Graph generation for synthetic stock data

Bachelor assignment paper

Matthijs Reus

Abstract— **There will always be a finite amount of time over which stocks have been recorded. Therefore, synthetic stock data create fabricated timelines where investment strategies can be tested and enhanced. I investigate how graph generation can assist in generating synthetic stock data.**

Graphs are used to understand how objects like computers, people, molecules or companies are related. In this work, I use graphs to connect companies with similar stock data in a company correlation graph. The graphs will be encoded as a sequence, to allow a recurrent neural network to be trained and generate new graphs. In turn, such graphs can be used to make predictions about the future stock values.

My results show that this model can generate company correlation graphs similar to the original data set. This approach is the first to use graph generation for synthetic stock data.

I. Introduction

ACCURATE financial models and informed decision making are crucial for investors of all sizes, big and small, alike. Recent advancements in machine learning and machine learning accessibility, paired with the representational power of graphs, present new opportunities for developing innovative approaches to modeling stock market behaviour. In this paper, I explore the use of graphs, graph generation, and recurrent neural networks (RNNs) to generate hypothetical stock data that can assist in creating and testing investment strategies.

The overall aim of the project is to investigate if using (partial) graph representation of stock data and graph generation can improve the generation quality of the synthetic stock data. To this end, I pursue the following research question:

How can we use stock data to generate inter-company graphs?

To this end, I propose methods to retrieve the graph dataset from the stock data, and to generate similar yet different graphs, using RNNs, to assist in generating hypothetical stock rates. The pioneering aspect of this work lies in its use of AI-generated company-correlation graphs to improve the generation of synthetic stock data.

The underlying idea for using *company correlation graphs* is that stocks do not exist on their own, but in a market. These graphs capture in a comprehensive way the similarity and dependency between companies. By transforming time series data (which is how stock data is currently collected) into correlation graphs, I can incorporate these additional factors that influence stock fluctuation.

To then generate new correlation graphs, I employ the GraphRNN model [1], [2], which utilizes RNNs in an encoder-decoder architecture. The encoder converts the correlation graph into a latent space representation, which enables the decoder to generate similar yet distinct graphs. The training of the GraphRNN model involves learning the node and edge distributions from a dataset of correlation graphs, which I created from time series stock data.

The remainder of this paper is organized as follows. I build a foundation by introducing relevant background and related work in section II. Then, I provide a high-level description of the proposed system in section III, which contains subsections describing each part in detail. I empirically evaluate the proposed approach, and present the method and results of the evaluation in Section IV. I conclude in section VI, summarizing my finding and proposing suggestions for future developments.

II. Background and Related Work

A. Data

The data used to create graphs is collected from Yahoo Finance [3]. This source provides, for a large number of companies, time-series with the following information: open value, highest value, lowest value, close value, adjusted close value, and volume. The time series have a daily interval, and are taken over a year. For the purpose of this research, we collected data from 24-5-2022 until 22-5-2023. This time interval is suggested by the beneficiary of this research, Peracton Ltd. which provided example data for a single company [4]. Then to increase the data I extend with the top 40 companies from S&P500, ranked by weight, according to [5].

The data is collected as an archive of `csv` files, with a companies ticker symbol as filename. Each file corresponds to a single company, and the data in the files is organized as seen in table I.

B. From time series to graphs

Since the general task of this project is to use graph generation to generate better hypothetical stock data, the raw data needs to be transformed into graphs. No single model or procedure exists for this transformation. For example, L. Lacasa et al. models the time series as spikes, and two datapoints are connected if their created edge does not intersect another spike [6]. A survey of such methods is done by V. Silva et al. [7], including *correlation graphs*. As my requirements are a multivariable time series input and undirected unweighted graphs output, the survery indicates the correlation graph is a good fit.

C. Correlation graph

A *correlation graph* can imply relations between multiple time series. It is obtained by calculating the correlation matrix of a set of time series. X. Yin et al. describe how the correlation matrix can be converted to a graph by thresholding with a value τ [8]. The thresholded correlation matrix can then be interpreted as an adjacency matrix of a graph. This graph is the correlation graph.

D. Graph generators

There are many types of graph generators. To understand these options, I have conducted a preliminary survey, where I included the main survey on the topic [9], [10]. Although algorithmic approaches, like RMAT [11] or MUSKETEER [12] do exist, recent research has left those approaches behind to capture more complex relations. My survey indicated that modern techniques rely almost exclusively on machine learning in order to learn the node and edge distributions. For this work I selected, based on its for flexibility in graph sizes, an auto-regressive approach for generating correlation graphs .

TABLE I
THE STOCK DATA FORMAT (AS PRESENT IN THE .CSV TIME-SERIES FILES).

Filename: <company ticker symbol>.csv						
Date	Open	High	Low	Close	Adj Close	Volume
...	...	...	...	...	...	...

E. Recurrent Neural Networks (RNN)

The main advantage of RNNs is that information from past computations can be used for future computations, in order to more accurately predict the next step. This way, more contextual and complex decisions can be made to create more accurate graphs. To be able to use past information, it is important that the network keeps the same weights and biases. This prevents the model size from exploding. The RNN model in fig. 1 shows how previous results are taken from memory to influence current prediction.

LSTM and GRU, two more advanced implementations of RNN, deal with the vanishing/exploding gradient problem the original RNN has. During back propagation, the gradient gets multiplied each time with weight w_2 (see fig.1). A factor bigger than one lets the gradient explode; a factor smaller than one vanishes the gradient. If the weight w_2 is set equal to one, it is no longer a weight but a wire. Expanding the model with a remember and forget gate greatly reduces the vanishing gradient problem, and creates what is called a GRU model (see fig. 2).

F. GraphRNN

By writing the graph as a *breadth first search* sequence of nodes, tools like RNN can be used for graph generation. J. You et al. use two RNN models to create new graphs [2]. The two models operate in a encoder-decoder relation. The first encodes the BFS sequence into a smaller latent space. This encoding ensures there will be similar, yet different graphs made by the decoder. Moreover, they introduces *maximum mean discrepancy* as a method to evaluate graph likeness.

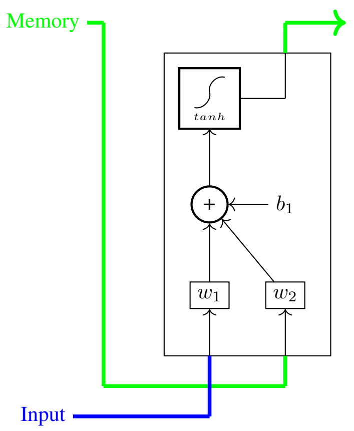

Fig. 1. RNN model architecture

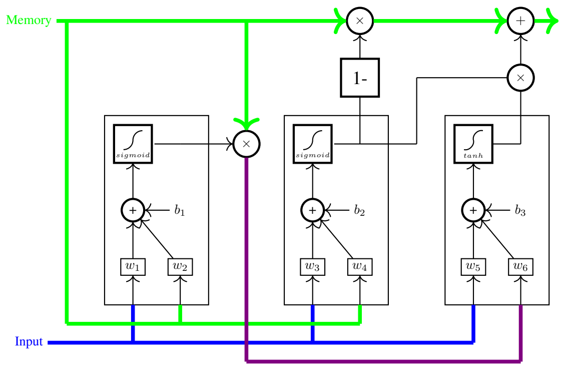

Fig. 2. GRU model architecture

G. NN models to FPGA

E. Nurvitadhi et al. have shown that FPGAs can be efficient for running GRU models [13]. Thus, we consider the additional research question of **how the created model can be run on an FPGA**.

To do so, we consider High level synthesis (HLS), a process that converts the high level code – e.g. Python, C++, C – to a hardware description language e.g. VHDL, Vitis, or Verilog. For machine learning applications, there are a couple libraries/tools to assist in this matter: hls4ml, Intel oneAPI AI Analytics Toolkit, and Xilinx Vitis AI are the most popular ones. For this project, we investigate whether hls4ml can provide HLS for the provided RNN model. hls4ml is the smaller and least complex open-source tool for using HLS in machine learning, so we chose it for its lack of complexity.

III. Design & Implementation

The main idea of this paper is to use stock correlation graphs and generate new ones to assist in creating more realistic ”alternative” time series. In this section I present the proposed method - see fig.3 - and discuss each component which is in the scope of this paper. Specifically, I describe the conversion from time series to correlation graph, training the RNN model, and creating the graphs. The conversion back to time series and a filter for selecting realistic datasets (based on economic and stock market parameters) is considered outside of the scope of this work.

A. Correlation graphs

Using jupyter lab[1], the .csv are loaded into a panda dataframe. These dataframes are then ordered in new dataframes, each containing the information of one data type - e.g. open value or high value - and one company per column. The correlation between each of these columns is calculated; the result is a matrix containing all the correlations between companies for a certain stock indicator. Since the correlation between companies A to B is the same as companies B to A, the correlation matrix is symmetrical across the main diagonal. Thus, the resulting correlation graph is undirected.

The correlation matrix is thresholded with a value τ to create a matrix with binary values, which can be translated to graphs in the `networkx` environment. X. Yin et al. found in heir results the best value for $\tau = 0.9$ [8]. However, for my dataset, initial results indicated this value was too large. Thus, the edge count is doubled, using fig. 6. I trained another model with correlation graphs with $\tau = 0.8$, which will be further explained in section IV.

The graphs of all different types are collected into a single list to form the training dataset. These different data types can all be put into the same list because they all relate and influence each other significantly. This is also beneficial for the size of the training set. Thus, the training dataset consists of 6 types of stock data over 13 months creating a total of 78 graphs.

[1]https://jupyter.org/try-jupyter/lab/

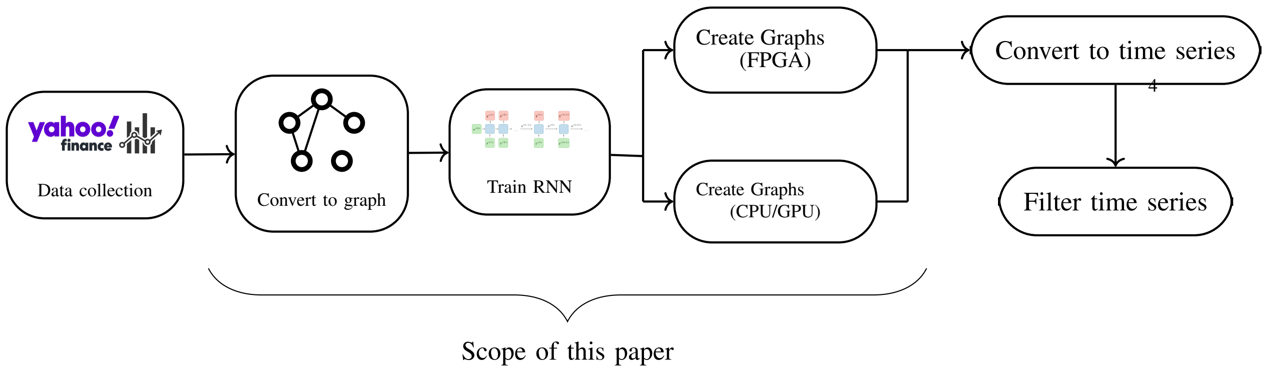

Fig. 3. Overall architecture of my approach to improve the generation of financial data through inter-company graph generation.

B. Training the RNN

The hardware used to train the model is a node on the DAS-5 supercomputer, which contains an Intel Xeon CPU and an NVIDIA TitanX video card [14]. The model parameters are largely kept the same as those used by J. You et al. for their GraphRNN paper, with a few minor alterations. It is known that the data set always has 40 graphs so the output of the model is always a 39 by 39 matrix. Furthermore, the amount of epochs is increased from 3000 to 9000, as the loss was still decreasing at 3000. The training and testing data is split to 80% into the training set. The loss function for backward propagation is kept to be binary cross-entropy, which is specialized for classification with two classes. In my case, the classes are 1 and 0, representing connected and unconnected nodes. Before training on my data sets, I validated the GraphRNN correctness by confirming the results using the `community4` data set [2].

C. Creating graphs

The output of the RNN model is a matrix of $(n-1) \times m$, where n is the maximum amount of nodes and m is the maximum previous node the generated node can connect to. The encoding stores whether the n-th node connects with the m-th previous node. The width of this matrix thereby also determines the maximum amount of edges a single node can have. This encoded adjacency matrix is then decoded into a full adjacency matrix as seen in fig. 4.

Please note that the encoding can create a graph which does not have to be connected. An example of this case is presented in fig. 4; its corresponding graph is presented in fig. 5. This process is described in more detail in the appendix VII.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Fig. 4. Example of raw GraphRNN output and its converted adjacency matrix.

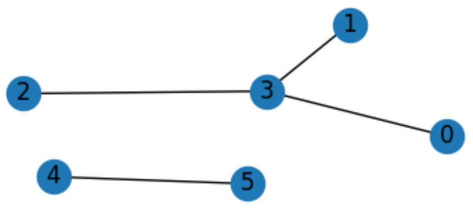

Fig. 5. Graph corresponding to fig.4.

IV. Evaluation and Results

A. Correlation graphs

Running the graph creation process for different values of τ leads to the results presented in fig.6. Note that the maximum amount of undirected edges possible is $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$, which in the case of 40 companies is 820. One can observe that increasing the value for τ decreases the amount of edges, as expected. For the chosen τ of 0.9, the average amount of edges per graph is 96 and for τ of 0.8 is 199.

Note also that the graph with $\tau = 0.9$ (fig. 8) has a lot of islands, and especially single node islands. The original model does not support single islands, and had no islands at all in the original data set. This was the main reason to create another dataset, with $\tau = 0.8$ (fig. 7). Note that these graphs have less

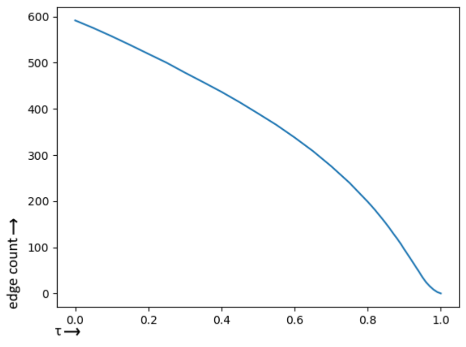

Fig. 6. Average edge count per graph versus τ

islands and (as seen also from fig.6), on average, double the edge count. All the 78 graphs with $\tau = 0.9$ formed the data set `companies090` and all the 78 graphs with $\tau = 0.8$ created the data set `companies080`.

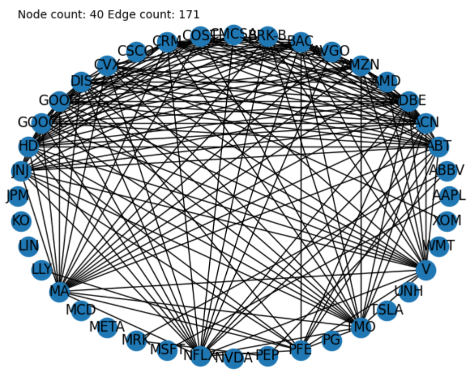

Fig. 7. Correlation graph with $\tau = 0.8$

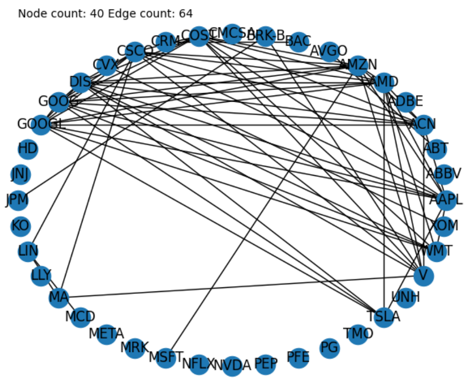

Fig. 8. Correlation graph with $\tau = 0.9$

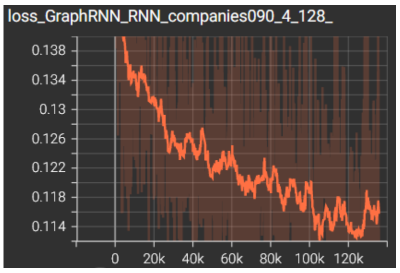

Fig. 9. Loss for data set community4

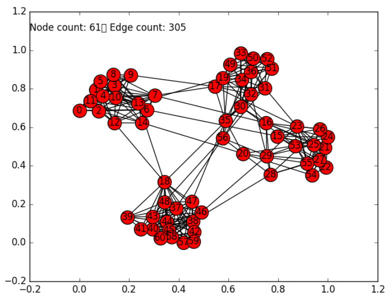

Fig. 10. Generated graph from data set community4

B. *RNN Training Results*

The very first action to confirm the correct working of the virtual environment, and of the model itself, is to train the model on the community4 data set. In fig.9 it can be seen that the loss steadily decreases with training. In fig.10 is a generated graph of the community4 trained model, as the name community4 suggests, there are four communities, all interconnected to each other.

The next step is training on the companies080 and companies090 data sets. The smoothed loss lines in fig.11 and reffig:loss-090 show that the model consistently improves itself by training. The decrease from 0.14 to 0.11 shows an increase of 20% in predictive capabilities. It can also be seen that the model struggles capturing the graph properties of companies090 consistently. In table II the NLL for train and test are very similar. This proves that the model is capable on graphs outside of the train set.

TABLE II
NEGATIVE LOG-LIKELIHOOD(NLL)

	companies080		companies090	
epoch	Train NLL	Test NLL	Train NLL	Test NLL
3000	123.45	123.13	80.81	80.78
6000	117.57	117.87	77.11	77.13
9000	113.79	114.01	74.80	74.64

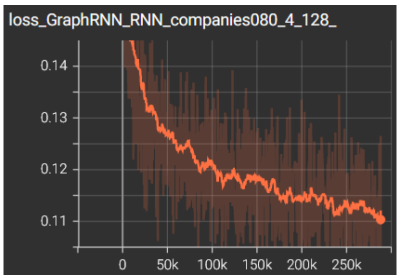

Fig. 11. Loss for dataset companies080

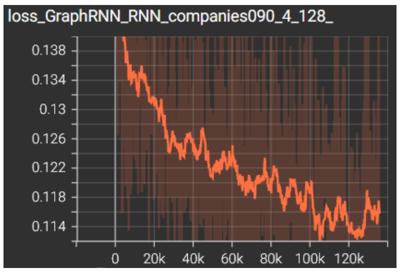

Fig. 12. Loss for dataset companies090

C. *Creating graphs*

During the initial graph creation, I noticed that the graphs would always be connected and be of smaller size than requested. This happens because islands are removed from the generated graphs in the original GraphRNN application. However, my dataset contains a lot of islands, which are likely relevant for the financial case. Therefore, I updated the generation code to allow islands.

Two examples of generated graphs are presented in fig.13 and reffig:gen-090. The difference in edge count due to the difference in threshold is immediately visible. In table III the MMD on

TABLE III
MMD AT EPOCH 9000

Dataset	MMD Degree	Cluster	Orbit
companies080	0.036	0.355	0.073
companies090	0.067	0.334	0.081

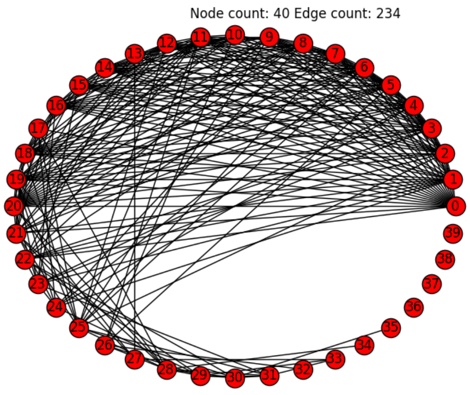

Fig. 13. Generated graph with $\tau = 0.8$

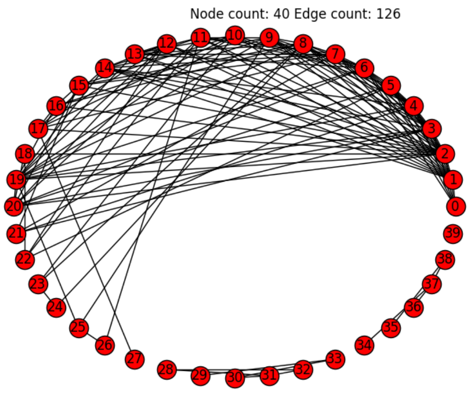

Fig. 14. Generated graph with $\tau = 0.9$

these graph properties tells us that the graph is comparable to what A. Khajenezhad et al. found for GraphRNN[15]. However, the Cluster MMD is higher than what J. You et al. found[2]. This might be explained by the fact that the MMD scores are highly dependent on the data set. The only significant difference between data sets is the Degree

MMD. It is likely that since there are more edges the model is able to learn more about where and when to place them.

I executed one final test, to measure the speed of the model. The model is able to create 1024 graphs in 479 milliseconds on the DAS-5, using an NVIDIA TitanX video card. Speed is important, because the more time series are accepted by the filter, the larger the data set for investment strategy training can be.

V. Porting to FPGA

A. *Goal*

All major deep learning frameworks support GPU compute to enhance the speed of training and inferencing AI models. However, computing is not limited to CPUs and GPUs. In recent years, FPGAs have been successful as accelerators, also for AI workloads. FPGAs can provide efficient inference by altering the logic circuit on their chip, thus creating performance by custom parallel computations or data representation. For example, E. Nurvitadhi et al. have shown that FPGAs can be very efficient for running GRU models [13]. With gained efficiency, more graphs can be generated and energy consumption can be reduced. Therefore, My goal is to provide an FPGA version of the model.

B. *Attempts and tools*

To convert the trained GRU model to an FPGA configuration requires either low-level programming using a hardware description language (HDL), or high level synthesis (HLS), translating high level code like python to an HDL specification. hls4ml is a framework designed for HLS in a deep learning context; the architecture of the framework is presented in fig. 15.

As seen in figure 15, the framework requires the model in a specific format, which is then converted, in a couple of stages, into an HDL representation.

The original pytorch model failed to import to hls4ml. Another attempt was done with a model converted to the portable ONNX format [17], which is supported by hls4ml. However, I discovered the pytorch and ONNX converters do not support GRU layers[2]. However, the converter from keras does

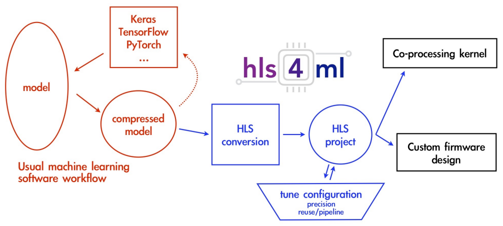

Fig. 15. Using the HLS4ML architecture [16].

support GRU layers[3]. Therefore, I attempted to convert the pytorch model to keras. Unfortunately, the converter from pyTorch to keras did not support the `ConstantOfShape` operation. which in turn meant that the conversion to Keras failed.

C. *Current status and future work*

At this time it is not possible to convert the pytorch GRU model to HDL. However, hls4ml is in active development, and it is likely support for other converters and/or for GRU layers will improve.

VI. Conclusion

My work has shown that recurrent neural networks are able to generate similar, yet different, company correlation graphs. These artificial graphs will help in the creation of better synthetic stock data. First, I extract company correlation graphs from raw stock data. I implemented a recurrent neural network, trained to learn the node and edge distributions. Finally, this trained model generates new company correlation graphs.

I have encountered a couple of limitations during my research; addressing these factors could enhance the results of this paper. First, the dataset I used was limited to one sample per day. A data set with higher frequency, where data points are collected at intervals in the range of minutes instead of a day, will ensure better correlation metrics. Furthermore, I have selected a specific conversion method from time-series to correlation graphs; other methods could also be explored. Finally, I have selected a specific model architecture, dictated by the GraphRNN implementation. More exploration of the model could further improve graph generation.

[2]Determined by source code inspection at https://github.com/fastmachinelearning/hls4ml/tree/main/hls4ml/converters.

[3]See footnote 2

There are also several aspects left for future work in my research. First, the correlation graphs could be further converted to time series. Connecting this section of the project to a conversion back to time series and the "reality check" filter would enable the model to be better optimized for its use case. It would also confirm whether the addition of generated graphs improves the quality of synthetic stock data.

Furthermore, the correlation graphs are put in a common pool of graphs. Instead of generating graphs that are similar, one could attempt to predict the state of the next correlation graph. An additional RNN model using the hidden state should be able to predict the next hidden state. This requires a lot more data and the data to be structured as a sequence of graphs instead of graphs on their own.

Ever since the release of "Attention is all you need" paper, transformer models have been replacing RNNs in a lot of areas [18]. Graph generation also experienced its introduction to transformer models with the Gransformer [15]. This model can generate models with lower MMD on some data sets and is known for capturing more complex dependencies. This model might also improve upon the companies080 and companies090 data sets.

The first company graph generator has been made. Yet this is just the opening of a field for graph generation for stock prediction applications.

Acknowledgements

I would like to express my gratitude to all the individuals who have contributed to the completion of this paper. First and foremost, I would like to extend my appreciation to my supervisor, Ana-Lucia Varbanescu, for her guidance and feedback. Additionally, I would like to acknowledge the contributions of Tristan Laan, who consulted on the difficulties surrounding the hls4ml framework. I am grateful for their aid.

References

[1] M. Reus, "code." [Online]. Available: https://github.com/Mathi18R/code

[2] J. You, R. Ying, X. Ren, W. L. Hamilton, and J. Leskovec, "GraphRNN: Generating Realistic Graphs with Deep Auto-regressive Models," 2018.

[3] Yahoo!, "Yahoo! finance," accessed: May 30, 2023. [Online]. Available: https://finance.yahoo.com/

[4] P. Ltd., "Peracton," accessed: June 23, 2023. [Online]. Available: https://peracton.com/

[5] Slickcharts, "S&p 500 companies - s&p 500 index components by market cap," accessed: May 30, 2023. [Online]. Available: https://www.slickcharts.com/sp500

[6] L. Lacasa, B. Luque, F. Ballesteros, J. Luque, and J. C. Nuño, "From time series to complex networks: The visibility graph," *Proceedings of the National Academy of Sciences*, vol. 105, no. 13, pp. 4972–4975, 2008. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.0709247105

[7] V. F. Silva, M. E. Silva, P. Ribeiro, and F. Silva, "Time series analysis via network science: Concepts and algorithms," *WIREs Data Mining and Knowledge Discovery*, vol. 11, no. 3, p. e1404, 2021. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1404

[8] X. Yin, D. Yan, A. Almudaifer, S. Yan, and Y. Zhou, "Forecasting stock prices using stock correlation graph: A graph convolutional network approach," in *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021.

[9] M. Reus, "Large scale graph generation with user defined properties," 2023.

[10] Y. Zhu, Y. Du, Y. Wang, Y. Xu, J. Zhang, Q. Liu, and S. Wu, "A survey on deep graph generation: Methods and applications," 2022.

[11] D. Chakrabarti, Y. Zhany, and C. Faloutsosz, "R-mat: A recursive model for graph mining," 2004.

[12] A. Gutfraind, I. Safro, and L. A. Meyers, "Multiscale network generation," 2015.

[13] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of fpga, cpu, gpu, and asic," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–4.

[14] VU Amsterdam, "DAS-5," accessed: June 26, 2023. [Online]. Available: https://www.cs.vu.nl/das5/

[15] A. Khajenezhad, S. A. Osia, M. Karimian, and H. Beigy, "Gransformer: Transformer-based graph generation," 2022.

[16] "hls4ML," accessed: June 26, 2023. [Online]. Available: https://cms-ml.github.io/documentation/inference/hls4ml.html

[17] "ONNX documentation," accessed: June 30, 2023. [Online]. Available: https://onnx.ai/onnx/intro/

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

VII. Appendix

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & & & & \\ 0 & 0 & & & \\ 1 & 1 & 1 & & \\ & 0 & 0 & 0 & \\ & & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & & & & \\ 0 & 0 & & & \\ 1 & 1 & 1 & & \\ 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & & & & & \\ 0 & 0 & & & & \\ 0 & 0 & 0 & & & \\ 1 & 1 & 1 & 0 & & \\ 0 & 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Fig. 16. Example of raw GraphRNN output and its converted adjacency matrix

To go from RNN output to adjacency matrix each row of the encoding is flipped and inserted with an offset. The diagonal is then filled with zeroes. This implies there is a maximum amount of edges per node determined by the encoding. A main diagonal is added ensuring no encoding space is wasted to self loops. Then to make the graph undirected it is mirrored across this new main diagonal.