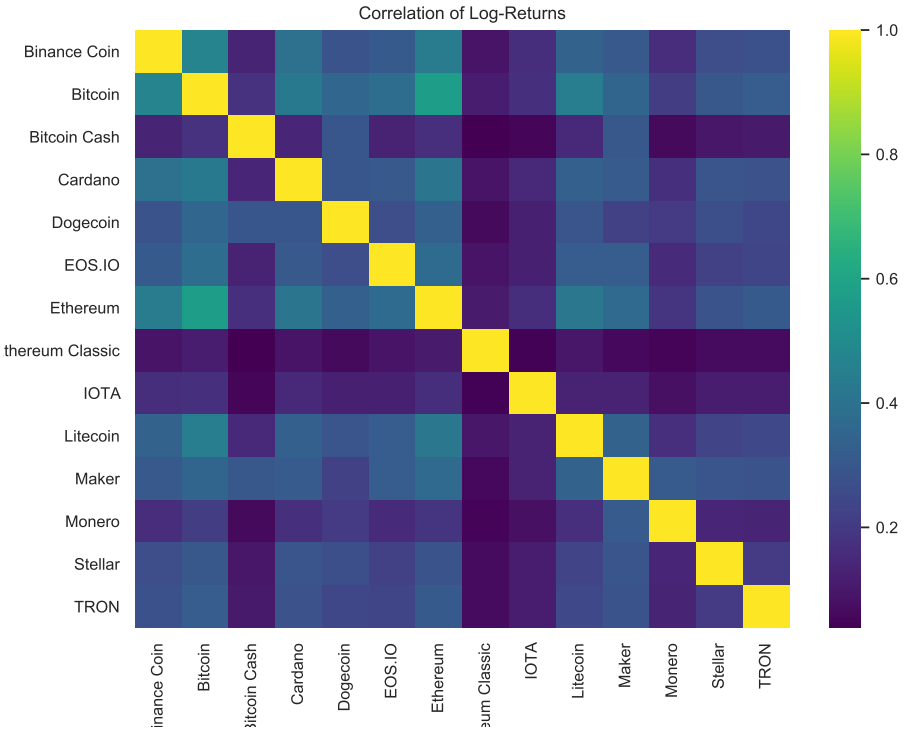

Figure 1: Correlation of log returns calculated on minute close prices of assets. Note the high correlation among the most commonly traded assets like Bitcoin/Ethereum and Bitcoin/Binance Coin.

? 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 **?** with slight adjustment. For our model, we omit $\tilde{\mathbf{D}}^{-1/2}$ from the update rule because it is poorly defined given $\tilde{A}$ as the correlation matrix has negative degree nodes. Therefore, we have the following

$$f(\mathbf{H}^{(l)}, \mathbf{A}) = \sigma(\tilde{\mathbf{A}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}) \tag{4}$$

where variables are as defined above.

3 Experiments

We use the crypto data described above to train and test the performance of the different models. Due to computational limitations, we take the last 100k time steps with 90k time steps for training and the last 10k used as a test set. For LSTM and combined models, a sequence of 10 time-steps is taken together as inputs to the model. For GCN, a single frame and the 10-day lookback correlation matrix is used as inputs. Training hyperparameters were tuned manually to find a reasonable best-performing combination.

Our baseline model is a single-layer vanilla LSTM model with a hidden size of 14. Hidden and memory states are initialized to zero tensors. The embeddings at each of the 10 steps is flattened and fed through a final fully connected layer for predictions. Combined models use the baseline model

and a single-layer GCN model that uses the above update and creates a 3-dimensional embedding that is then fed through a fully-connected layer to get the model's prediction.

3.1 Results

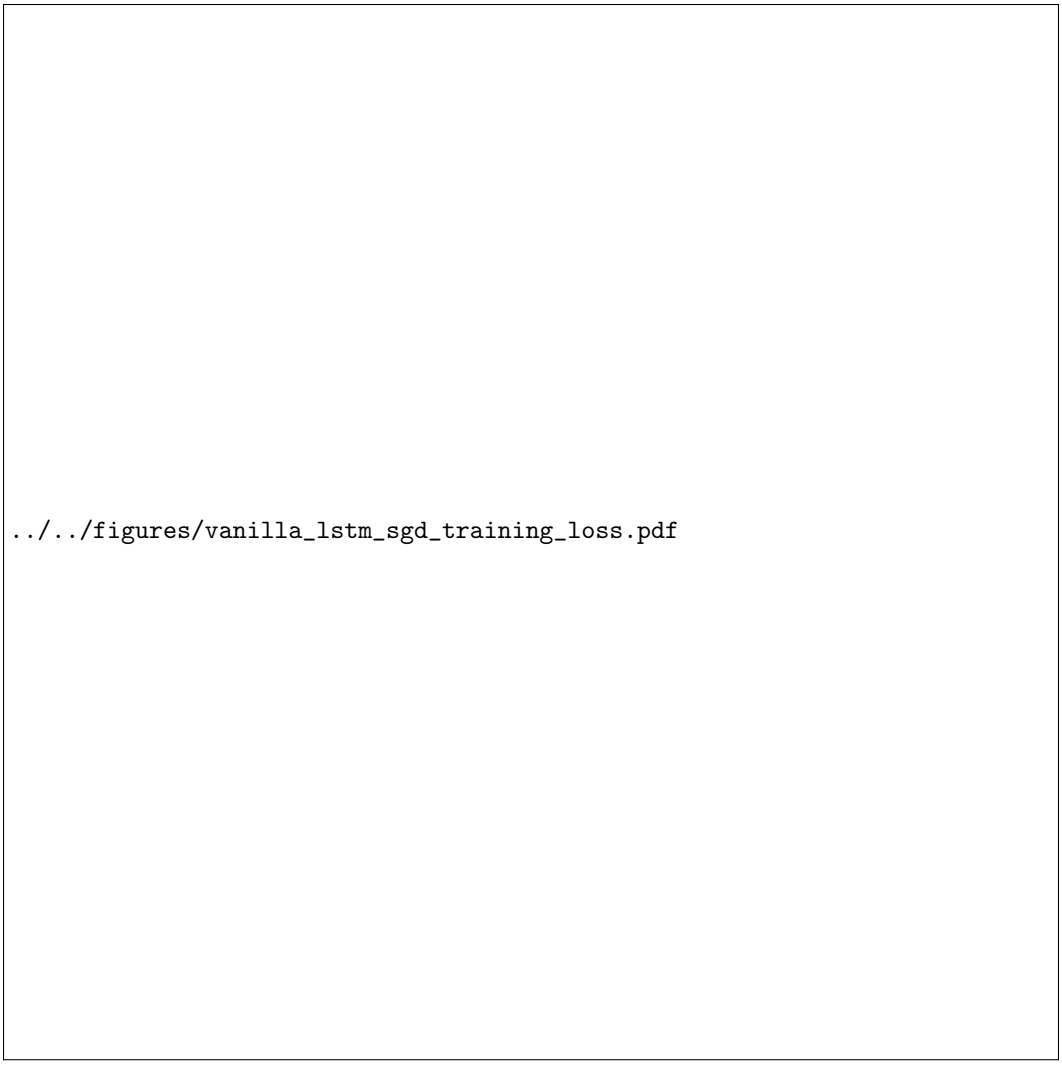

Figure 2: Training loss of baseline LSTM model

Model	Optimzier	Learning Rate	Momentum	MSE
LSTM	SGD	0.0001	0.9	0.0002
Additive				
Sequential				

Table 1: Hyperparameters for best performing model with average MSE on validation set

4 Discussion

Pending results

A Appendix