Graph Neural Networks for Asset Management*

Grégoire Pacreau
Quantitative Research
Amundi Asset Management
gregoire.pacreau@ens-rennes.fr

Edmond Lezmi
Quantitative Research
Amundi Asset Management, Paris
edmond.lezmi@amundi.com

Jiali Xu
Quantitative Research
Amundi Asset Management, Paris
jiali.xu@amundi.com

November 2021

Abstract

It is impossible to analyze an asset taken in isolation, without taking into account the wider picture of the market. This fact is behind the extensive use of copulas or vector autoregressive models in finance, which allow to model dependencies between assets. In this paper, we look at the graph-based method to model inter-asset behavior. Graphs are ubiquitous when representing relationships, whether to model social network interactions, disease spread, traffic, or supply chain information. It allows for a very intuitive representation of market interconnections. We show how several types of market information can be translated into graphs and show some graph-based tools for market analysis. Furthermore, neural convolution layers have been developed which allow for more expressive neural models. Just like Euclidean convolution layers on images, they promise to contextualise each individual asset during prediction. We show the effect of three graph neural layers on the stock return forecasting problem. Using these forecasts, we build portfolios and show that graph layers act as a stabilizer to classical methods like LSTM, reducing transaction costs and filtering out high-frequency signals. We also study the effect of different graph-based information on the forecast and observe that in 2021, supply chain information becomes much more informative than sectoral or correlation-based graphs.

Keywords: Machine learning, graph learning, hypergraph learning, graph neural networks, graph convolutional networks, graph attention networks, quantitative asset management, backtesting, trading strategy.

JEL classification: C45, C53, G11.

*We would like to thank Factset for providing us with the Revere supply chain database and Thierry Roncalli for his valuable advice on writing this paper. We also thank Mohamed Ben Slimane and Takaya Sekine for their help accessing the various datasets we used in this work, and Fréderic Lepetit for his insight in the world of equity trading.

1 Introduction

Deep Learning performs well on problems like image recognition, where the algorithms can interpret each pixel as part of a bigger picture. So called convolution layers analyze every part of the picture in the context of the neighboring pixels. This allows generative algorithms to correctly reproduce the structure of photographs and paintings in a realistic and organic manner. Just like pixels in an image, financial assets do not exist in isolation. Their analysis benefits from studying the underlying structure of the markets. For instance, the correlation between assets must be taken in account when controlling a portfolio's exposure to market risk, with tools such as copulas. Other types of relationships between assets are also essential to build a complete picture of the market inter dependencies. If one wants to build investment portfolios using machine learning, this structural information must be incorporated.

The main issue when trying to transcribe a market's structure is that, contrary to images, it is hard to represent it in a Euclidean space. Yet, the convolution layers for vectors, images and shapes are based on the Euclidean distance. Each element is convolved with those closest to it. Market relationships are more complex, they occur at different level, each with different strength and relevance. Relationships can be Boolean, for instance if two stocks are from similar sectors, or real-valued if they are linked by correlation. The most natural way to represent this diversity of structures is through graphs.

Recently, graph neural networks have gathered some attention from researchers in many fields to model non-Euclidean relationships. Graph neural layers are now fundamental in the study of medical molecules (Stokes et al., 2020), they can model traffic (Wu et al., 2018) and analyze relationships in bibliography, etc. They enable the structure of the data to be encoded independently from their individual features, relying on the propagation of the information through the graph. In particular, graphs arise naturally when looking at supply chain information (Wu and Birge, 2014).

In this paper, we examine how graphs can model inter-stock relationships to inform portfolio construction. Our study looked at several deterministic uses of graph for asset management, but its primary focus was on graph neural networks. Section 2 provides a brief introduction to graph theory with some examples of naturally occurring graph structures in finance. Section 3 builds upon this introduction to introduce neural layers that profit from this structure to propagate information. Section 4 extends the ideas of propagation to a generalization of graphs, which also come with its neural layer. Finally, Section 5 provides our experimental setup and Section 6 our results.

2 Information propagation with graphs

Many forecasting techniques focus on the temporal context to produce an estimate. However, financial assets do not exist in a vacuum and more information than a stock's history is required to assess its movement. A spatial representation of stocks can provide insights into their dynamic, which is why we are trying to apply graph theory to stock markets.

2.1 Convolution operations in Euclidean space

The need for the spatial propagation in complex forecasting or classification tasks is widely accepted in many machine learning problems. For instance, this fact is central in image processing, where each pixel needs to be coherent with the ones surrounding it. This is done through convolution layers, which compute a value from the cross-correlation between

a pixel and its neighbors. The generic formula for a convolution layer on an image X for the pixel at location (i, j) is given by:

$$Y_{i,j} = \sigma\left(\sum_{a=-M}^{M}\sum_{b=-M}^{M} W_{a,b}X_{i+a,j+b}\right) \tag{1}$$

where $Y_{i,j}$ is the result of the convolution of image X at pixel position (i, j). $M = \left\lfloor\frac{m-1}{2}\right\rfloor$ and m stands for the convolution kernel size, which controls the number of neighboring pixels seen to compute the end value for each pixel. σ is a nonlinear function, $W_{a,b}$ is a learnable weight matrix depending on the relative position of the neighbor pixel to the studied pixel. The added context provided by these layers brings impressive results for noise removal, image compression and reconstruction, etc.

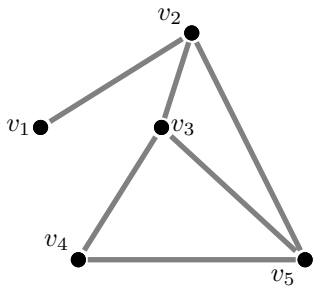
This family of convolution layers is defined for Euclidean spaces and makes sense when the only relationship between the inputs is their physical proximity in an Euclidean space. However, financial assets have more than one way being close to each other. Many different relationships might link assets together. When dealing with stocks for instance, one might be interested in the sector in which the company operates to compare it to others in the same situation, or one might be interested in the health of its customers and competitors, or even whether this enterprise is actually a joint venture. If one is to consider such a wide variety of information, a more complex representation of this information is required than that provided by a Euclidean space, where the curse of dimensionality will make it difficult to define a relevant similarity measure. However, if we see the markets as a network of assets linked together by these relationships, then we can easily represent this ecosystem using graphs, which have the benefit of coming with their own convolution operator.

2.2 A primer on graph theory

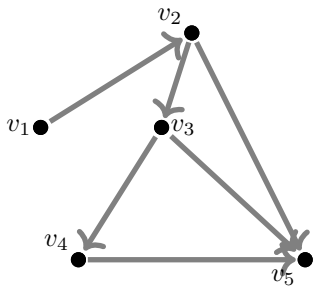
Graphs are a mathematical object reported to have been invented by Leonard Euler in 1735 to model bridges over a river in Königsberg. It is composed of a set of vertices, V, which are connected by a set of edges E. More specifically, the set E is defined as being a subset of $V \times V$, where $\times$ is the Cartesian product between sets. For $i, j \in V$, there exists a connection between i and j if, and only if, $(i, j) \in E$. A graph can also be represented by a square matrix, named the adjacency matrix A, such that $A_{i,j} \neq 0 \iff (i, j) \in E$. The values in the adjacency matrix can act as the weights of the connections, where higher values mean a stronger connection. A non weighted graph will have a binary adjacency matrix. If A is symmetric then the graph is called undirected, since information flows from i to j and from j to i indifferently. Otherwise it is called a directed graph. Other special kinds of graphs include the bipartite graph, where vertices can be separated into two classes where no two vertices in a same class are connected, and complete graphs, where every vertex is connected to every other. Some examples are given in Figure 1.

Graph theory provides several tools for the study of graphs. Subsets like network theory or spectral graph theory focus on propagation in a graph, with applications in epidemiology (Cutura et al., 2020; Sehanobish et al., 2021) or traffic forecasting (Wu et al., 2018). The degree matrix is a matrix fundamental in describing a graph's structure. The degree matrix D is a diagonal matrix which gives the number of neighbors of each vertex. It is directly related to the adjacency matrix, since $\forall i \in \{1, \ldots, N\}, D_{i,i} = \sum_{j=1}^{N} A_{i,j}$. It allows to build the graph Laplacian, which is defined as $L = D - A$. Since the diagonal in the adjacency matrix is 0, it combines without loss the information about the number and the position of each vertex's neighbors. The graph Laplacian acts like a regular Laplacian operator for

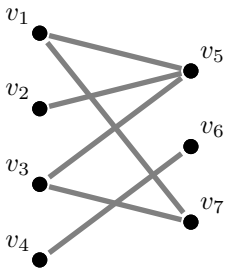
Figure 1: Examples of graph



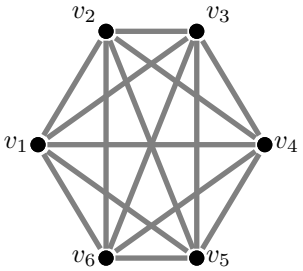
(a) An undirected graph with five vertices



(b) The same graph with directed edges



(c) A bipartite graph



(d) The complete graph with six vertices

describing propagation through differential equations. For instance, one can show that heat propagation in a graph verifies:

$$\frac{d\phi}{dt} + kL\phi = 0 \tag{2}$$

where ϕ is the temperature vector of the graph's nodes and k is the heat capacity of the network. Furthermore, the eigenvalues of the graph Laplacian inform us on the structure of the graph. The number of connected components, i.e. the number of unlinked subsets of vertices, is given by the multiplicity of the eigenvalue 0. In other words, if there is a path from every vertex to another, then 0 will not be an eigenvalue of the Laplacian. The second smallest eigenvalue of the Laplacian is also an indicator of the conductance of the graph[1], which is a measure of the connectivity in the graph. The higher the conductance, the faster an information is transmitted to the entirety of the graph. The Laplacian matrix has many other properties, some of which will be further discussed in the following sections and in Appendix B on page 41.

To illustrate further the concept of graphs and to prove their relevance in finance, we can show that a vector-autoregressive (VAR) model can be easily represented as a weighted directed graph where vertices are time series. Let X_t be a N-dimensional time series following a VAR(1) process, i.e. for $A \in \mathbb{R}^N$ and ϵ_t a white Gaussian noise, $X_{t+1} = AX_t + \epsilon_t$. Then A can be interpreted as an adjacency matrix representing the links between assets. At every time step, information flows through the edges and updates each vertex's value. This can be extended to any kind of VAR models, as demonstrated in Calkin and Lopez de Prado (2014). In practice, however, these graphs grow exponentially in the number of time steps and quickly become impractical when dealing with several stocks. In our study, we will not

[1] See Appendix D of page 49.

represent temporal dependencies through graphs, but only so-called spatial relationships between stocks, with temporal behavior being embedded beforehand.

2.3 Message passing

The operation consisting in sending information from one node to another is called Message Passing. This family of mechanisms will act like the convolution operations in Euclidean spaces, whereby they allow for the dissemination of information to neighboring nodes. They were introduced by Battaglia et al. (2016) and Gilmer et al. (2017).

Message passing can be separated into three steps: the message creation, the aggregation and the update. In the message creation, the information of a node is transformed into an embedding. Then the aggregation step disseminates the message to every neighbor of the node. Finally, the update step will look at every received message and modify the value of the node accordingly. More formally, the three steps can be modelled as functions. Let $\mathcal{X}$ be the space of the nodes' information and $\mathcal{Y}$ the message space. Then the message creation step is defined by a function $message : \mathcal{X} \mapsto \mathcal{Y}$ and the update step by a function $update : \mathcal{Y} \mapsto \mathcal{X}$. Let us suppose that we have a graph where every node i contains a vector of real numbers $\vec{v}_{i,t}$ at time t. Let $\mathcal{N}$ be a function that, given a node i, returns the set of neighbors of i. The aggregation operator can be any function $f : \mathcal{Y} \mapsto \mathcal{Y}$ such that:

$$\vec{v}_{i,t+1} = f(\{\vec{v}_{j,t}, j \in \mathcal{N}(i)\}) \tag{3}$$

Furthermore, f must be node permutation invariant. Since there isn't any natural order over the neighbors of a node, the behavior of f should not change according to the order in which the messages arrive. For instance, the averaging operator on neighbors can be a message passing function:

$$\vec{v}_{i,t+1} = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \vec{v}_{j,t} \tag{4}$$

This simple averaging message passing mechanism helped define the SAGE neural layer, a precursor in graph neural networks (Hamilton et al., 2018). Indeed, a graph neural layer is simply a differentiable parametrised message passing mechanism. In the case of the SAGE layer, the parameters are weights in the average. In Section 3, we will see some more advanced message passing algorithms and their related graph neural layer. In particular, the graph Laplacian and its links to heat dispersion in the graph allow us to construct robust message passing.

Remark 1. *Some papers try to define message passing with node permutation sensitive functions or even propagation mechanisms that are not message passing (Hamilton, 2020). However, the majority of the literature abides by the message passing theory of Gilmer et al. (2017).*

2.4 Creating graphs for portfolio management

Relational information can easily be transcribed into graph form. We provide three graphs built using financial information on the MSCI stocks.

2.4.1 Sector graph

Sector information like the GICS sector index can easily be transcribed into a graph. Indeed, we can choose edges such that nodes are linked if and only if the assets are from the same

sector: $V = \{(i,j), i \text{ and } j \text{ are in the same sector}\}$. Taking the highest capitalizations of the MSCI index, Figure 2a at page 7 is the resulting adjacency matrix, with white meaning $A_{i,j} = 0$ (*i.e.* $(i,j) \notin V$) and black $A_{i,j} = 1$ (ie $(i,j) \in V$). This relationship leads to several sets of complete subgraphs, named cliques. In Figure 2a, each black square forms a clique. This translates in Figure 2d to several unconnected subgraphs. One problem that may arise from such a representation is that complete graphs see a polynomial growth of the number of edges. Indeed, for N the number of vertices, a complete graph will have $\frac{N(N-1)}{2}$ edges. For large graphs, this may become a bottleneck in terms of computation time. We will see in Section 4.3.3 that a more parsimonious representation can be made of sectoral information using a generalization of graphs named hypergraphs.

2.4.2 Correlation graph

Correlation or any similarity measure can be transformed into graph form, either by directly taking the cross-correlation matrix as a weighted adjacency matrix, or by creating a binary matrix through thresholding. Figure 2b displays such a binary adjacency matrix, where $A_{i,j} = 1$ if and only if the absolute value of the correlation between the two assets is greater than 0.5. This kind of graph is particularly interesting since they are easy to create and do not rely on outside information. Similar matrices can be built using other similarity measures, such as mutual entropy (Dionisio et al., 2004; Cellucci et al., 2005) or dynamic time warping (Sakoe and Chiba, 1978), a similarity measure widely used for time series. Whatever the measure, the threshold must be chosen *ad hoc*, though some papers choose it by examining the graph's structure for various values. A good idea would be to examine the number of connected components for various thresholds. If it is close to the number of vertices, then the threshold is too high and almost no edges are created. If this number is close to one, then it is likely that the threshold is too low, and the graph is almost complete.

2.4.3 Supply chain graph

Lastly, any network information about an asset can be transformed into a graph. We decided to showcase this fact using supply chain relationships. Similarly to the previous graphs, we simply build an adjacency matrix such that $A_{i,j} = 1$ if, and only if, there exists a link between company i and company j in the FactSet Revere database[2]. To illustrate this, we take Revere's supply chain database and extract the supplier and customer relationships. Graphs are a natural representation for such information, since they maintain the structure of the network of relationships. Several studies have investigated supply chain graphs for financial purposes (Cohen and Frazzini, 2008; Bloomberg Quant Research, 2020). Figure 2c shows the adjacency matrix where $A_{i,j} = 1$ if and only if i is either a customer or a supplier of j. Figure 2f provides the resulting graph.

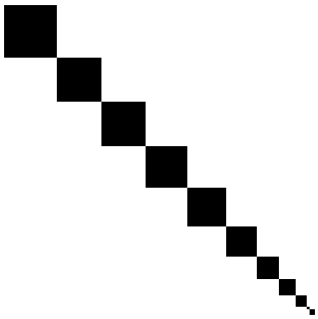
Supply chain information is not the only one that can be transformed into a graph. For instance, papers Kim et al. (2019) and Feng et al. (2019) create graphs using information such as "is the company owned by another?", "has a fund invested in both these companies?" or "do these companies trade in the same country?". Since the number of possible graphs linking assets is high, most studies use a combination of several criteria to increase accuracy.

2.4.4 Combining graphs

We often find ourselves with several relevant relationships, each leading to their own graph. It is then possible to combine the information from several graphs, either by averaging the

[2]More information at https://open.factset.com/products/factset-supply-chain-relationships/en-us.

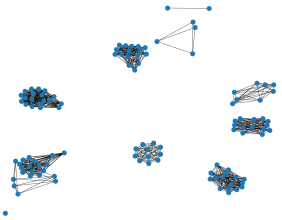
Figure 2: Three graphs computed on the highest capitalizations of the MSCI World index



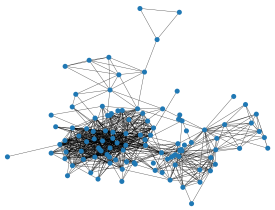
(a) Same sector adjacency matrix using the GICS sector index

(b) Correlation based adjacency matrix, thresholded at 50%

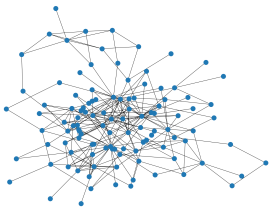
(c) Supply chain based adjacency matrix using the FactSet Revere database



(d) Representation of the sector graph



(e) Representation of the correlation graph



(f) Representation of the supply chain based graph

values of their adjacency matrices or successively transmitting the information.

Averaging makes sense if the relationships are on the same level. In other words, if the relationships are just a way to join two companies' information and that the order or the importance of the relationship does not matter, then we can simply combine the matrices by taking the union of their set of edges. If the graph is weighted, then the weight of each edge will be the average of the weights of this edge in all graphs. If we take the case of a discrete time process, this would represent the following structure:

$$X_{t+1} = \frac{1}{2}\left(AX_t + BX_t\right) + \epsilon_t \tag{5}$$

where A and B are two adjacency matrices. The second makes sense if there is a hierarchy between relationships. This is the method used in Kim et al. (2019), where their graphs have a different granularity. For a given company, they separate sectoral information from company to company information, first sharing the information inside each sector according to the graph of the latter relationships, and then in a second time sharing between sectors. For our discrete process example, this method would be equivalent to the following model:

$$\begin{aligned} X_{t+1} &= AY_t + \epsilon_t^1 \\ Y_t &= BX_t + \epsilon_t^2 \end{aligned}$$

In our study we use machine learning algorithms based on the graphical structure of our data. In this case, instead of averaging, since the algorithm is able to learn the relative importance of edges, we simply take the union of the edge sets, or in the case of binary

Figure 3: Toy graph representing fictitious sector (in grey) and supply chain (in red) relationships

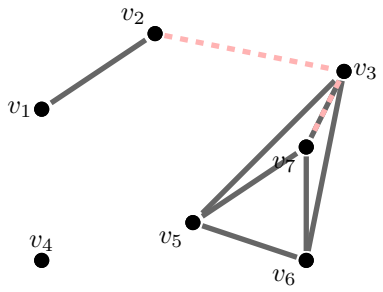
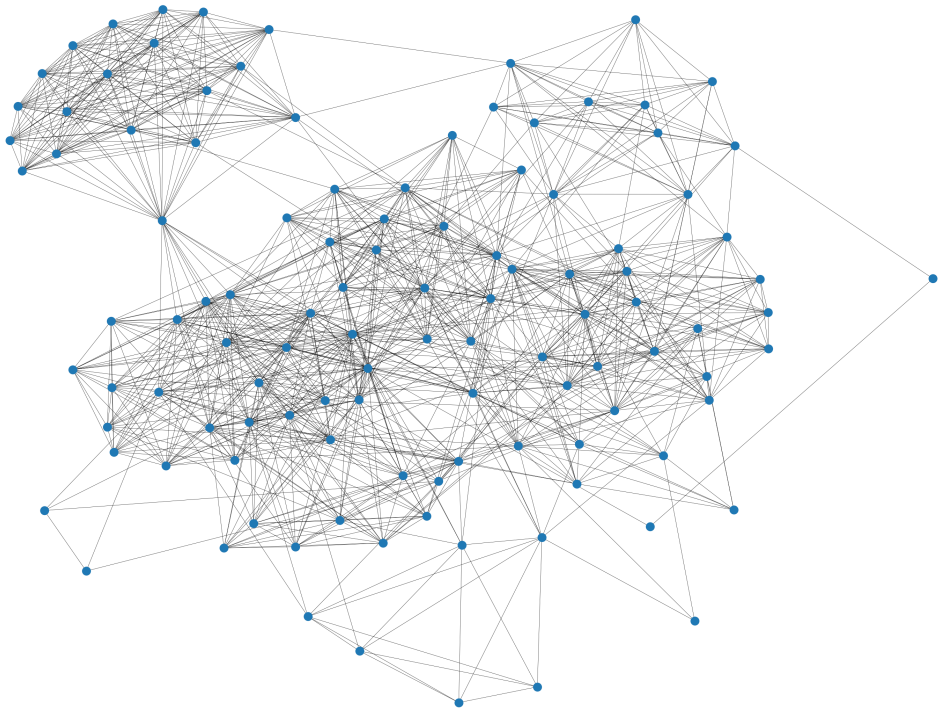


Figure 4: Kamada-Kawai representation (from the NetworkX package) of the union graph on the MSCI World index



valued matrices the element-wise maximum of the adjacency matrices. This reduces the sparsity of the adjacency matrix but gives more leeway to the learning algorithm.

A representation of what a union graph would look like is given in Figure 3. It shows a toy example of the union of the sector and supply chain graph on seven fictitious assets. The supply relationship links v_2, v_3 and v_7 in a dashed red line, while sectors are represented in grey. We see that although v_3 and v_7 are linked both by supply chain and sector relationships, there is only one line. That is because the union between two graphs is the union between two sets, which does not allow for redundancy. The union graph of the three matrices we described is provided in Figure 4.

2.5 Deterministic portfolio creation using graph theory

Before we look into machine learning algorithms for graphs, it is interesting to see how deterministic methods applied on graphs can help in portfolio construction. We will detail three different uses of graph theory in asset management.

2.5.1 Centrality and supply chain

Since a graph displays the dependencies between assets, it allows for the easy propagation of information between assets. There are many ways to propagate information. The simplest set of techniques involve applying aggregation functions on the information of the neighbors. This is the notion behind message passing algorithms. Seeing the simultaneous evolution of the asset information and its closest neighbors gives more context for decision making. A form of max-pooling message passing for strategy building was shown in Bloomberg Quant Research (2020), where the performance of each company's most important client is used to correct the predicted returns of their stocks. This means that if a client accounting for more than 10% of the total profits of the company has its value fall, this will be reflected in the graph and sent to the node. Cohen and Frazzini (2008) showed that for supply chain data, there exist a natural inertia in the markets which makes an important customer's bad performance only reflect on the supplier after about a month. Bloomberg Quant Research (2020) found that with the graph structure, they can identify these movements almost immediately and can act on this information in time.

In any propagation problem, the centrality of a node will have a high influence of how sensitive this node is to changes in the market. Centrality measures quantify how connected a node is to the rest of the graph. A high centrality measure can indicate that the node has many neighbors, or that it is the neighbor of a very connected subgraph. For example, in a social network, a highly central user would be one having many followers. Similarly in traffic forecasting, blocking a central junction will have a high impact on the overall traffic. For example, in Figure 3, the node v_7 is more central than v_1. Cohen and Frazzini (2008); Wu and Birge (2014) and Wu (2015) have studied the way in which centrality measures can reveal the exposure of a stock to market fluctuations by estimating how easily a shock will be propagated towards it. In particular, together with Bloomberg Quant Research (2020), they provide a series of linear regressions involving centrality measures and graph structural information and they show that the added returns they provide are independent from the common Fama-French market factors.

The centrality of a node can be defined in several ways. The simplest form of centrality is degree centrality:

$$C_d(i) = d(i) = \sum_{j \in V} A_{i,j} \tag{6}$$

which defines as central the nodes having the most neighbors in the graph. In a directed graph, this can be turned into two different measures according to whether we define as neighbors the mother nodes ($\{i, (i,j) \in E\}$) or the daughter nodes ($\{j, (i,j) \in E\}$). Other measures include eigenvector centrality:

$$C_e(i) = \frac{1}{\lambda} \sum_{j \in V} A_{i,j} C_e(j) \Leftrightarrow A\mathbf{C}_e = \lambda \mathbf{C}_e \tag{7}$$

which extends degree centrality to infinite walks in the graph. Here λ is a predetermined eigenvalue of the adjacency matrix, often either the smallest eigenvalue or the second smallest. Eigenvector centrality inspired Google's PageRank centrality measure:

$$C_p(i) = \frac{1}{\lambda} \sum_{j \in V} A_{i,j} \frac{C_p(j)}{\max\left(1, \sum_{k \in V} A_{k,j}\right)} \tag{8}$$

which weights the score of each neighbor by their own number of neighbors. Wu (2015) provides a short comparison of several centrality measures on the supply chain network. In this paper, degree centrality favours tech companies, the father centric (in their case counting the number of customers) favours financial institutions and the son centric (the number of suppliers) retailers such as WallMart. The paper then builds long only equal weighted portfolios on the most central nodes for each measure and computes the regression against the S&P 500 index. They show that the centrality measures are not always correlated to one another and that excess returns on portfolios created from the supplier centrality predict future stock returns.

2.5.2 Strategy on clustering

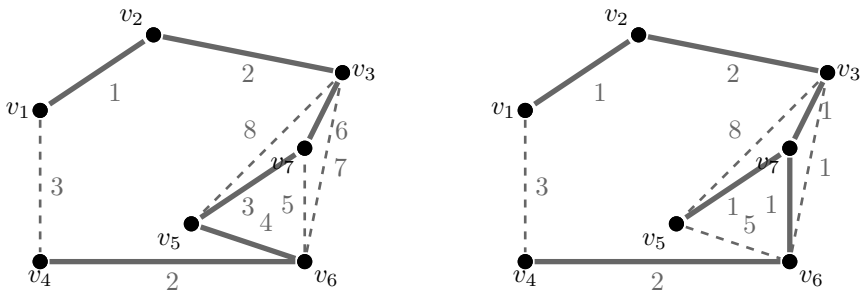
A simple way to use graph theory for portfolio construction is through the use of clustering. Graph clustering has been used mostly in image processing, but some research is starting to introduce time series clustering through graph theory (Zhu et al., 2016). Shen et al. (2012) provided a way to predict stock movements according to a clustering of the stock universe. Though the paper builds its clusters on a hypergraph, the general idea is still relevant for graphs.

Graph clustering can be done in a number of ways. The classic method is through the min-cut problem[3]. This graph theory problem takes a weighted graph and tries to find a set of edges which would separate the graph into two unlinked sets of vertices. This set of edges is named a cut. A good clustering method is to look for the cut that minimizes a given measure. Classic algorithms include the Stoer-Wagner method or the probabilistic Kagner's algorithm.

We can also use spectral clustering algorithms, based on the eigenvalues of the Laplacian matrix. In particular, a family of clustering techniques use an identity called Cheeger's inequality[4], which grants efficient bipartite clusterings according to the second eigenvalue of the Laplacian. Laplacian based decomposition can be extended so that any number of clusters can be achieved, not only two. The idea is to apply classic clustering techniques onto the Euclidean world defined by the eigenvectors of the Laplacian instead of the non-Euclidean world of graph nodes and edges. These clustering techniques are very efficient. Indeed, computing the Laplacian matrix is simply making the product and sum of matrices in which only one is non-diagonal. The cost of eigenvalue decomposition is then dominated by that of a matrix multiplication.

[3] See Appendix D at page 49.
[4] This inequality is presented in Appendix D at page 49.

Figure 5: Two weighted version of our toy graph and their MST found by Kruskal's algorithm (in bold)



(a) A weighted version of the toy graph (b) Different weights lead to a different MST

Graph clustering can then be used on assets for portfolio building. An idea developed in Dees et al. (2019) and Arroyo et al. (2021) is to cluster assets to compute the covariance of returns on each clusters separately. This allows for a faster and more stable estimation of the covariance matrix which can then be used in the optimization portfolio. It also leads to strategies on several equally weighted portfolios, each representing a cluster, themselves weighted in a global portfolio according to how many iterations of a clustering algorithm were needed to obtain the clusters (Dees et al., 2019).

2.5.3 Signal building using graph theory

It has been shown that structural information about graphs can reflect contractions in financial markets. Kaya (2015) generated a graph between several assets of different nature (commodities, stocks, etc.) using mutual information, but with a slightly different approach than ours. Instead of thresholding, they took the complete graph, weighted by the mutual information, and looked for the minimum spanning tree (MST). A tree is a particular family of graphs where there exists only one path from one vertex to another. The MST is defined in Cormen et al. (2009) as the tree where every path between two vertices is minimal. The MST can be found using Prim's or Kruskal's algorithm, whose precise description features in Cormen et al. (2009) and whose implementation exists in NetworkX (Hagberg et al., 2008).

Algorithm 1 Kruskal's algorithm for finding the minimum spanning tree (MST)

Require: $G = (V, E)$ a graph and $W : E \to \mathbb{R}$ the weights of each edge
  $E = \emptyset$
  $forest = \{\{i\}, i \in V\}$
  Sort all $e \in E$ in increasing values of $W(e)$
  for $(i, j) \in E$ (sorted) **do**
    if i and j not in the same tree of *forest* **then**
      $E = E \cup \{(i, j)\}$
      Drop the trees of i and j from *forest*
      Add the union of the dropped trees to *forest*
    end if
  end for
  return E

The idea behind computing the MST is that for a market where values of mutual entropy range widely, the MST will be a very deep tree, one where each node has few neighbors. On the contrary, for a market where all returns look alike, the MST will be very wide, with some central nodes having many neighbors. Figure 5 shows two examples of weights on the same graph, with the resulting MST computed with Kruskal's algorithm (algorithm (1)). We see that changes in the weights can lead to different topologies for the MST. This comes from the fact that node v_7 was closer to its neighbors than before, it illustrates the sort of contractions that Kaya (2015) focused on. Indeed, according to Kaya (2015), crisis lead to a contracted market where every asset-to-asset mutual entropy becomes alike. This allows for signals based on eccentricity measures on the nodes of the MST. Several measures exist to indicate how wide or tall a graph is. Eccentricity is a measure defined for every vertex. It computes the longest path to any other vertex. A low eccentricity means that every node is reachable in a small number of hops in the graph. It is linked with a high centrality. However, eccentricity is defined by node, for a graph level metric two eccentricity-based measures exist to describe the whole graph. The first is the diameter of a graph, which is the maximum eccentricity over every node. In other words, the diameter is the length of the longest possible path. A diameter of one would indicate that every node is connected to every other, i.e. the graph is complete. A second measure is the radius of a graph, which is the minimum of eccentricities. The radius gives information on how connected each node is to the most central nodes of the graph. In other words, whatever vertex we choose, there will be a vertex that will need at least as many hops as the radius to be reached.

The idea behind Kaya (2015) is to examine these measures for the MST. If the topology of the MST changes during a crisis, this will reflect on either the radius or the diameter. In the example of Figure 5, the first graph has a diameter of 6 (the maximum path is $(v_1, v_2, v_3, v_7, v_5, v_6, v_4)$) and a radius of 3. The second has a diameter of 5 (with path $(v_1, v_2, v_3, v_7, v_6, v_4)$) and a radius of 3. Kaya (2015) thus creates a signal on several assets ranging from commodities to bonds and stocks. Their study suggests that every asset category tends to be linked to the others of the same categories, except during crisis where they observe a more homogeneous behavior. In order to replicate their experiments on stocks only, we built signals based on the diameter and radius of the MST, as well as on the mean and maximum degree of the graphs' node. We did not notice any improvement in returns or on the Sharpe ratio when using this signal to balance an equally weighted portfolio. Our tests indicate that when dealing only with stocks, the variation of the eccentricity measures on the MST are too small to build a viable signal.

3 Graphs and neural networks

Although powerful, graphs are subject to a combinatorial explosion of their edges when increasing the number of nodes. Neural networks have proven very useful when forecasting N-dimensional time series using graph information.

3.1 Two graph convolutional layers

Since graph are more complex in structure than Euclidean spaces, the classical convolution layer (equation (1) at page 3) does not apply. Many alternatives exist in the literature. For instance, the Python library PyTorch Geometric (Rozemberczki et al., 2021) provides an extensive list of contenders with their torch implementation. The PyTorch library is a popular machine learning library which we used for all experiments (Paszke et al., 2019). We focused on two convolution layers that are both representative of the different approaches to graph convolution and are very popular in the literature.

3.1.1 Kipf and Welling's graph convolutional layer

Otherwise called the GCN layer, it is the more theoretically sound layer of the two. Introduced in Kipf and Welling (2017), it uses graph spectral theory and the graph Laplacian matrix to define the message passing mechanism through a graph. In this section we will provide the formula and some intuition of the layer, a more complete explanation of the layer is provided in Appendix B at page 41.

The convolution family of layers for Euclidean spaces, used for image processing (2D) or time series analysis (1D), is based on the cross-correlation operations from spectral theory. Many objects from the classical spectral theory can be generalized to the graph world, like for instance Fourier transforms and the Laplacian operator (Chen et al., 2021). The main object behind graph spectral theory and the convolution operator is the Laplacian matrix. Let A be the adjacency matrix of a graph and D its degree matrix. For stability reasons, the normalized Laplacian matrix $D^{-\frac{1}{2}}LD^{-\frac{1}{2}}$ is often preferred, which is simply a rescaled version of the Laplacian so that the diagonal terms are all equal to 1.

Appendix B at page 41 provides an introduction to the relevant graph spectral theory tools and proofs of the formula of the layer. Here is a quick summary. The convolution operator can be defined using the Fourier transform of a function over graphs and the eigenvectors of the Laplacian matrix. This leads to very computationally intensive analytical expression, which Kipf and Welling (2017) reduces to the following approximation. Let Θ be a matrix representing the convolution weights. A convolution layer for graph features X of output Y is:

$$Y = \sigma\left(\Theta * X\right) \sim \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta\right) \tag{9}$$

where $*$ is the graph convolution operator, $\tilde{A} = A + I_N$ is the adjacency of the graph with added self loops and $\tilde{D} = D + I_N$ its degree matrix. Here Θ is the only learnable parameter. This layer simulates one propagation step through the graph after applying a filter defined by the Θ matrix. Since all but two matrices are diagonal, the complexity of this layer's computations is dominated by the multiplication between an $N \times N$ matrix and the feature matrix, which is of size $N \times F$, with F the number of features of each node. As such, this layer has a $O(N^2F)$ complexity.

3.1.2 Graph attention layer (GAT)

Attention mechanisms are a very common tool in machine learning when dealing with high dimensions. The idea is to multiply each feature with a scalar named the attention coefficient. Just like in a classical regression, the higher the attention coefficient the more impact the features will have on the overall result of the network. This mechanism is very popular since it allows us to visualize the importance given by the network to a specific information. It is a mechanism that can be easily adapted to every data encoding, such as graphs, for instance. The idea behind graph attention is that the cross correlation between each node is weighted according to a learned attention coefficient (Velickovic et al., 2018). This attention coefficient is then a measure of the relevance of a specific node to another.

More formally, each attention coefficient is computed from a learnable weight matrix $\mathbf{W}$ and an attention vector $\boldsymbol{a}$, which are applied to the two nodes' features. The result is then put through a non-linear activation function, which is in Velickovic et al. (2018) a Leaky-ReLU function, and finally they are normalised using the softmax function. Let X_i be the node features and $[u||v]$ denote the concatenation operation between two vectors u and v. The complete formula of the attention coefficient of node j with regard to node i is as follows:

$$\alpha_{i,j} = \text{softmax}\left(\text{LeakyReLU}\left(\boldsymbol{a}^\top\left[\mathbf{W}X_i||\mathbf{W}X_j\right]\right)\right) \tag{10}$$

Once every coefficient is defined, we create the message consisting of a weighted average of the features of each node's neighbor. The aggregation of the neighbors' information can then be obtained by:

$$Y_i = \sigma\left(\sum_{j \in \mathcal{N}_i \cup i} \alpha_{i,j} \mathbf{W} X_j\right) \tag{11}$$

where σ is a non linear transformation. Alternatively, instead of averaging, the concatenation of features has been proposed if one wants to keep every information about the neighbors' state. In our regression and classification experiments, averaging was used. Another extension of the layer provided by Velickovic et al. (2018) is a multi-head approach, by which K independent attention mechanisms are trained independently and their results averaged. We did not use this extension in our study.

An interesting property of the layer, outside of its very intuitive message passing mechanism, is its computational complexity. Given F features, F' predicted features, $|V|$ the number of nodes and $|E|$ the number of edges, the complexity of the GAT computation is in $O(|V|FF' + |E|F')$. Since we will have fewer than ten features, and that the predicted feature is the stock's return, i.e. of size one, the computational complexity of the GAT layer is linear in the number of edges and vertices. For sparse adjacency matrices, this makes it a very computationally efficient layer. In particular, it is less computationally intensive than the GCN layer.

The main advantage of the graph attention-based layer is that one can extract the attention coefficients and visualize them. Thus, if the model contains a single GAT layer, we can partially explain the model's result. This is a rare thing in neural networks. When dealing with supply chain relationships in particular, using the NetworkX library one can produce a visualization of the supply chain graph at a given date weighted by the attention coefficients. This can provide an empirical validation in small practical examples where a neighbor's effect is well understood.

3.2 Discussion on the two layers

It is important to note that the GCN layer is a particular case of a convolutional filter over every node in the graph. In particular, in the construction of this layer (which is detailed in Appendix B at page 41) we assume that we can restrict ourselves to diffusion among the direct neighbors. This means that we can compare the GCN layer to the GAT layer directly. This is done in Bronstein et al. (2021), where they argue that the GAT layer is nothing but a more complex GCN layer. Indeed, the GCN layer takes into account every neighbor's features according to a fixed weight matrix Θ, whereas the GAT layer computes a specific attention coefficient for each edge.

Another interesting remark in Bronstein et al. (2021) is that both layers are a particular case of a message passing mechanism. Indeed, in the case of the GCN the message sent by each is its feature and the aggregation function is a weighted mean. For the GAT layer, the message is the feature times the attention coefficient between the sender node and the receiver node and the aggregation function another weighted mean. Bronstein et al. (2021) thus establishes the following classification:

$$\text{GCN} \subset \text{GAT} \subset \text{message passing} \tag{12}$$

GATs have several advantages over GCNs (Velickovic et al., 2018). They can handle directed graphs for instance, whereas the GCN is built assuming that the Laplacian matrix is a symetric semi-definite matrix, which in turn limits its use to undirected graphs. The GAT,

as with all attention networks, is robust to high dimensional data. In particular, a highly connected graph will be better handled by the GAT layer since the attention mechanism will select particularly relevant connections amidst all the neighbors. On the contrary, the fixed weights of the GCN will lead to a very fast averaging of every nodes' information in a densely connected graph, erasing all the specificities of individual nodes. This issue is known as the smoothing problem (Bronstein et al., 2021; Hamilton, 2020) and can affect GAT as well, but to a lesser extent. Finally, GAT layers are more flexible to changes in the graph. The attention mechanism is independent from the edges of the nodes' position, it computes for every two nodes a new coefficient at every iteration. As such, removing or adding edges will have few impact as long as the mechanism can compute a coefficient over the newly connected edges.

On the other hand, the smoothing of information provided by GCNs can be desirable in some cases. GCN originate from a low order Chebyshev expansion of the convolution of the features with a filter. The frequency profile of this layer is thus known analytically as demonstrated in Appendix E at page 50. Its spectral behavior cuts some bands of frequencies in the middle to high range (Muhammet et al., 2020). In financial forecasting, this can stabilize the predictions by cutting out outliers in the market.

3.3 Adding temporal dependencies

Our graphs are a representation of spatial relationships and do not account for temporal dependencies. Even though temporal relationships can be represented using graphs (Calkin and Lopez de Prado, 2014), this dramatically increases the number of vertices and nodes, leading to an explosion of computation time. Instead, we decide to embed temporal information using a recurrent neural layer. This temporal embedding is the message creation function of our message passing mechanism.

The majority of the literature includes temporal information in the form of an embedding, given by a recurrent neural network, which is then provided as a feature to the graph layer (Feng et al., 2019; Wu et al., 2018; Murphy et al., 2021). In other words, they first study the temporal dynamics of each series before convolving on the spatial dimension. The layer is often the long short term memory layer (LSTM), which we describe in Appendix A at page 40. This recurrent layer is widely used for speech recognition (Graves et al., 2013) and has been shown to perform well on stocks (Kim et al., 2019). A slight variation of the order between LSTM and graph layers can be found in Yu et al. (2018), where a graph convolution is performed before and after the temporal layer. These methods are easy to implement using the PyTorch library, since its LSTM layer can easily be combined with PyTorch Geometric layers. They also allow to perform ablation studies to capture the effect of both the LSTM layer and the graph convolution, since both steps are clearly delimited in the code.

Another approach showed in Chen et al. (2018) blends the LSTM architecture and the GCN layer by using message passing in every equation. The details of the implementation, which is an example of a non-message passing layer, are also described in Appendix A. However, the intricacy between the two mechanisms makes it difficult to study their individual effects. Since our study aims to show how graphs can help for portfolio management, it seems more relevant to keep the graph and temporal layers separated to study their individual effects.

3.4 Generating adjacency matrices

In Section 2 we showed different ways to build graphs linking assets together when dealing with stocks. There is a subset of machine learning research which aims at finding structural information in a dataset without any prior knowledge. Indeed, the amount of information available can be overwhelming if we do not know what kind of relationships are truly influential during the message passing phase. When using a GAT model, for instance, we end up finding that a majority of the links have an attention coefficient close to zero, meaning that we could have avoided adding them in the first place. Since for very large graphs the number of edges has an impact on computation time, sparsity in the adjacency matrix might be desirable. Many ideas have flourished around generating the sparsest relationship graph that still holds as much information as possible. In particular, causal discovery (Pearl, 2019) brings several means to compute structural graphs in an agnostic fashion.

The optimization program behind causal discovery often comes back at finding a sparse directed acyclic graph linking all series together. Several techniques were developed to solve this problem. All aim at finding, if the data was generated by a Bayesian network, a resulting graph that is as similar as possible to the true graph. Methods include the LINGAM algorithm (Shimizu et al., 2006), using Independent Component Analysis; the Graphical Normalizing Flow algorithm and the NOTEARS class of optimization problems (Zheng et al., 2018; Romain and d'Aspremont, 2020) along with their recent improvements through penalization (Kyono et al., 2020).

Algorithm 2 The topological sort algorithm for verifying acyclicity

Require: G = (V,E) a graph and the DFS function defined in algorithm (3)
  $sorted = \emptyset$
  $pending = \emptyset$
  $isAcyclic =$ **true**
  for $i \in V$ **do**
    $isAcyclic = isAcyclic$ **and** DFS(i)
  end for
  return $isAcyclic$

Algorithm 3 Depth First Search (DFS) procedure for acyclicity testing

Require: $i \in V$
  if $i \in pending$ **then**
    return false # The DFS algorithm found a cycle in the graph
  end if
  if $i \in sorted$ **then**
    return true
  end if
  $pending = pending \cup \{i\}$
  for $j \in \mathcal{N}(i)$ **do**
    DFS(i)
  end for
  $pending = pending \backslash \{i\}$
  $sorted = sorted \cup \{i\}$
  return true

However, these techniques falter when faced with data generated by a random VAR

Table 1: Similarity metrics between the true matrix and the CASTLE generated matrix after 100 random experiments, compared to the theoretical metrics associated with randomly generated graphs.

Metric	Completeness	Precision	Hamming
Acyclic Causal Model	0.46	0.41	0.15
Random VAR	0.26	0.15	0.22
Baseline	0.125	0.125	0.234

model, which may allow for loops in the resulting graph. To show this fact, we run experiments on synthetic graphs with an edge density similar to the supply chain graph we obtained for the MSCI World data. The synthetic matrices have size 20×20 and 50 distinct edges. The generated adjacency matrices are asymmetric, 100 are completely random and another 100 are generated so that they be acyclic. Acyclicity is obtained by drawing randomly new edges until the graph passes the topological sort acyclity test described in algorithms (2) and (3). In Table 1 we provide several metrics computing how close the CASTLE generated adjacency matrix is compared to the true matrix. We optimize the hyperparameters of the CASTLE penalization on another batch of 100 random VAR graph models. The metrics are the completeness, which computes how many of the true edges are in the generated matrix, the precision, which computes the proportion of edges in the generated matrix actually are in the true matrix, and the hamming distance, which evaluates the edit distance between the two matrices. The baseline is the theoretical mean distance between two completely random matrices with this edge concentration. As we can see, the precision and the Hamming distance of the CASTLE generated matrices on non-acyclic graphs is close to the random theoretical distances. This means that, since we cannot assume the acyclicity of a hypothetical stock price generating graph process, we will not rely on these techniques to add edges to our graphs.

3.5 Static and dynamic graph learning

In our experiments, when dealing with small time intervals (around 5 years) we decide to assume that the graphs did not change. This is true for sector information, but we had to make sure it was realistic for supply chain and correlation graphs. To this end, we compute the Hamming distance between adjacency matrices for the same companies at different times. We find that after five years, less than 10% of the existing supply chain relationships changed. We decide that for experiments on the MSCI World All Returns index, we could assume that the graphs are static in our five-year period. However, when dealing with S&P 500 companies, one must take into account the changes in composition of the index. Thus, our S&P 500 experiments will allow for the dynamic update of adjacency matrices to account for the removals and additions of companies in the index, even if this leads to weaker performances.

4 From graphs to hypergraphs

The graph representation is a very intuitive way to represent relationships between data points. However, when the number of edges increase, some structural information may be lost. In particular, if two nodes are linked by several relationships, the graph will only encode it with one edge. Hypergraphs are a generalization of graphs which provide a more informative structure, at the expense of some useful theoretical properties.

4.1 Definition

A hypergraph is a graph-like structure $G = (V, E)$ where edges are no longer defined as a relationship between two nodes, but instead a realtionship between a whole set of nodes. This means that instead of being a subset of $V \times V$, we now have $E \subset \mathcal{P}(V)$, where $\mathcal{P}(V)$ is the set of all sets of elements of V. Every set $e_i \in E$ is called a hyperedge. While in a graph two nodes can only be linked in one way (either the vertex exists or it does not), in a hypergraph two nodes might belong to several hyperedges together. Hypergraphs are thus a generalisation of graphs, since one can define graphs as being a particular family of hypergraphs where every hyperedge is either a couple or a singleton of nodes. Instead of an adjacency matrix which would link nodes to each other, a hypergraph is better represented by an incidence matrix which links nodes to their hyperedges. Let N be the number of nodes and M the number of hyperedges. The incidence matrix H is defined as $\forall i \in \{1, \dots, N\}, \forall j \in \{1, \dots, M\}$:

$$H_{i,j} = 1 \iff \text{node } i \text{ is in hyperedge } j \tag{13}$$

and 0 otherwise. From this matrix, we can define two diagonal degree matrices. The first is the node degree matrix D, which counts for each node how many hyperedges contain it:

$$D_{i,i} = \sum_{j=1}^{M} H_{i,j} \tag{14}$$

And the second is the hyperedge degree matrix B, which simply gives the cardinal of each hyperedge:

$$B_{j,j} = \sum_{i=1}^{N} H_{i,j} \tag{15}$$

There exist a definition of a directed hypergraph. While in a directed graph every vertex in an edge could either be a parent or a son vertex, in a directed hypergraph every hyperedge is partitioned into a tail and a head subset, denoted by $H(e_i)$ and $T(e_i)$. In this configuration, information can only flow from tail vertices to head vertices. Still, we will focus in this study exclusively on undirected hypergraphs.

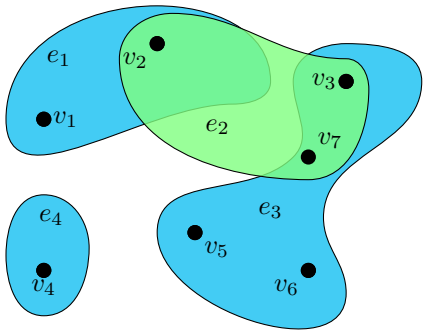
4.2 A generalization with more freedom

When looking at our sector matrix in Figure 2a (page 7), it is obvious that there are better ways to take this information into account than to link every single node to every other in the same sector. Rather than taking a sector as an fully fledged entity, graphs multiply the number of relationships while forgetting the simpler structure. For instance, in a typical regression, one might want to have the sector as an independent factor, which would have its own coefficient, common to every asset. In other words, we could want to treat the assets in the same sector as a cohesive set, instead of as individual nodes. This is where hypergraphs come into play.

Another way to look at this structure is with metro lines. While metro lines have classically been represented as graphs for tasks like pathfinding (Cormen et al., 2009), this completely overlooks any information about lanes, the only relevant factor being whether one can hop from one station to another. For traffic flow prediction, this could come with an issue. A passenger could want to stay in the same lane, even though the graph might indicate a faster route with many lane changes. If each lane is a hyperedge, then the dispersion of information will act more in line with the true representation of the metro's structure.

In Figure 6 we present a hypergraph version of the graph in Figure 3 of page 8. The hyperedges e_i are represented in blue for sector information and in green for supply chain

Figure 6: The toy example of Figure 3 (page 8), in hypergraph form



relationships. Notice how the sector relationship linking v_3, v_7, v_5, and v_6 is now represented by a unique set instead of the six edges of the clique. These six edges can now be seen as one entity in our calculations. This allows both more abstraction, since now sectors exist as a unique object, but also more freedom, since v_3 and v_5 are linked twice, once through their sector relationship and a second time through the supply chain relationship with v_2. In Figure 3, both relationships are represented by a single common edge. This solves the issue of combining several relationship types together. Every relationship is transcribed as a set and does not blend with different relationships. However, some information might be lost in the set structure. In our toy example, v_2 and v_7 are not directly linked, but since they are in a supply chain relationship with v_3, they belong to the same hyperedge. By gaining abstraction, we lost some of the finer details of the structure of the graph.

4.3 Examples of hypergraphs

Just like we provided practical examples of graph theory applied to asset management, we will detail the applications of hypergraph theory.

4.3.1 From graphs to hypergraphs

The hypergraph structure is very suited to sector information. We say that i and j are in the same hyperedge if and only if they are in the same sector. This gives us a partition of the assets along the sectors, which can then be manipulated as an entity in itself. However, for sparser relationships, like supply chain for instance, there is more than one way to build a hypergraph. This comes from the fact that many aggregation functions can be defined on a graph's nodes (Cormen et al., 2009). For example, in a very similar way to the sector hypergraph, one can make every connected component of a graph a hyperedge. However, this might lose the structure of the original graph, and thus forget important information. One example could be metro lines. Given the graph representing a metro, since one can get from every station to any other without leaving the graph, this method would lead to every single metro station being stacked into the same hyperedge. This is obviously an issue, since we would have lost all structural information.

Another way to build hypergraphs is to look at the adherence of every node. That is, for all nodes, we create a hyperedge with them and their direct neighbors. This is less brutal than the previous version and enables us to maintain sparsity in the new hypergraph. This in turn can be a curse, since having many very small hyperedges defies the point of the structure. This would be the case for very sparse graphs, but for highly connected graphs

this technique provides interesting hypergraphs. Indeed, this hyperedge creation strategy is used by Zhu et al. (2016) to transform similarity matrices into hyperedges using thresholding. In our experiments, we choose one of the two strategies empirically. We count the number of hyperedges using the first method and decide based on the number of hyperedges whether to switch to the second one. For instance, if the first method gives us one unique hyperedge, the hypergraph is useless since it lost all structural information. The second method is therefore preferred.

One last way, specific to similarity measures and presented in Shen et al. (2012), is to use frequent itemset mining. These data mining methods, also known as association rule mining, was introduced by Aggarwal and Yu (1998). It is a family of techniques that clusters data points according to how frequently they appear together. The adaptation of this idea to stock markets, introduced in Shen et al. (2012), is to cluster together stocks according to how frequently their movements are synchronised. We set a minimum threshold, then count every time two stocks went the same direction and save this number in a matrix S, such that $S_{i,j} = n \iff$ companies i and j were synchronised at most during n consecutive days. We then create the adjacency matrices:

$$A^n_{i,j} = \mathbb{1}_{S_{i,j} \geq n} \tag{16}$$

from which we generate a sequence of hypergraphs incidence matrices H^n using the first method. The final hypergraph is then the union hypergraph

$$G^{n_{\min}} = \left(V, \bigcup_{n \geq n_{\min}} E^n \right) \tag{17}$$

with E^n the hyperedges defined by the incidence matrix H^n and $n_{\min}$ an arbitrary threshold. The intuition behind this hypergraph is that companies whose changes in profitability are synchronised often will be linked by many hyperedges. This allows us to build partitions based on the number of overlapping hyperedges through hypergraph clustering.

4.3.2 Hypergraph partitioning

Hypergraph clustering is a much-studied subject, spearheaded by its importance in electronics. Indeed, the hypergraph structure enables the mathematical representation of the connectivity of components on a printed circuit. Therefore, clustering algorithms were developed to find the most efficient welding pattern (Shen et al., 2012). Although hypergraph neural networks are very recent, clustering on hypergraphs is a well-studied problem in computer science. Clustering hypergraphs can provide a way to refine highly connected hypergraphs obtained through frequent itemset mining or graph to hypergraph transformation.

Shen et al. (2012) provides a very interesting case study for the refinement of hypergraph information for stock prediction. Their idea is to use clustering to find a partition of the stocks on which they build a signal. This signal is based on information about the history of each asset taken individually, corrected by an average on all assets of a given cluster. Let S be a partition of the assets, let T be a fixed window of time and let P^i_t be the normalised price of asset i at time t. The paper defines two signals:

$$f(S) = \sum_{v \in S} \frac{1}{|v|} \sum_{i \in v} \left(P^i_t - P^i_{t-T} \right) \tag{18}$$

and

$$T'(i) = \sum_{j=1}^{T} \frac{(P_t^i - P_{t-j}^i)}{\sum_{j=1}^{T} j} j \tag{19}$$

And their signal is then simply, for asset $i \in S$:

$$R(i) = 0.7T'(i) + 0.3f(S) \tag{20}$$

In particular the authors predict an upper movement of the stock's value if the signal is positive, and downward if it is negative. The authors build their partitioning using spectral clustering on the frequent itemset hypergraph we described earlier. We implement this signal for the S&P 500 index stocks from 2010 and study its accuracy. The paper promises 80% accuracy over the 2009/2010 period, which we were not able to reproduce on the 2020/2021 period. Figure 7 shows the ratio of correct predictions and false predictions. The precision of the signal hovers around 50% in every period, with the two dents in upward prediction being compensated by a better downwards precision. This is not completely a bad news, since when decomposing the assets by quantiles (Figure 8), we see that the precision is much higher for well-performing stocks than for low-performing stocks. The two dents correspond to periods of uncertainty and match drawdowns in the S&P 500 index. We tried to build a long 10% strategy based on this signal and find great results in the 2021 period, where trend following is very efficient due to the constant upwards trend. It is however underperforming in 2020, where it misses the correction after the drawdown (as shown in Figure 9). Table 2 displays the annualized returns, volatility, Sharpe ratio, maximum drawdown and Calmar ratio of this strategy annually, with the same metrics computed on the S&P 500 index being provided in Table 3.

Table 2: Performance of a long 10% strategy based on the trend following signal of Shen et al. (2012)

year	Perf (in %)	Vol (in %)	SR	MDD (in %)	MDD/Vol
2020	-19.13	39.43	-0.49	39.02	0.99
2021	54.97	18.19	3.02	5.61	0.31

Table 3: Performance of S&P 500 index

year	Perf (in %)	Vol (in %)	SR	MDD (in %)	MDD/Vol
2020	15.59	33.83	0.46	33.92	1
2021	32.26	13.16	2.45	4.23	0.32

4.3.3 From hypergraphs to graphs

In our work, we must be able to generate graphs out of pre-existing hypergraphs. There are two main methods for this transformation: clique expansion and star expansion. Cormen et al. (2009) defines a clique as as a fully connected subgraph in a given graph. In our toy example in Figure 3 of page 8, there are three cliques: (v_1, v_2), (v_2, v_3, v_7) and (v_3, v_5, v_6, v_7). Each of these cliques are hyperedges e_1, e_2 and e_3 respectively. In this case, the inverse transformation is trivial. Each hyperedge will become a clique in the new graph, connecting every vertex of one hyperedge to another. This is the operation known as clique expansion. But it may come with some issues. For instance, if we generate a hypergraph out of the graph in Figure 3 using the connected components method, then the resulting graph will see e_1,

Figure 7: Precision of the predictions based on the trend following signal

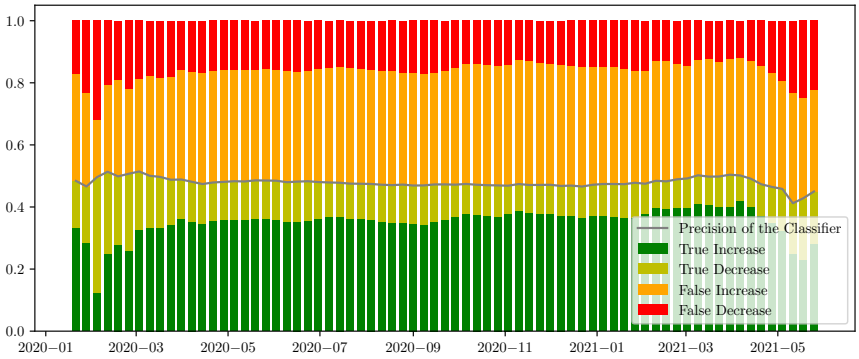


Figure 8: Precision of the trend following signal, with companies segregated by quantile according to their true returns

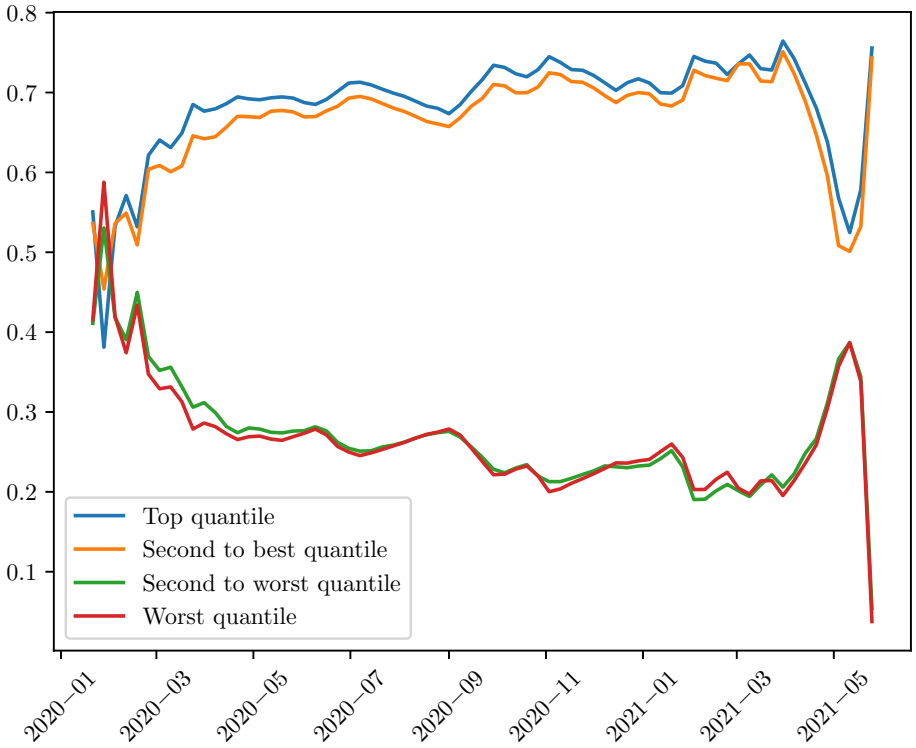


Figure 9: Performance of a long 10% portfolio based on the Trend Following signal, compared to a equally weighted portfolio on the S&P 500 companies

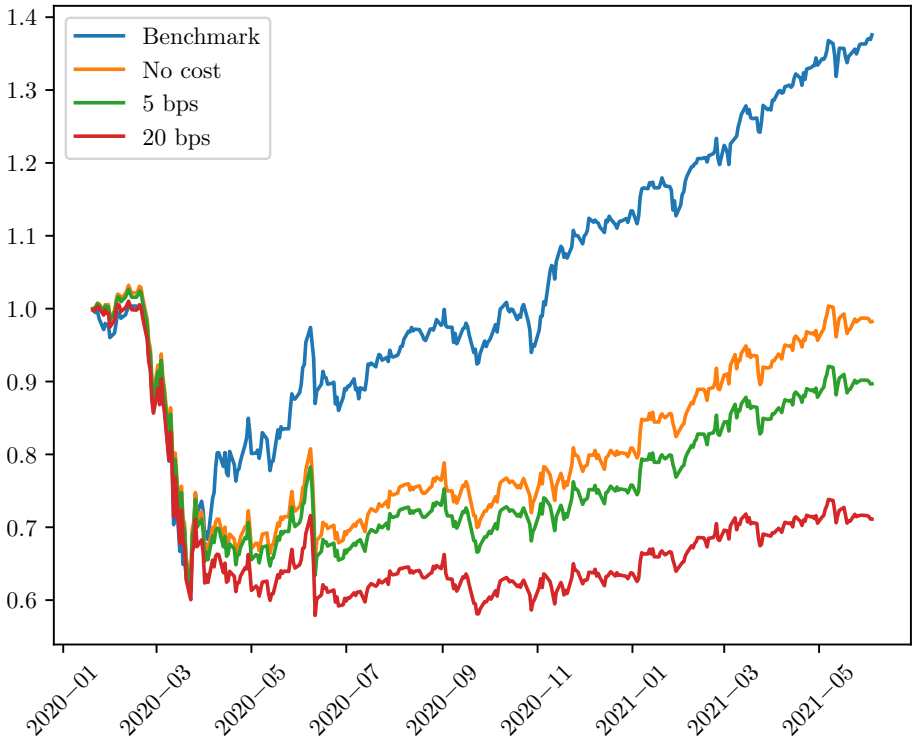
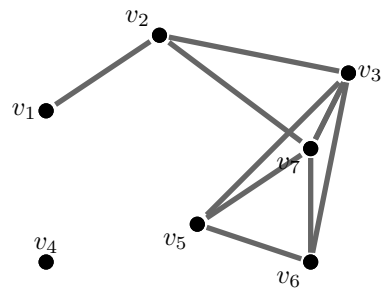
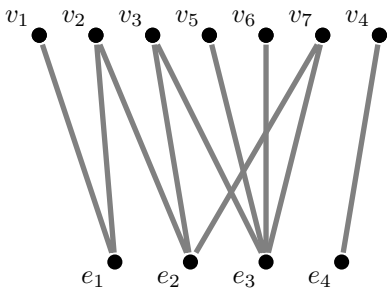


Figure 10: Examples of graphs resulting from the hypergraph of Figure 6 (page 19)



(a) The clique expansion of the toy hypergraph of Figure 6



(b) The star expansion of the toy hypergraph of Figure 6

e_2 and e_3 be one unique hyperedge. The clique expansion will thus add many non-existing relationships between vertices such as v_1 and v_5.

The star expansion allows a lossless transfer from the hyperedges to graphs. The idea is to add to the graph vertices as many new vertices as there are hyperedges. The resulting graph is then the bipartite graph linking every vertex to its edge vertex. An example is provided in Figure 10b, where the star expansion of the hypergraph of Figure 6 is shown. More formally, for $G = (V, E)$ a hypergraph, its star expansion is $G^* = (V \sqcup E, E^*)$ (here $\sqcup$ shows that there is no redundancy between the nodes from V and the nodes of E), where:

$$E^* = \{(i, j), i \text{ is a node belonging to hyperedge } j\} \tag{21}$$

This structure modifies the meaning of nodes, which for a machine learning experiment might be confusing. It also means that in the case of a graph convolution, one needs two iterations for the information of neighboring nodes to reach the receiving node. If the hyperedges are of small cardinal, clique expansion might make more sense. For instance, in Figure 10a, we see that the resulting graph is similar to the original toy graph of Figure 3. In particular, by encoding a graph as a hypergraph of maximum hyperedge cardinal 2, the clique expansion results in the original graph.

4.4 Hypergraph convolution layers

Since graphs and hypergraphs are related, some of the graph's theory remains relevant when building a neural layer. However, there is no single definition of the Laplacian of a hypergraph (Agarwal et al., 2006). Chan et al. (2016) provides a definition that kept most of the graph Laplacian properties and from which Bai et al. (2019) defines a convolutional layer very similar to the GCN layer of Kipf and Welling (2017). A more complete explanation of the hypergraph convolutional layer is given in Appendix C at page 44.

Let Θ be a real diagonal matrix, let $H \in \{0, 1\}^{N \times M}$ the incidence matrix of the hypergraph. Let also D be the node degree matrix and B the hyperedge degree matrix. The expression of the convolution layer of Bai et al. (2019) is then very similar to the GCN layer of Kipf and Welling (2017), but it uses the two degree matrices:

$$Y = \sigma\left(D^{-\frac{1}{2}} H B^{-1} H^\top D^{-\frac{1}{2}} X \Theta\right) \tag{22}$$

We recognise the approximation of the Laplacian multiplied by a diagonal filter matrix. Here $HB^{-1}H^\top$ acts like the adjacency matrix for a graph. D and B are diagonal matrices, so their inverse is computed in $O(N)$ and $O(M)$ respectively. However, there will also be a multiplication of two non diagonal matrices in this expression because of the presence of $H^\top$, making this layer more computationally intensive with a complexity in $O(N^2M)$ instead of $O(N^2)$ for the graph convolution layer. This complexity difference might be attenuated if H is sparse, or if the number of hyperedges is small, i.e. $N \gg M \approx 1$.

A graph attention layer is also defined in Bai et al. (2019), but it supposes that there exists a measure of similarity $f : V, E \mapsto \mathbb{R}$ between a node and a set of nodes (ie between vertices and hyperedges). This similarity function can be an element-wise operation on all the members of the hyperedge, or can be directly computed between the vertex's features and an edge feature. Once such a function is defined, the formula is very similar to the attention mecanism in Equation (10) of page 13:

$$H_{i,j} = \text{softmax}(\sigma(f(\mathbf{W}X_i, \mathbf{W}X_j))) \tag{23}$$

Then we apply Equation (22), with this new incidence matrix instead of the actual incidence matrix of the hypergraph. However, PyTorch Geometric only allows for the second case,

which does not coincide to our use case. This is why we do not test the attention hypergraph layer in our study. Bai et al. (2019) also provides a quick demonstration that the graph convolution layer is a particular example of the hypergraph convolution layer, which we display in Appendix C at page 44.

5 Portfolio construction using graph neural networks

We now have an intuitive way to model inter-assets relationships and mechanisms to propagate the information using neural network layers. In this section, we detail the architecture we have built to test these mechanisms.

5.1 Description of the models

The models we are interested in combine a temporal layer and a graph neural layer. Table 4 details the generic architecture we settle on, with the convolution layer being either a GAT, a GCN or a hypergraph (HGCN) layer. We choose the learning rate empirically after trying several proposals from the literature. Most research papers use Leaky ReLU or tanh activation functions. After disappointing results with the latter, which lead to a disappearing gradient, we consider the LeakyReLU function. Finally, the dropout for every single layer is set at 0.5 to avoid overfitting.

This architecture is the same for every experiment, when performing ablation studies we simply remove layers from this main skeleton. In the remainder of our experiments, the name LSTM designates this architecture without any convolution layer; GCN, GAT and HGCN this architecture with the corresponding convolution layer, but without an LSTM layer, and GCN*, GAT* and HGCN* this architecture with the corresponding convolution layer and the LSTM layer.

The initial weights of each layer is subject to randomness, thus experiments vary from each other depending on the random seed used during training. We use Monte Carlo estimations to showcase the variance of the results depending on the model.

Table 4: Structure of the LSTM + GAT model for predicting returns

LSTM Layer	size=[|features|, 16]
Convolutional Layer	size=[16, 16]
Linear layers	size=[16, 8, 4, 1]
Dropout	0.5
Activation	LeakyReLU
Final activation	Linear or Sigmoid
Optimiser	Adam
Learning Rate	10^{-3}
Batch Size	32

5.2 Features

Our approach to feature selection is somewhat naive. We keep things simple by focusing on the returns, with no added information such as quantitative factors or macro-economic measures about the companies. For every stock, we compute the daily returns, the annualized returns over one and two weeks, as well as one, two, three and six months. We also compute the annualized volatility and a MACD signal comparing one week to one month.

This information comprises our feature vector on every day. Although we are confident that with macroeconomic information in the features, as in Bloomberg Quant Research (2020), better performance can be achieved, in this paper we are mainly interested in the agnostic performance of the models. When using a LSTM layer, we supply two weeks' worth of data in the temporal phase and recover the last hidden vector to feed it to the graph layers.

Regarding the graphs, we take the sector information matrix, the supply chain matrix derived from the FactSet Revere database and the absolute correlation matrix thresholded at 50%. The final adjacency matrix is the union matrix of these three pieces of information, with some added random edges (accounting for less than 1% of the total edges). The sector and supply chain data are updated every day, but due to computation time the correlations are only computed every week, then copied for every day of the week. The hypergraph used is also a union of hypergraphs. The first hypergraph in the union is the sector hypergraph, the second the next neighbor hypergraph derived from the supply chain graph and finally the next neighbor hypergraph derived from the 50% thresholded correlation graph.

5.3 Dataset separation

The datasets only contain assets we allow in our portfolios, which in the case of the MSCI World Index are assets with market capitalization above 100 billion USD (around 100 companies) and for the S&P 500 index the companies part of the index. This choice is made to limit computation time but may have an influence on our model's performance. Indeed, it is possible to add nodes to a graph which pass their information through but on which no backpropagation occurs. These nodes are called transductive nodes, they provide a wider contextualization to the nodes we work on but increase the computation time by adding nodes and edges to the overall graph (Hamilton, 2020). In the supply chain graph, we see that our choice does not have much of an effect on stocks like Apple and Microsoft, who have many relationships with other high capitalization companies, but will rarefy the relationships of stocks such as Walmart, whose relationships are mostly with lower capitalization stocks.

To simulate the way a trading desk operates, we separate our datasets into small chunks of six months each. The first two years of the dataset are fed to the model in a first training run: they act as the historical data a desk would have when opening. In all of our figures, the first two years (2016 and 2017 for the MSCI World Index, 2010 and 2011 for S&P 500 index) show in sample results over this training period. The model then predicts without being updated for the next six months, after which the new data is used for a new training run, which is used to predict the next six months. This is replicated at every sub-dataset of six months until all the dataset is seen. Furthermore, the last 10% of every training set are used as a validation set for an early stopping mechanism. This means that these points are not used during training, but a loss is computed at every backpropagation. If this validation loss does not go below its minimum in a fixed number of backpropagation (in our case 30 iterations), the learning is stopped, thus limiting the chances of overfitting. Lastly, the data is being aggregated in batches of 32 time steps using the Pytorch Geometric batching scheme.

5.4 Backtest metrics

We compute several metrics for each year to give an idea of the portfolio's performance. First, we show the annualized returns and the volatility of the portfolio during the period. The Sharpe ratio is then computed, without risk free rate. We add the maximum drawdown and the Calmar ratio (maximum drawdown divided by volatility) to further examine the

risk taken by the portfolios. All returns are annualized, even in 2021, although we did not look at the entire year. Since we annualized the MSCI and SP500 indexes in 2021 as well, the comparisons in the tables are still relevant, although the returns are a bit inflated.

5.5 Losses for stock movement prediction

In our experiments, we try to forecast the performance of stocks in order to build long only portfolios. Out of the literature three loss functions caught our attention.

5.5.1 Stock return forecasting

The first method is the forecasting of stock returns using mean square error minimization. The loss is between the predicted return $\hat{Y}$ of all N assets and their true new period returns Y is as follows:

$$\mathcal{L}oss(Y, \hat{Y}) := MSE(Y, \hat{Y}) = \frac{1}{N} \sum_{i=1}^{N} \left(Y_i - \hat{Y}_i \right)^2 \tag{24}$$

We then select the highest quantile of predicted returns at each rebalancing date and invest an equal amount of wealth in each asset of the highest quantile.

5.5.2 Stock return ordering

The regression problem can suffer from instabilities or a lack of convergence. Yet, although stock return forecasting is a difficult problem, in our case we just want to classify stocks from least performing to best performing. A set of machine learning architecture called Learning2Rank promises to solve most of the issues by only focusing on the relative ordering of each asset, instead of the true forecast (Feng et al., 2019). The labels are then no longer the true stock return, but an ordering of each asset from lowest to highest. The loss function is made up of two parts. The first is the mean square error function to ensure that an asset's class is not far away from the truth, whereas the second is a regularization that ensures that the relative ordering of each asset is preserved:

$$\mathcal{L}oss(Y, \hat{Y}) = MSE(Y, \hat{Y}) + \phi \sum_{i=1}^{N} \sum_{j=1}^{N} \max \left(0, (\hat{Y}_i - \hat{Y}_j)(Y_i - Y_j) \right) \tag{25}$$

where ϕ is a constant to be defined as a hyperparameter. Similarly to the regression phase, we then take the assets in the highest quantile and invest an equal amount of wealth in them until the next portfolio rebalancing. We do not settle for this solution however, since we encounter a few issues when coding this loss function. Indeed, although the penalization of the MSE loss allows, in theory, for better results in this setup by adding information relevant to quantile building, its expression has many non-differentiable points due to the maximum function. Furthermore, computing the element-wise gradient is cumbersome and increases computation time.

5.5.3 Stock movement classification

Hsu et al. (2021) and Kim et al. (2019) argue that focusing solely on a stock's relative movement increases the stability of the portfolios. Instead of regression, they propose a classification problem which tries to predict whether a stock will increase in value, instead of by how much. As such, the labels Y_i are binary, with 0 meaning that the stock returns are negative and 1 positive. Due to the natural skew of equity returns, the two classes are heavily unbalanced, with many more positive returns than negative returns. To balance out

the classes, instead of looking at the crude returns, we compare the stock returns to the market returns:

$$\begin{aligned} \mathrm{Y}_i &= 0 \text{ if } r_i - r_m \leq 0 \\ \mathrm{Y}_i &= 1 \text{ if } r_i - r_m > 0 \end{aligned} \tag{26}$$

Once the two classes are defined, the loss is the cross entropy between the true and predicted (noted $\hat{Y}_i$) class distributions. Since the values of the labels are either 0 or 1, a sigmoid function is applied after the last layer of the forward pass. The predicted value denotes the confidence of the network in its classification. A value close to one means that the network predicts the stock will be higher than the market with high confidence, close to zero means that the network predicts that the stock will underperform the market with high confidence. Since this is a binary classification problem, the loss is the binary cross entropy (BCE) loss:

$$\mathcal{L}oss(Y, \hat{Y}) = -\frac{1}{N}\sum_{i=1}^{N} Y_i \log(\hat{Y}_i) + (1 - Y_i)\log(1 - \hat{Y}_i) \tag{27}$$

We still take the highest quantile. It is important to notice that here we do not select assets that have the highest returns, but instead assets that will most probably have higher returns than the market. This might lead to lower returns than the other two losses but should improve the precision of the selection. In other words, we expect the assets selected in the long only portfolio to be more systematically better than the market than the regression or ordering schemes.

6 Results

6.1 Experimental setup on the MSCI World index highest capitalizations

We study the added value of the GCN, GAT and HGCN layer through an ablation study on the MSCI World index. We cherry pick stocks on which we have all the required data, and which are of high capitalization (greater than USD 100 billion). Using the graph shown in Figure 4 on page 8, we train two graph-based models and our hypergraph model twice, once with the regression loss (MSE) and once with the classification loss (BCE). The seven models are:

- an LSTM layer without convolution (LSTM);
- a GCN layer without temporal encoding (GCN);
- a GCN layer with a LSTM temporal encoding (GCN*);
- a GAT layer without temporal encoding (GAT);
- a GAT layer with a LSTM temporal encoding (GAT*);
- a HGCN layer without temporal encoding (HGCN);
- a HGCN layer with a LSTM temporal encoding (HGCN*).

We use these models to build long only portfolios with the top 5%, 10% and 20% stocks according to the learned score, which we call Long 5% (around six stocks selected at each rebalancing), Long 10% (eleven stocks) and Long 20% (twenty-two stocks). We compare

each metric to those computed on the MSCI World index, which is our benchmark. 2016 and 2017 are in sample periods, i.e. the metrics are computed on data which is seen during training. From 2018 onwards the metrics are computed out of sample: these are the years to look at to study the predictive power of our models. 2019 and 2021 are stable years with low volatility and high returns, whereas 2018 and 2020 are years with high drawdowns and a high volatility. We rebalance our portfolio daily to study the theoretical impact of each layer, keeping in mind that when accounting for transaction cost this rebalancing frequency will become cumbersome. To build the Long 5% portfolios, we train ten models using a different seed for each, then we compute the mean of their performance. This allows to show the influence of the random initialisation of the layers on the asset selection task and demonstrates that some layers are more stable than others. The results are shown in Tables 6 on page 53. We would have wanted to compute a standard deviation on more samples and for every portfolio, but we are constrained by computation time. This is why Tables 7 and 8 (respectively at pages 54 and 55), representing the Long 10% and Long 20% portfolios respectively, show the results of one neural network trained on a fixed seed.

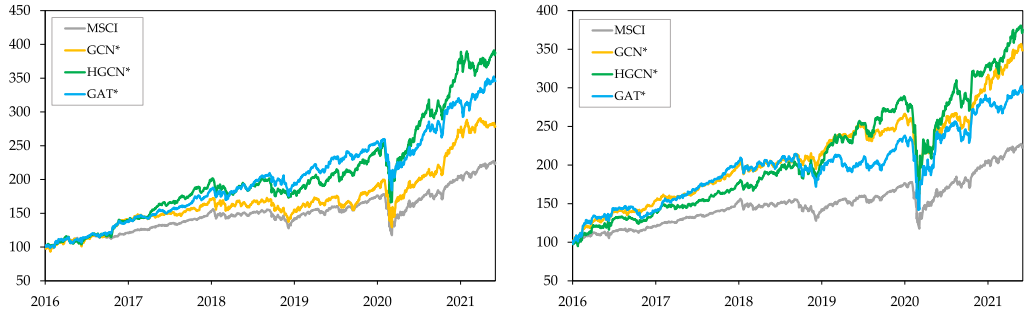
6.2 Discussion on the MSCI World index backtests and ablation study

First of all, it is clear that the classification loss is more stable than the regression loss. In particular, the classification loss performs much better in terms of returns and of Sharpe ratio in 2018, where the regression seems to have overfitted for every model. Surprisingly, the classification loss leads to better in sample returns, although it does not give as much information on returns as the regression loss. This validates the use of classification when using machine learning for portfolio building. In our Long 5% portfolio experiments the GAT and the GCN layers do not seem to yield significant differences during the backtests. This can be explained by the fact that we did not take advantage of the directed nature of GATs, training instead both networks on the same structural undirected graph in Figure 4. What is interesting is that during the low volatility years of 2019 and 2021 the performance of the LSTM drives the performance of every model. Indeed, in these periods, the models with an LSTM layer have better returns than the ones that do not. However, our experiment validates the intuition that graph layers act like filters. In 2020, the graph layers severally reduce the standard deviation of returns and of volatilities of the LSTM layer. They act like a stabilizer in this period, providing fewer but more consistent returns. The hypergraph layers show the most consistent returns in the Monte Carlo experiments of the Long 5% portfolios, especially in 2018.

6.3 Graph ablation study

We study the influence of the three graphs composing our union graph. This enables us to study the relative importance of sector, correlation-based and supply chain information in our models. Indeed, the three graphs have very different topologies, which in turn influence our result. As shown before, the sector matrix has several unconnected cliques. In this graph, no diffusion is allowed between firms of a different sector while some sort of averaging occurs within sectors. The supply chain graph is very sparse compared to the other two: it has only a few connections, but they are more meaningful. Lastly, the correlation graph is highly connected and has only a few unconnected components. According to the theory, this is where GAT layers should thrive. Table 12 at page 59 presents the same experiment as the monte carlo procedure for the Long 5% portfolio on the MSCI World index, this time using the three graphs in isolation. We see that each graph is able to beat the MSCI World index

Figure 11: Cumulative returns of the portfolios on the MSCI World with a starting wealth of 100USD



(a) Using the classification loss

(b) Using the regression loss

comfortably, which proves that the information they hold is relevant. The correlation graph is by far the highest performing, in particular when GATs are used, which confirms the idea that GATs thrive in over connected graphs. However, in 2021, it is the supply chain graph that shows the best performance, despite it lagging behind in the previous years. This might come from the fact that supply chain has become particularly relevant after the Covid-19 crisis, replacing sectors as the most informative structural representation of stocks. Finally, we see that the difference between the GAT* and GCN* models' performances are smaller on the supply chain graph, validating the use of the GCN layer on sparse graphs.

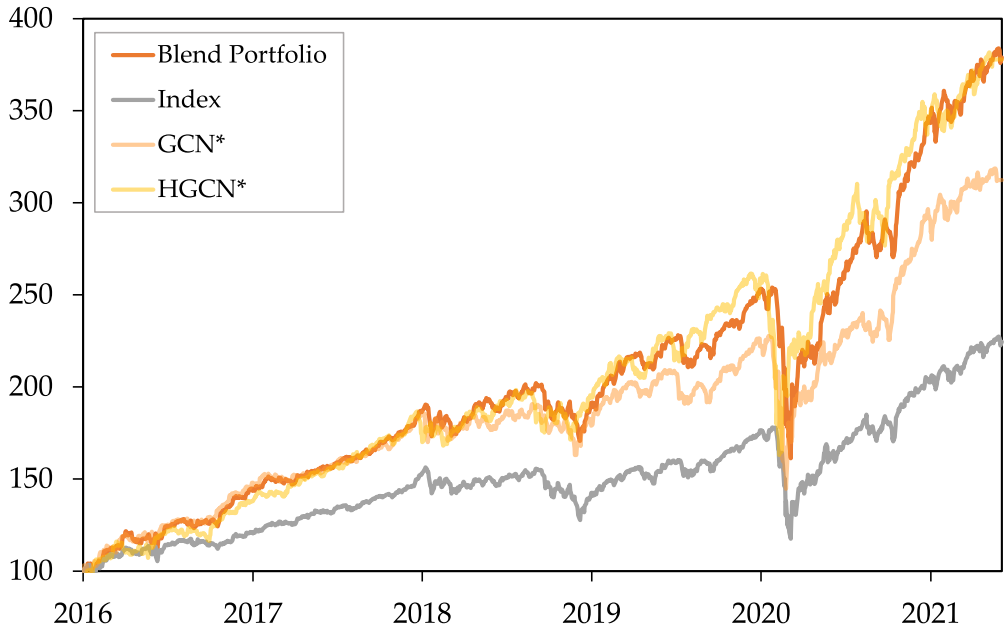
6.4 Blending Strategies

We observe that on long 10% portfolios, some shortcomings of the GCN* model are compensated by the HGCN* and vice versa. In particular, the GCN* model achieves much higher returns than the HGCN* in stable years while the HGCN* holds up to high volatility years such as 2015 or 2020. This is why we combine the two strategies using a volatility-based signal. The idea is to compare, at every time t, the annualized volatility of the S&P500 index over the last three months and the annualized volatility over the last six months and to compute how this difference compares to previous differences. To this end, we compute the percentile of each volatility difference with regards to the distribution of the last three years' worth of differences. This provides a signal value between 0 and 1 which can be used to weight the importance of the two models. Finally, the signal is smoothed using a five-day window exponentially weighted moving average (EMA). The final signal α_t is thus:

$$\alpha_t = \text{EMA}_{5d}\left(\text{quantile}_{3y}(\text{Vol}_t^{3m} - \text{Vol}_t^{6m})\right) \tag{28}$$

which gives an indication of the relative volatility profile of the period compared to the past. We then invest $\alpha_t * W_{t-1}$, with W_t here being the wealth at time t, according to the HGCN* strategy and $(1 - \alpha_t) * W_{t-1}$ according to the GCN* strategy. This means that if the volatility of the last three months is abnormally high, we are more confident in the HGCN* strategy, while we prefer the GCN* strategy if the volatility is abnormally low. The cumulative returns of such portfolios are represented in Figure 12 alongside the cumulative returns of the GCN* and HGCN* based portfolios for both the S&P 500 and the MSCI World universes. We see clearly that for both universes the blend portfolio improves the quality of the predictions by bringing in higher returns than both models.

Figure 12: Cumulative returns of the blend portfolio on the MSCI World index with a starting wealth of 100USD



6.5 Experiments on the S&P 500 Index

The S&P 500 provides a larger universe on which to invest, with around 500 stocks instead of the 100 we used in the MSCI World index. It also allows us to train our models for a longer period, starting in 2010 instead of 2016. To replace supply chain information, we use wikidata relationships to enrich sectoral and correlation information. The wikidata information was found in paper (Feng et al., 2019) as a hypergraph, from which we derive a graph by clique expansion (see Section 4.3.3 at page 21).

This time, we allow the set of stocks on which we trade to change. Since we study a longer timeline of ten years, the S&P 500 composition changes. A Boolean mask is used so that the loss does not take into account the error on nodes that are not in the S&P 500 at time t. To add to this more dynamic approach, we also allow the adjacency matrix to change according to time, with correlation and sectoral information being updated weekly.

6.6 The effect of adding the Amundi alpha score

Up to this point, features are computed based on past returns. We expect external information to improve the performance of our models. To prove this fact, we add to our features the Amundi alpha score of each company. This score is an equally weighted aggregation of several factors depending on information ranging from the momentum, the risk and the estimated growth of the stock. It provides an all-round assessment of the underlying company's situation. As such, we expect the performances to improve when using this information.

To show the influence of this score on our models, we train two GCN with LSTM models on the S&P 500 index using our architecture, on the same number of epochs and the same random seed for both. One is trained with the features of the previous experiment on the

Table 5: Influence of the Amundi alpha score on the performance of the portfolio on S&P 500 built using GCNs and a classification loss (bold shows the best for each strategy).

Year		2016		2017		2018		2019		2020		2021	
Alpha Score		yes	no	yes	no	yes	no	yes	no	yes	no	yes	no
Perf (in %)	SP500	14.47		17.79		7.74		29.83		11.68		54.1	
	long 5	**17.24**	13.97	**31.87**	14.13	**10.77**	-10.46	22.16	**32.21**	**35.82**	8.32	**17.52**	9.79
	long 10	**19.94**	19.36	**25.36**	15.88	**1.06**	-11.91	31.62	**33.10**	**29.78**	11.40	11.54	**18.83**
	long 20	19.82	**20.38**	**21.53**	12.73	**2.35**	-9.91	**33.08**	32.10	**19.92**	11.52	10.34	**22.20**
Vol (in %)	SP500	10.91		5.88		17.06		11.85		39.93		12.3	
	long 5	**15.89**	22.81	**10.15**	10.91	**18.43**	18.72	**15.02**	16.25	**35.05**	43.62	**18.22**	19.30
	long 10	**15.97**	22.48	**10.10**	10.69	18.56	**18.04**	**14.90**	15.74	**36.06**	41.71	18.77	**18.23**
	long 20	**13.81**	19.52	**8.43**	9.56	**16.27**	17.19	**12.99**	15.14	**34.10**	42.59	**15.38**	17.37
SR	SP500	1.33		3.02		-0.45		2.52		0.29		4.40	
	long 5	**1.08**	0.61	**3.14**	1.30	**0.58**	-0.56	1.47	**1.98**	**1.02**	0.19	**0.96**	0.51
	long 10	**1.25**	0.86	**2.51**	1.49	**0.06**	-0.66	**2.12**	2.10	**0.83**	0.27	0.61	**1.03**
	long 20	**1.43**	1.04	**2.55**	1.33	**0.14**	-0.58	**2.55**	2.12	**0.58**	0.27	0.67	**1.28**
MDD (in %)	SP500	4.68		3.01		17.66		7.13		37.88		3.53	
	long 5	**-13.79**	-18.98	**-4.55**	-5.41	**-15.08**	-23.13	-11.07	**-10.51**	**-30.78**	-50.07	**-4.51**	-6.51
	long 10	**-12.34**	-17.51	**-5.02**	-5.78	**-18.70**	-22.97	**-9.92**	-10.68	**-30.38**	-47.06	-7.36	**-6.21**
	long 20	**-8.10**	-13.58	**-2.90**	-5.08	**-16.34**	-22.48	**-7.33**	-10.02	**-32.03**	-47.61	**-5.79**	-5.84]
MDD/Vol	SP500	0.43		0.51		1.03		0.6		0.95		0.29	
	long 5	0.87	**0.83**	**0.45**	0.50	**0.82**	1.24	0.74	**0.65**	**0.88**	1.15	**0.25**	0.34
	long 10	**0.77**	0.78	**0.50**	0.54	**1.01**	1.27	**0.67**	0.68	**0.84**	1.13	0.39	**0.34**
	long 20	**0.59**	0.70	**0.34**	0.53	**1.00**	1.31	**0.56**	0.66	**0.94**	1.12	0.38	**0.34**

MSCI World index, the other is trained with the same features along with the Amundi alpha score. The results of the backtests are given in Table 5, where we can clearly see that the portfolio built using the Amundi alpha score as a feature has superior performances. The new feature reduces the volatility while increasing returns in the majority of periods. It also has a positive effect on the maximum drawdown. 2021 is the only year in which both the long 10% and long 20% have a better Sharpe ratio without the score as a feature, but the positive effect of the score on high volatility periods is obvious.

Other quantitative factors or macroeconomic information can be added as a feature. For instance, Bloomberg Quant Research (2020) uses the market capitalization and the relative importance of a supply chain relationship on profits. However, this information is scarce in our datasets. Still, this experiment with the Amundi alpha score provides an indication that more informative features about the firm behind the stock should substantially improve the performances of the portfolios.

6.7 Discussion on the S&P 500 index backtests

We compare the results of the models on the S&P 500 companies from 2010 to June 2021 to the actual S&P 500 index's performances in Tables 9, 10 and 11 at pages 56 to 58. We see that the classification loss is no longer as stable as before, it leads to some severe under performance in stressed years. Regression brings higher returns, which can be explained by the fact that classification does not really classify stocks according to their predicted performance but rather the probability that they will be even slightly better than the market. This is why classification still leads to less volatility, except on hypergraphs where there are no noticeable differences. The year 2015 is interesting since it shows the two graph layers completely missing the mark, whereas the hypergraph layers are beating the index. Here the classification loss does not help the GCN layer, but improves on the GAT's performance. All in all, the hypergraph layer is the only one consistently outperforming the S&P 500 index in all three portfolios.

6.8 Discussion on the relevance of the classical machine learning measures for market finance

On the one hand several research papers show very high accuracy and precision on the classification problem but have disappointing backtests. On the other hand, our models tend to show disappointing accuracy and precision measures (the classification networks all have accuracies around 50%), yet outperform several classical strategies. This discrepancy is confirmed in Kim et al. (2019). In this study, the authors examine both the accuracy and recall of their classification of stocks, but also compute the annualized returns and Sharpe ratios of their backtests. Although their method has both the best mean accuracy and the best mean Sharpe ratio over their testing periods, in the other models they showcase the accuracy and backtest performance are completely uncorrelated, with the best classifiers performing much worse in the backtest than more crude methods.

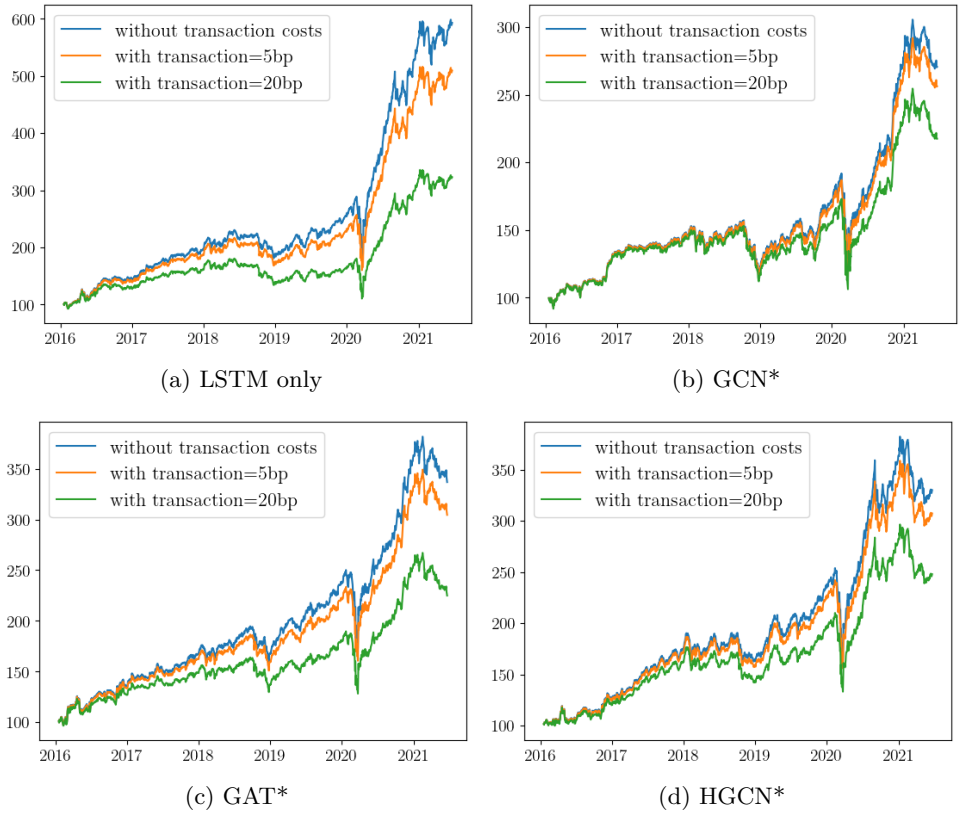
This paradox can be explained in many ways. The first is simply that classification losses do not take into account the amount by which a stock will have increased in value, but rather only the probability that it will increase in value. This means that highly performing but volatile stocks might be given a worse score by the network than low performing but very stable ones. Both could, for instance, be classified as increasing in value, but in the quantile selection only the latter will be added, hence the backtest yielding disappointing returns. Another potential reason, which also adds to the first point, is the fact that neural networks will have a hard time taking into account mean reversion. Indeed, our lookback window for our LSTM cells is two weeks, which might be too short to properly identify what the mean pattern might be. This can lead the network to assume the data to be much more stable than it actually is.

All in all, we are interested in whether the stocks selected are good enough, and less in whether the allocation is optimal. This is why the majority of our testing relies on financial backtests over strategies. This is also why we always compare the results to the related index. This benchmark represents the mean performance of the market, beating it is a hard task. Furthermore, the consistency with which we beat the benchmarks in our experiments, even with Monte Carlo estimation, indicates that there is more than randomness at work, even with an accuracy around 50%.

6.9 Transaction costs

Our experiments used a daily rebalancing of the portfolios to provide a theoretical measure of the relevance of the methods. However, in practice, this approach is unrealistic. Indeed, transaction costs will plague the portfolios' returns if we rebalance that often. Furthermore, this kind of turnover can be detrimental to an asset management firm the size of Amundi, though smaller funds could make it work. This is why we also looked at the performance of our strategies with a weekly rebalancing to study the effect of the turnover on transaction costs. Figure 13 shows the impact of transaction costs on the cumulated returns of a Long 5% portfolio with weekly rebalancing. In blue we represent the portfolio cumulative returns without transaction costs, in orange with 5 basis points of transaction costs and in green 20 basis points of transaction costs. Although LSTMs provide higher returns than the other models over the period, it also comes with a much higher turnover. Adding a GCN layer may decrease the overall returns without transaction costs, yet the two strategies are neck and neck with 20 basis points of transaction costs. Overall, we can see the filtering effect on the high frequency signals in the graph of the GCN and the HGCN layers arising from their

Figure 13: Comparison of the effect of transaction costs on performance on the MSCI world for long 5% strategies with weekly rebalancing



(a) LSTM only

(b) GCN*

(c) GAT*

(d) HGCN*

spectral nature[5], which stabilizes the selection and reduces transaction costs. The LSTM and GAT layer change their predictions widely according to the situation which in turn leads to higher turnover.

[5]See Appendix B on page 41 for the link with Chebyshev expansions and Appendix E on page 50 for details on the frequency profile of the layer.

7 Conclusion

We implement a graph neural architecture for portfolio building on stocks. We use ablation studies to look at the effect of each layer and find that the graph convolution provides improvements on naive recurrent layer forecasting. Results were more promising on the MSCI stocks than on the S&P 500, although adding quantitative information such as the Amundi alpha score improves the backtesting performances. We also use graph to try deterministic algorithms to build signals. Finally, we apply the more recent hypergraph layers in our architecture, where they show promising results.

We observe that spectral theory-based layers like GCN act like filters on graph wide signals, they stabilise the neural networks and reduce the variance generated by the random initialization of weights. Attention mechanisms are more robust on very dense graphs but are susceptible to overfitting. On sparse but informative graphs, such as supply chain graphs, the GCN layer is to be preferred. Furthermore, the added information of the hypergraph structure allows for a more stable forecast during high volatility periods, they can be used in conjunction with graph layers to largely improve portfolios. All in all, graph neural networks allow for more flexible strategies than deterministic algorithms on graphs.

There are many ways to improve on our work. We could provide more quantitative factors as features to the algorithms to try to further improve the performance on the S&P 500. Another potential improvement of our architecture is to add more depth to the neural network, although the smoothing problem may then arise as an issue. Lastly, papers using hypergraphs for financial forecasting recommend a blend of graph and hypergraph layers, where graph layers propagate information inside the hyperedges before the hypergraph layer models inter hyperedge relationships. This should provide the best of both worlds, blending the clique-based propagation of hypergraphs with the finer graph message passing.

References

Agarwal, S., Branson, K., and Belongie, S. J. (2006). Higher order learning with graphs. In Cohen, W. W. and Moore, A. W., editors, *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148 of *ACM International Conference Proceeding Series*, pages 17–24.

Aggarwal, C. C. and Yu, P. S. (1998). A new framework for itemset generation. In Mendelzon, A. O. and Paredaens, J., editors, *Proceedings of the Seventeenth ACM Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*, pages 18–24.

Arroyo, A., Scalzo, B., Stankovic, L., and Mandic, D. P. (2021). Dynamic portfolio cuts: A spectral approach to graph-theoretic diversification. arXiv, 2106.03417.

Bai, S., Zhang, F., and Torr, P. H. S. (2019). Hypergraph convolution and hypergraph attention. In *Pattern Recognition*. arXiv, 1901.08150.

Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D. J., and Kavukcuoglu, K. (2016). Interaction networks for learning about objects, relations and physics. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems (NEURIPS 2016), December 5-10, 2016, Barcelona, Spain*, pages 4502–4510.

Bloomberg Quant Research (2020). Supply chain momentum strategies with graph neural networks (working paper).

Bronstein, M. M., Bruna, J., Cohen, T., and Velickovic, P. (2021). Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. arXiv, 2104.13478.

Calkin, N. and Lopez de Prado, M. (2014). Stochastic flow diagrams. In *Algorithmic Finance*, pages 21–42.

Cellucci, C. J., Albano, A. M., and Rapp, P. E. (2005). Statistical validation of mutual information calculations: Comparison of alternative numerical algorithms. In *Physical Review E*, volume 71, page 066208.

Chan, T. H., Louis, A., Tang, Z. G., and Zhang, C. (2016). Spectral properties of hypergraph laplacian and approximation algorithms. In *Journal of the ACM*. arXiv, 1605.01483.

Cheeger, J. (1969). A lower bound for the smallest eigenvalue of the laplacian. In *Proceedings of the Princeton conference in honor of Professor S. Bochner*, pages 195–199.

Chen, J., Xu, X., Wu, Y., and Zheng, H. (2018). GC-LSTM: graph convolution embedded LSTM for dynamic link prediction. arXiv, 1812.04206.

Chen, Z., Chen, F., Zhang, L., Ji, T., Fu, K., Zhao, L., Chen, F., Wu, L., Aggarwal, C. C., and Lu, C.-T. (2021). Bridging the gap between spatial and spectral domains: A theoretical framework for graph neural networks. arXiv, 2107.10234.

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734.

Cohen, L. and Frazzini, A. (2008). Economic links and predictable returns. In *Journal of Finance*, volume 63(4), pages 1977–2011.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, 3rd Edition*. MIT Press.

Cutura, G., Li, B., Swami, A., and Segarra, S. (2020). Deep demixing: Reconstructing the evolution of epidemics using graph neural networks. arXiv, 2011.09583.

Dees, B. S., Stankovic, L., Constantinides, A. G., and Mandic, D. P. (2019). Portfolio cuts: A graph-theoretic framework to diversification. In *ICASSP 2020*.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NEURIPS 206)*, pages 3837–3845. arXiv, 1606.09375.

Dionisio, A., Menezes, R., and Mendes, D. A. (2004). Mutual information: a measure of dependency for nonlinear time series. In *Physica A: Statistical Mechanics and its Applications*, volume 344, pages 326–329.

Feng, F., He, X., Wang, X., Luo, C., Liu, Y., and Chua, T.-S. (2019). Temporal relational ranking for stock prediction. In *ACM Transactions on Information Systems (TOIS)*, volume 37, page 27.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272.

Graves, A., Mohamed, A., and Hinton, G. E. (2013). Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., and Millman, J., editors, *Proceedings of the 7th Python in Science Conference*, pages 11–15, Pasadena, CA USA.

Hamilton, W. L. (2020). Graph representation learning. In *Synthesis Lectures on Artificial Intelligence and Machine Learning*, volume 14, pages 1–159. Morgan and Claypool.

Hamilton, W. L., Ying, R., and Leskovec, J. (2018). Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NEURIPS 2017)*, pages 1025–1035.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. In *Neural Computation*, volume 9, pages 1735–1780.

Hsu, Y.-L., Tsai, Y.-C., and Li, C.-T. (2021). Fingat: Financial graph attention networks for recommending top-k profitable stocks. arXiv, 2106.10159.

Kannan, R., Vempala, S., and Veta, A. (2000). On clusterings: good, bad and spectral. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 367–377.

Kaya, H. (2015). Eccentricity in asset management. In *Journal of Network Theory in Finance*, volume 1, pages 1–32.

Kim, R., So, C. H., Jeong, M., Lee, S., Kim, J., and Kang, J. (2019). Hats: A hierarchical graph attention network for stock movement prediction. arXiv, 1908.07999.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations (ICLR 2017)*. arXiv, 1609.02907.

Kyono, T., Zhang, Y., and van der Schaar, M. (2020). CASTLE: regularization via auxiliary causal graph discovery. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NEURIPS 2020)*, volume 33. arXiv, 2009.13180.

Levie, R., Monti, F., Bresson, X., and Bronstein, M. M. (2019). Cayleynets: Graph convolutional neural networks with complex rational spectral filters. In *IEEE Trans. Signal Process.*, volume 67, pages 97–109.

Muhammet, B., Guillaume, R., Pierre, H., Benoit, G., Sébastien, A., and Honeine, P. (2020). When Spectral Domain Meets Spatial Domain in Graph Neural Networks. In *Thirty-seventh International Conference on Machine Learning (ICML 2020) - Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, Vienna, Austria.

Murphy, C., Laurence, E., and Allard, A. (2021). Deep learning of contagion dynamics on complex networks. arXiv, 2006.05410.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems (NEURIPS 2019)*, volume 32, pages 8024–8035. Curran Associates, Inc.

Pearl, J. (2019). The new science of cause and effect, with reflections on data science and artificial intelligence. In Baru, C., Huan, J., Khan, L., Hu, X., Ak, R., Tian, Y., Barga, R. S., Zaniolo, C., Lee, K., and Ye, Y. F., editors, *2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, December 9-12, 2019*, page 4.

Romain, M. and d'Aspremont, A. (2020). A bregman method for structure learning on sparse directed acyclic graphs. arXiv, 2011.02764.

Rozemberczki, B., Scherer, P., He, Y., Panagopoulos, G., Astefanoaei, M. S., Kiss, O., Béres, F., Collignon, N., and Sarkar, R. (2021). Pytorch geometric temporal: Spatiotemporal signal processing with neural machine learning models. arXiv, 2104.07788.

Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. In *IEEE Transactions on Acoustics, Speech, and Signal Processing*, volume 26, pages 43–49.

Sehanobish, A., Ravindra, N. G., and van Dijk, D. (2021). Gaining insight into sars-cov-2 infection and COVID-19 severity using self-supervised edge features and graph neural networks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021) February 2-9, 2021*, pages 4864–4873. AAAI Press.

Seo, S. and Liu, Y. (2019). Differentiable physics-informed graph networks. arXiv, 1902.02950.

Shen, Y., Hu, J., Lu, Y., and Wang, X. (2012). Stock trends prediction by hypergraph modeling. In *2012 IEEE International Conference on Computer Science and Automation Engineering (ICSESS 2012)*, pages 104–107.

Shimizu, S., Hoyer, P. O., Hyvärinen, A., and Kerminen, A. J. (2006). A linear non-gaussian acyclic model for causal discovery. In *Journal of Machine Learning Research*, volume 7, pages 2003–2030.

Stankovic, L., Mandic, D. P., Dakovic, M., Brajovic, M., Dees, B. S., and Constantinides, T. (2019). Graph signal processing - part I: graphs, graph spectra, and spectral clustering. arXiv, 1907.03467.

Stokes, J. M., Yang, K., Swanson, K., Jin, W., Cubillos-Ruiz, A., Donghia, N. M., MacNair, C. R., French, S., Carfrae, L. A., Bloom-Ackermann, Z., Tran, V. M., Chiappino-Pepe, A., Badran, A. H., Andrews, I. W., Chory, E. J., Church, G. M., Brown, E. D., Jaakkola, T. S., Barzilay, R., and Collins, J. J. (2020). A deep learning approach to antibiotic discovery. In *Cell*, volume 180, pages 688–702.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018*.

Wu, J. and Birge, J. R. (2014). Supply chain network structure and firm returns. SSRN, 238547.

Wu, L. (2015). Centrality of the supply chain network. SSRN, 2651786.

Wu, T., Chen, F., and Wan, Y. (2018). Graph attention LSTM network: A new model for traffic flow forecasting. In *2018 5th International Conference on Information Science and Control Engineering (ICISCE 2018)*, pages 241–245.

Yu, B., Yin, H., and Zhu, Z. (2018). Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 3634–3640.

Zhang, S., Ding, Z., and Cui, S. (2020). Introducing hypergraph signal processing: Theoretical foundation and practical applications. In *IEEE Internet of Things Journal*, volume 7, pages 639–660.

Zheng, X., Aragam, B., Ravikumar, P., and Xing, E. P. (2018). DAGs with NO TEARS: Continuous Optimization for Structure Learning. In *Advances in Neural Information Processing Systems (NEURIPS 2018)*, volume 31, pages 9492–9503.

Zhu, Z., Peng, Q., and Guan, X. (2016). A time series clustering method based on hypergraph partitioning. In *2016 International Conference on Progress in Informatics and Computing (PIC 2016)*, pages 27–31.

Zimmermann, H., Tietz, C., and Grothmann, R. (2012). Forecasting with recurrent neural networks: 12 tricks. In Montavon, G., Orr, G. B., and Müller, K., editors, *Neural Networks: Tricks of the Trade - Second Edition*, volume 7700 of *Lecture Notes in Computer Science*, pages 687–707.

A The LSTM layer

The LSTM layer was introduced by Hochreiter and Schmidhuber (1997) as a recurrent neural layer more robust to long and complex temporal dependencies. Like all recurrent layer, the LSTM cell keeps information from its precious state in a hidden vector, which will be combined to the new data in order to incorporate temporal information. It is structured into several gates, each operating a transformation of the inputs. The particularity of LSTM units is the presence of a so called forget gate, which gives the unit the ability to forget its previous state for some values of the input. This enables the cell to react to sudden changes in the behavior of the time series by not taking into account information relating to a previous regime. LSTM are widely use for any task dealing with temporal information, from speech recognition (Graves et al., 2013) to time series forecasting, or even complex AI for games like DOTA or Starcraft 2.

The layer is governed by the following equations:

$$\begin{aligned} f_t &= \sigma\left(\mathbf{W}_f x_t + \mathbf{U}_f h_{t-1} + b_f\right) \\ i_t &= \sigma\left(\mathbf{W}_i x_t + \mathbf{U}_i h_{t-1} + b_i\right) \\ o_t &= \sigma\left(\mathbf{W}_o x_t + \mathbf{U}_o h_{t-1} + b_o\right) \\ \tilde{C}_t &= \tanh\left(\mathbf{W}_c x_t + \mathbf{U}_c h_{t-1} + b_c\right) \\ C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\ h_t &= o_t \odot \tanh(C_t) \end{aligned} \tag{29}$$

where $\odot$ is the hadamard product of two vectors. It has three inputs: the new data x_t, the hidden state of the cell at the previous time step h_{t-1} and the cell state at the previous time step C_{t-1}. The cell state acts like a memory which can be forgotten should x_t and h_{t-1} reach certain values. This would make the signal f_t be equal to zero and thus will filter C_{t-1} out in the second to last formula. i_t and o_t, called the input and output gates respectively, act in a similar fashion. The input gate weights the signal $\tilde{C}_t$, which is a perceptron layer on the input x_t and hidden state h_{t-1}, whereas the output gate weights the cell state C_t before it becomes the new hidden state h_t. Figure 14 presents a schematisation of the inner workings of a LSTM cell, where two black lines merging symbolise the concatenation of the vectors.

The hidden and the cell state vectors must be initialised, though some papers recommend Gaussian or Xavier initialisation (Zimmermann et al., 2012), we used the PyTorch default with $h_0 = C_0 = \vec{0}$. The LSTM cell is sometimes criticised for its forgetting mechanism, which can hide long term relationships in temporal data. This is why the simpler GRU unit of Cho et al. (2014) is often found instead. This cell has a similar structure than the LSTM cell, but without the f_t gate. However LSTM cells have a history of being used for financial series forecasting and appear in many graph neural network for finance research papers (Hsu et al., 2021; Kim et al., 2019).

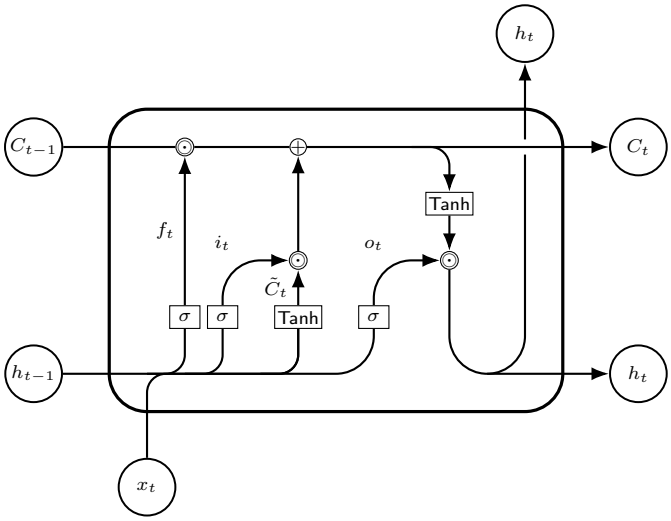
A graph extension of the LSTM layer, proposed by Chen et al. (2018), modifies the formulas in equations 29 by replacing every occurrence of h_{t-1} and c_{t-1} by a graph convolution

of these vectors with the nodes' neighbors:

$$
\begin{aligned}
f_t &= \sigma\left(\mathbf{W}_f x_t + GCN_f(h_{t-1}) + b_f\right) \\
i_t &= \sigma\left(\mathbf{W}_i x_t + GCN_i(h_{t-1}) + b_i\right) \\
o_t &= \sigma\left(\mathbf{W}_o x_t + GCN_o(h_{t-1}) + b_o\right) \\
\tilde{C}_t &= \tanh\left(\mathbf{W}_c x_t + GCN_c(h_{t-1}) + b_c\right) \\
C_t &= f_t \odot GCN_C(C_{t-1}) + i_t \odot \tilde{C}_t \\
h_t &= o_t \odot \tanh(C_t)
\end{aligned}
\tag{30}
$$

where the GCN function denotes a Chebyshev approximation of the graph convolution operator as described in Appendix B. In our study, we chose to remain close to the implementation of Yu et al. (2018); Hsu et al. (2021); Kim et al. (2019) and others and keep the LSTM layer as is presented in equations 29 and the graph convolution separated.

Figure 14: Graphical illustration of the LSTM architecture



B Graph spectral theory and convolution layer

This Appendix details the construction of the GCN layer. This layer is basically a convolution filter in the graph spectral domain. This section is heavily inspired by Kipf and Welling (2017), with added information about the graph spectral theory found in Chen et al. (2021).

B.1 Graph fourier transform

A graph is a non Euclidean representation of relationships. As such, we lack the foundation for applying many algorithms. Spectral transforms like the Fourier transform enable us to project the data in a vector space parametrised by the eigenvalues of the Laplacian. The general idea is to first transform the data into the spectral domain, in which we can apply the known theory, before applying the inverse transform to return to the real data. For instance, k-means or DBSCAN clustering algorithms only make sense in an Euclidean

space, so instead of trying to define distances between nodes, one can simply apply these algorithms in the spectral domain and use the resulting labels in the true domain.

Let $G = (V, E)$ be a graph, where V is the set of vertices of G and E the set of its edges. Let A be the adjacency matrix of G, that is the matrix such that

$$A_{i,j} = \begin{cases} 1, & \text{if } (i,j) \in E \\ 0, & \text{otherwise} \end{cases} \tag{31}$$

where D is the diagonal matrix such that $D_{i,i} = \deg(i)$ and deg is the degree function of a vertex, i.e. the function counting the number of neighbors of each vertex. The Laplacian matrix is then defined as $L = D - A$. A normalised version of the Laplacian is $\tilde{L} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$, which makes every diagonal term equal to 1. The Laplacian matrix can be used as a discrete Laplace-Beltrami operator for solving differential equations on the graph.

The normalised Laplacian of an undirected graph is a symetric semi-definite matrix, meaning that its eigenvectors $(\mu_l)_{l \in \{1,N\}}$ form an orthogonal basis. As such, one can define a graph Fourier transform of any function $f : V \mapsto \mathbb{R}$ and its inverse by projecting on the eigenvectors of a Laplacian:

$$\hat{f}(\lambda_l) = \mathcal{F}(f)(\lambda_l) = \sum_{i \in V} f(i)\mu_l(i) \tag{32}$$

and

$$\mathcal{F}^{-1}(\hat{f})(i) = \sum_{l=0}^{N} \hat{f}(\lambda_l)\mu_l(i) = f(i) \tag{33}$$

where λ_l denotes the lth eigenvalue of the Laplacian matrix. This Fourier transform acts very much like the classical Fourier transform. For instance, Parseval's identity is still verified:

$$||f||_2^2 = ||\hat{f}||_2^2 \tag{34}$$

Furthermore, one can define a convolution operator in the graph domain such that it keeps the properties of the classical convolution operator in the spectral domain:

$$(f * g)(x) = \sum_{l=0}^{N-1} \hat{f}(\lambda_l)\hat{g}(\lambda_l)\mu_l(x) \tag{35}$$

Just like the classical convolution operator on $\mathcal{L}^2(\mathbb{R})$, graph convolution is the multiplication in the graph spectral domain : $\widehat{f * g} = \hat{f}\hat{g}$. Similarly, it verifies every distributivity properties found in the $\mathcal{L}^2(\mathbb{R})$ theory.

B.2 Convolution with a filter and Chebyshev approximation

In theory, once given an analytical expression of the convolution operator, creating a convolution neural layer is immediate. For instance, the convolution layer in Euclidean space is exactly the discreet cross-correlation operator. Kipf and Welling (2017) thus introduce a layer convolving a filter $g_\theta = \text{diag}(\theta), \theta \in \mathbb{R}^N$ and the node features $x \in \mathbb{R}^N$. The filter will simply be the function that we will optimise in the learning phase, similar to the weight matrix of a linear layer. The analytical expression of the convolution gives:

$$(g_\theta * x) = U g_\theta U^\top x \tag{36}$$

where U is the Laplacian's eigenvectors matrix. The problem with the explicit graph convolution operator is that it requires to compute the eigenvector matrix of the Laplacian, which can be very computationally intensive for large or dense graphs. Instead, Kipf and Welling (2017) consider a truncated expansion of the normalised Laplacian $\tilde{L}$ with Chebyshev polynomials. The idea is to express g_θ as a function of the Laplacian eigenvalues, which can be expanded by a truncated Chebyshev expansion. It is interesting to note that this is not the only way to build a convolution filter on the spectral domain, other filters have been proposed using other kernels (Muhammet et al., 2020). To improve the stability of the layer, the authors rescale the normalised Laplacian's eigenvalues using spectral properties of the Chebyshev kernels (proof in Appendix E at page 50):

$$(g_\theta * x) \approx \sum_{k=0}^{K} \theta_k T_k \left(\frac{2}{\lambda_{\max}} \tilde{L} - I_N \right) x \tag{37}$$

where $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x), T_1(x) = x, T_0(x) = 1$, θ_k are the Chebyshev coefficient related to the filter g_θ and K the order of the Chebyshev expansion. The paper argues that empirically, the maximum eigenvalue of the normalised Laplacian is around 2, so $2/\lambda_{\max} = 1$. The interest of this formula is that we can limit ourselves to $K = 1$, and thus avoid any power of L. Indeed, taking the Kth power of the Laplacian matrix means taking into account information from all nodes at path-distance K from the origin (i.e. nodes such that there exists a path with fewer than K nodes between them). Since we only want a convolution layer to look at direct neighbors, just like an image convolution layer only looks at pixels next to those it focuses on, we can set $K = 1$. A generalisation of GCN which does not use this hypothesis, and for which K is seen as a hyperparameter is ChebNet (Defferrard et al., 2016). In the end, they get the following approximation:

$$(g_\theta * x) \approx \theta_0 x + \theta_1 \left(\tilde{L} - I_N \right) x \tag{38}$$

with θ_0 and θ_1 two arbitrary parameters. If we choose $\theta_0 = -\theta_1 = \theta$, which simply restricts the number of parameters in the convolution layer, we finally get the simplified graph convolution operator:

$$(g_\theta * x) \approx \theta \left(I_N + D^{-1/2} A D^{-1/2} \right) x \tag{39}$$

B.3 The graph convolution layer

Kipf and Welling (2017) also look at stability issues with this expression of the convolution operator. The eigenvalues of $\tilde{L}$ are empirically situated between 0 and 2, which may lead to instability when using the Chebyshev approximation multiple times in a row. This is bound to happen since neural layers are meant to be piled onto one another. The authors use the *renormalisation trick*, which replaces $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ by $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, where $\tilde{A} = I_N + A$ and $\tilde{D}$ is its degree matrix. $\tilde{A}$ can be interpreted as the adjacency matrix with added self loops and the matrix $\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$ is known in graph spectral theory as the renormalised symmetric adjacency matrix. This filter comes with lower eigenvalues ($\lambda_{\max} \approx 1$) and has been shown empirically to hold similar effects than the normalised adjacency matrix (Chen et al., 2021). The approximation then becomes, for σ a non linear function:

$$Y = \sigma \left(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} X \Theta \right) \tag{40}$$

where $\Theta = \text{diag}(\theta)$ is a matrix with the parameters of the filter. Though it seems that more matrix multiplication are done than the analytical expression of the convolution operator, D

and Θ are diagonal matrices. This means that the matrix $\tilde{D}^{-\frac{1}{2}}$ can be computed in $O(N)$. The costliest operation is the multiplication with X which is of complexity $O(N^2F)$ where F is the number of features associated to X, while the complexity of an eigenvalue decomposition is that of the multiplication of two square matrices, i.e. $\Theta(N^a)$, $2 < a < 3$, using the Landeau notation for above and below domination. We notice that eigenvalue decomposition algorithms have other steps than the matrix multiplications driving its asymptotic behavior, so even for a small number of vertices, the GCN approximation still decreases computation time.

C Hypergraph spectral theory

Hypergraphs are a generalisation of graphs, but by relaxing the constraint on the edges they lose some properties, among which the straight forward definition of the Laplacian operator.

C.1 Hypergraph Laplacians

Let $G = (V, E)$ be a hypergraph, defined by V the set of vertices and $H \in \mathbb{R}^{N,M}$ the incidence matrix. Let $B \in \mathbb{R}^M$ be the edge degree matrix and $D \in \mathbb{R}^N$ the vertex degree matrix. Several definitions of the Laplacian of such a hypergraph exist in the literature, though they cannot always be expressed as a simple function of one-another like the graph Laplacian (Agarwal et al., 2006).

Most hypergraph Laplacians use a transformation from hypergraphs to graphs to build the associated graph Laplacian. This is the idea behind the clique expansion Laplacian, the star expansion Laplacian and Rodriguez's Laplacian. The clique expansion Laplacian is simply the Laplacian matrix of the clique expansion graph of the hypergraph. Although a Laplacian is well defined on this graph, it will describe a slightly different propagation than the one truly happening in the hypergraph, since the transformation loses the information about the redundancy of relationships in the hypergraphs. Indeed, the Laplacian will only propagate once over nodes that might be linked through several hyperedges.

This issue is solved in the star expansion Laplacian, since the star expansion graph keeps intact every structural information of the hypergraph. But the Laplacian matrix of this new graph still does not match the propagation in the actual hypergraph. Since the star expansion graph links true nodes to hyperedge encoding nodes, one needs two propagation steps for the information of a node to reach its neighbors in the hypergraph. This means that the actual propagation in the hypergraph is a two-step process, with first every true node propagating to the hyperedge nodes, then the hyperedge nodes redistributing the resulting encoding to its true nodes. This is not the propagation defined through the Laplacian matrix.

Zhou's normalised Laplacian is the one that interests us for the properties of its eigenvectors, which are similar to those of the graph Laplacian. This Laplacian is derived from an operator first used for regularisations on vertex functions (Agarwal et al., 2006). It takes inspiration from a definition of the Laplacian matrix for graphs as being the unique matrix L such that:

$$\langle f, Lf \rangle = \frac{1}{2} \sum_{(i,j) \in V} A_{i,j} \left(f(i) - f(j) \right)^2 \tag{41}$$

where f is a function over the nodes, $\langle \cdot, \cdot \rangle$ the inner product between two functions and A is the adjacency matrix. This property of the Laplacian matrix comes from the fact that it is related to diffusion processes in the graph[6]. The $\frac{1}{2}$ term controls the redundancy of directed

[6] See Appendix D at page 47 for the different definitions of the graph Laplacian matrix.

edges in an undirected graph. Chan et al. (2016) adapted this penalisation for hypergraphs and found a matrix L such that:

$$\langle f, Lf \rangle = \frac{1}{2} \sum_{e \in E} \frac{1}{|e|} \sum_{(i,j) \in V} \left(\frac{f(i)}{\sqrt{d(i)}} - \frac{f(j)}{\sqrt{d(j)}} \right)^2 \tag{42}$$

This equation is slightly different than its graph equivalent. The division by two remains to account for the undirected nature of the hypergraph, but another redundancy appears with the set structure. Since two vertices can be linked through several hyperedges, they need, in order to control this new redundancy, to divide each $f(i)$ by $\sqrt{d(i)}$, which is the square root of the number of hyperedges i belongs to. The resulting matrix L is then:

$$L = I_N - D^{-\frac{1}{2}} H B^{-1} H^\top D^{-\frac{1}{2}} \tag{43}$$

This matrix is also related to diffusion in the hypergraph, Chan et al. (2016) uses this fact to demonstrate inequalities resembling the Cheeger equations on graphs. In particular, the second eigenvalue is also related to an adapted conductance measure on hypergraphs. However, the other eigenvalues are less informative of the structure. In particular, in order to define a hypergraph Fourier transform another representation is required, which uses tensors (Zhang et al., 2020). Contrary to graph spectral theory, which is already firmly in the state of the art, such hypergraph spectral theory is very recent.

C.2 Links with graphs

Let $G = (V, E)$ be a graph, let A and D be its binary adjacency matrix and degree matrix respectively. We can express the adjacency matrix as an incidence matrix by given each $M = |E|$ edges a unique number from 1 to M. We thus have:

$$A_{ij} = 1 \iff \exists! k \in \{1, \dots, M\}, (H_{ik} = 1) \text{ and } (H_{jk} = 1) \tag{44}$$

an such a k is different for each pair (i, j). This incidence matrix can define a hypergraph whose node set is V and whose incidence matrix is H.

This hypergraph formalism of graphs has two interesting properties. The first is that the degree matrix of the related hypergraph is exactly the degree matrix of the graph.

Proof. Let D' be the degree matrix of the hypergraph related to incidence matrix H. By definition, both D and D' are diagonal matrices. Let us then show that their diagonal terms are the same. Let $i \in \{1, N\}$, then:

$$D'_{ii} = \sum_{k=1}^{M} H_{ik} \tag{45}$$

We know that $H_{ik} = 1$ if, and only if, there exist a node j such that $(i, j) \in E$, such that for any such j there exists only one e_k. Thus counting the hyperedges e_k is the same as counting the neighbors j. This means that:

$$\begin{aligned} D'_{ii} &= \sum_{j=1}^{N} A_{ij} \\ &= D_{ii} \end{aligned} \tag{46}$$

□

Another property is that $HH^\top = A + D$. A simple intuition is to see the matrix H as an operator from node to hyperedges. H defines a transformation from node space to hyperedge space, and $H^\top$ a transformation from edge space to node space. Then $HH^\top$ links every node to every other node connected by a hyperedge, excluding itself. In our special case, this operator links the nodes to a single neighborhood without redundancy, which is exactly what the operator defined by $A + D$ does on graphs. Here is a more rigorous proof of this fact.

Proof. Since D is a diagonal matrix and A has no diagonal terms, we can show this property by separating two cases. This property is obvious for diagonal terms since, $\forall i \in \{1, \dots, N\}$:

$$\begin{aligned} HH^\top_{ii} &= \sum_{k=1}^{M} H_{ik} H^\top_{ki} \\ &= \sum_{k=1}^{M} H^2_{ik} \end{aligned} \tag{47}$$

Since H is a binary matrix:

$$\begin{aligned} HH^\top_{ii} &= \sum_{k=1}^{M} H_{ik} \\ &= D'_{ii} = D_{ii} \end{aligned} \tag{48}$$

As shown above. Now remains the case of non diagonal terms. Let $i, j \in \{1, \dots, N\}, i \neq j$:

$$\begin{aligned} HH^\top_{ij} &= \sum_{k=1}^{M} H_{ik} H_{jk} \\ &= \sum_{k=1}^{M} \mathbb{1}_{i \in e_k \text{ and } j \in e_k} \end{aligned} \tag{49}$$

Since such a hyperedge e_k is unique and exists if, and only if, $A_{ij} = 1$, this sum is either 1 if $(i, j) \in E$ or else 0, thus:

$$HH^\top_{ij} = A_{ij} \tag{50}$$

which ends the demonstration. □

C.3 Hypergraph convolution layers

Since the resulting hypergraph Laplacian is very similar to the graph Laplacian, Bai et al. (2019) simply adapts the layer of Kipf and Welling (2017) to provide the following formula:

$$Y = \sigma\left(D^{-\frac{1}{2}} H B^{-1} H^\top D^{-\frac{1}{2}} X \Theta\right) \tag{51}$$

Contrary to the graph convolution layer, this layer has no spectral justification. According to Bai et al. (2019), the motivation behind this form is that the GCN layer can be reinterpreted as a particular case of the Hypergraph layer, keeping with the idea that hypergraphs are supposed to be a generalisation of graphs. The demonstration of this fact is as follows. A graph can be interpreted as a hypergraph where any hyperedge's cardinal is 2. Hence, the

hyperedge degree matrix is simply $B = 2I_M$, with I_M the identity matrix of size M. Then, the layer becomes:

$$Y = \sigma\left(\frac{1}{2}D^{-\frac{1}{2}}HH^\top D^{-\frac{1}{2}}X\Theta\right) \tag{52}$$

We demonstrated in the previous section that $HH^\top = A+D$. By expanding this expression, we get:

$$\begin{aligned}\frac{1}{2}D^{-\frac{1}{2}}\left(A+D\right)D^{-\frac{1}{2}}X\Theta =& \frac{1}{2}\left(D^{-1}D + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}\right)X\Theta \\ =& \frac{1}{2}\left(I_N + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}\right)X\Theta\end{aligned} \tag{53}$$

This is the exactly the same form as Equation (39) of page 43. Indeed, the factor $1/2$ can be hidden inside the diagonal filter parameter matrix Θ. The renormalisation trick has to be applied to recover Kipf and Welling (2017)'s exact layer, therefore the stability issues will be present in this layer. However, no equivalent to the renormalisation trick exists for hypergraphs, hence the compromise.

D Properties of the Laplacian matrix

Clustering problems are fundamental to graph theory. Here we provide more details on clustering techniques. In particular, the use of the Laplacian and its eigenvalues in clustering provides a further justification of the GCN layer.

D.1 Graph calculus

In this paper we defined the graph Laplacian as a algebraic object from which we used the eigenvalue decomposition. We stressed the role of this operator in diffusion processes, arguing it was the graph equivalent of the Laplace-Beltrami operator on grids. Here we show that the graph Laplacian is in fact a natural operator in the field of graph calculus, thus reinforcing this link between the algebraic matrix and diffusion processes.

Let us begin by defining the graph gradient. Let $G = (V, E)$ be a weighted graph with adjacency matrix A. The gradient of graph G is a discrete operator $\nabla : \mathcal{L}^2(V) \to \mathcal{L}^2(E)$:

$$(\nabla f)_{i,j} = (f_j - f_i)\mathbb{1}_{(i,j)\in E} \tag{54}$$

where $f : V \to \mathbb{R}$ is a function over the vertices, or equivalently a vector of $\mathbb{R}^N$. Similarly, the divergent of graph G is an operator $\mathrm{div} : \mathcal{L}^2(E) \to \mathcal{L}^2(V)$:

$$(\mathrm{div}F)_i = \sum_{j\in\mathcal{N}(i)} A_{i,j}F_{i,j} \tag{55}$$

where $F : E \to \mathbb{R}$ is a function over the edges, or equivalently a square matrix of $\mathbb{R}^{N\times N}$. The Laplacian matrix on graphs can then be defined equivalently as:

$$\begin{aligned} & L = \Delta = -\mathrm{div}\nabla \\ \iff & L_i = \sum_{j\in\mathcal{N}(i)} A_{i,j}(f_i - f_j) \\ \iff & L = D - A \end{aligned} \tag{56}$$

Finally, we can define a curl operator on 3-cliques, i.e. cliques of three vertices, also known as the triangles of the graph. Let $\mathcal{T}$ be the set of triangles in the graph G. Then the curl of the graph is the operator:

$$(\text{curl}F)_{i,j,k} = F_{i,j} + F_{j,k} + F_{k,i} \tag{57}$$

and

$$(\text{curl}^*\mathcal{F})_{i,j} = \sum_{k,\ (i,j,k)\in\mathcal{T}} \frac{w_{ijk}}{A_{i,j}} \mathcal{F}(i,j,k) \tag{58}$$

where $\mathcal{F} \in \mathcal{L}^2(\mathcal{T})$ is a function over the triangles and w_{ijk} a weight coefficient defined for each triangle. These operators allow for the modelisation of complex physical systems on graphs, leading the way for differential equation solving on graphs. Seo and Liu (2019), for instance, use these operators to train neural networks on electro-magnetic differential equations. They also predict weather variation in a region with a mountain range and other factors to show how graph neural networks can improve the prediction of the forecasts.

D.2 On the Laplacian eigenvalues

The Laplacian matrix is related to random walks in the Laplacian. Indeed, the matrix $D^{-1}L$, also called the random walk normalised Laplacian, is exactly equal to $I_N - P$ with P the transition matrix of a random walk in the graph. Hence, the eigenvalues of the Laplacian give an indication of the topology of graph walks. For instance, if 0 is an eigenvalue of multiplicity more than 1, that means that there is a section of the graph that is unattainable from the rest. The number of isolated subgraphs, also called connex components, is given by the multiplicity of the eigenvalue 0.

The lowest eigenvalues of the Laplacian describes a bottleneck in graph diffusion speed. We saw that if the eigenvalue 0 appears more than once, then a random walk starting from any given node will never reach some parts of the graph. If zero is an eigenvalue of multiplicity one, then a diffusion process starting from any node will eventually have an impact on every node, but the speed at which the entirety of the nodes will be affected depends on another eigenvalue. Specifically, this speed is controlled by a measure called the conductance of the graph, which in turn is controlled through Cheeger's inequality by the second smallest eigenvalue of the Laplacian. The second smallest eigenvalue is fundamental in describing the topology of a fully connected graph, it links the conductance with clustering through the minimum cut problem.

D.3 The minimum cut problem

The graph Laplacian gives information on the sparsest cut of the graph, and can thus be used for clustering. Let us define the minimum cut problem. Let $G = (V, E)$ be a undirected weighted graph, let A be its adjacency matrix. Let $V_1, V_2 \subset V, V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$ a partition of the vertices of graph G. Let $C = \{(i,j) \in V, i \in V_1 \text{ and } j \in V_2\}$ be the cut, i.e. the set of edges that, if removed, would separate the graph G in exactly the subgraphs defined by V_1 and V_2. The cut is also sometimes called the frontier of sets V_1 and V_2, noted δV_1 or δV_2. We define the cost of the cut as follows:

$$W(C) = \left(\frac{1}{|V_1|} + \frac{1}{|V_2|}\right) \sum_{(i,j)\in C} A_{i,j} \tag{59}$$

The minimum cut problem consists in solving the discrete minimisation problem:

$$\min_C W(C) \tag{60}$$

This combinatorial problem can be rephrased using the Laplacian matrix. Let $x(C)$ be the vector such that:

$$x_i = \begin{cases} \frac{1}{|V_1|}, & \text{if } i \in V_1 \\ \frac{-1}{|V_2|}, & \text{if } i \in V_2 \end{cases} \tag{61}$$

Then the minimum cut program is equivalent to solving the problem (Cormen et al., 2009):

$$\begin{aligned} &\min_x x^\top L x \\ &\text{s.t. } x^\top x = 1 \end{aligned} \tag{62}$$

The solution to this program is exactly the eigenvector relative to the second smallest eigenvalue of the Laplacian. Indeed, the smallest eigenvalue is 0 since $L\mathbf{1} = \mathbf{0}$. This vector allocation $\mathbf{1}$ corresponds to a cut where V_2 is empty, which would make the minimisation undefined due to equation 61.

D.4 Link with the graph conductance

The conductance of a graph, described in equation 64 is a measure controlling the weight of the minimal cut as well as the convergence speed of a random walk towards the stationary distribution. Indeed, the conductance of a subgraph S can be interpreted as the probability that a random hop will escape the subgraph having started in it. As such, it can also give a measure of the quality of a clustering, since a it should be difficult for a random walk to get from a cluster to another(Kannan et al., 2000). The graph overall conductance ϕ_G being the minimum conductance over every subgraph, it gives an idea of the difficulty a random walk will have to reach every node at least once.

Let S be a subset of V. Let ∂S be the frontier between S and $\tilde{S} = V \backslash S$. The frontier is the cut inducing S as one of the two subgraphs. Since a cut is a set of edges, we can define $|\partial S|$ the cardinal of the frontier. Let finally $\text{vol}(S) = \sum_{i \in S} D_{i,i}$ be the volume of subgraph S, *i.e.* the sum of the degrees of every vertex in S. The conductance of a subgraph S is defined as:

$$\phi_S = \frac{|\partial S|}{\min\{\text{vol}(S), \text{vol}(\bar{S})\}} \tag{63}$$

and the overall conductance of the graph ϕ_G is defined as the maximum of the conductance of its subgraphs:

$$\phi_G = \min_{S \subset V} \phi_S \tag{64}$$

This graph theory measure is related to the eigenvalues of the Laplacian. Indeed, the Cheeger inequality (Cheeger, 1969):

$$\frac{\lambda_2}{2} \leq \phi_G \leq \sqrt{2\lambda_2} \tag{65}$$

constrains the conductance of a graph according to the second smallest eigenvalue of the Laplacian matrix. Thus, the sparsest cut of a graph can be approached by a fully spectral approach.

Since the conductance of a cut is a measure of the quality of its clustering, and that the minimum conductance is controlled by the second smallest eigenvalue of the Laplacian, which happens to solve the minimal cut problem, one can compare the conductance of any cut found through clustering algorithms and verify that it follows Cheeger's inequalities. If the cut does not fall within the domain defined using the eigenvalues of the Laplacian, then a better clustering exists. This demonstrates how the eigenvalues of the graph Laplacian

provide structural information about the graph. Furthermore, this relationship between cuts, clusters and eigenvalues can be generalised for every eigenvalue of a Laplacian. This validates spectral clustering, which is simply the use of classical clustering algorithms such as k-means or DBSCAN on the spectral domain, i.e. on the eigenvectors of the Laplacian Stankovic et al. (2019).

E Spectral analysis of Chebyshev graph filters

This section is heavily inspired by Muhammet et al. (2020). We provide a spectral analysis of Chebyshev spectral filters like the GCN layer, which justifies the choices in the construction of Appendix B.

Let G be undirected graph (ie A the adjacency matrix is symetric). Let $L = D - A$ be its Laplacian matrix. Let also $\boldsymbol{\lambda}$ be the vector of its eigenvalues and U the matrix of its eigenvectors.

E.1 Frequency profile of a graph convolution kernel

Muhammet et al. (2020) studies the frequency profile of the spectral convolution family of graph neural layers. The start from a general expression of spatial convolution layers:

$$Y = \sigma\left(\sum_{s=1}^{K} C_s X \Theta\right) \tag{66}$$

where Y is the convolved graph, X the feature matrix and Θ a learnable weight matrix. C_s is a convolution kernel approximating a message passing function. Spectral layers, they prove, is a special case of this spatial convolution family where the kernel C_s can be written as follows:

$$C_s = U \operatorname{diag}\left(F_s(\boldsymbol{\lambda})\right) U^\top \tag{67}$$

where F_s is a function providing the frequency profile vector of the kernel at order s. Since the Laplacian is symetric semi-definite, U is orthogonal and $U^{-1} = U^\top$, we can then isolate the frequency profile and get:

$$F_s(\boldsymbol{\lambda}) = \operatorname{diag}^{-1}\left(U^\top C_s U\right) \tag{68}$$

Since GCN layers use a Chebyshev kernel in its approximation, we can compute its exact frequency profile.

E.2 Frequency profile of Chebyshev expansions

The first Chebyshev convolution filter $C_0 = I_N$ has frequency profile $F_0(\boldsymbol{\lambda}) = \mathbf{1}$, with $\mathbf{1}$ the vector of 1s.

Proof. The identity kernel defines an all-pass filter. Using the formula given in the previous subsection, we have that:

$$\begin{aligned} F_0(\boldsymbol{\lambda}) &= \operatorname{diag}^{-1}\left(U^\top I_N U\right) \\ &= \operatorname{diag}^{-1}\left(I_N\right) \\ &= \mathbf{1} \end{aligned} \tag{69}$$

□

The second Chebyshev convolution filter is $C_1 = 2L/\lambda_{\max} - I_N$ and its frequency profile is $F_1(\boldsymbol{\lambda}) = \frac{2\boldsymbol{\lambda}}{\lambda_{\max}} - \mathbf{1}$.

Proof. Indeed, by definition of C_1 the second Chebyshev kernel:

$$\begin{aligned} F_1 &= \text{diag}^{-1}\left(U^\top C_1 U\right) \\ &= \text{diag}^{-1}\left(U^\top \left(\frac{2}{\lambda_{\max}} L - I_N\right) U\right) \\ &= \text{diag}^{-1}\left(\frac{2}{\lambda_{\max}} U^\top L U - U^\top I_N U\right) \\ &= \text{diag}^{-1}\left(\frac{2}{\lambda_{\max}} U^\top L U - I_N\right) \end{aligned} \tag{70}$$

Furthermore, since U is the eigenvector matrix of L, we have $LU = U\text{diag}(\boldsymbol{\lambda})$. Thus:

$$\begin{aligned} F_1 &= \text{diag}^{-1}\left(\frac{2}{\lambda_{\max}} U^\top U \text{diag}(\boldsymbol{\lambda}) - I_N\right) \\ &= \text{diag}^{-1}\left(\frac{2}{\lambda_{\max}} \text{diag}(\boldsymbol{\lambda}) - I_N\right) \\ &= \frac{2\boldsymbol{\lambda}}{\lambda_{\max}} - \mathbf{1} \end{aligned} \tag{71}$$

□

Here we obtained the frequency profiles of the two Chebyshev kernels used in the approximation leading to the GCN layer. In particular, this provides the frequency profile of the layer before the renormalisation trick. While this trick changes the profile, the HGCN hypergraph layer is inspired by the non-normalised version of the GCN. We also provide the general expression of any Chebyshev kernels to illustrate the generalisation of the GCN layer known as ChebNet, where the layer is based on higher order Chebyshev expansions (Defferrard et al., 2016). The kth frequency profiles are defined by:

$$F_k(\boldsymbol{\lambda}) = 2F_1(\boldsymbol{\lambda})F_{k-1}(\boldsymbol{\lambda}) - F_{k-2}(\boldsymbol{\lambda}) \tag{72}$$

Proof. By definition of the Chebyshev polynomials, we know that, for $k > 2$

$$C_k = 2C_1 C_{k-1} - C_{k-2} \tag{73}$$

We once again apply the general formula giving F_k according to C_k:

$$\begin{aligned} F_k &= U^\top \left(2C_1 C_{k-1} - C_{k-2}\right) U \\ &= 2U^\top C_1 C_{k-1} U - U^\top C_{k-2} U \end{aligned} \tag{74}$$

Since $UU^\top = I_N$, Muhammet et al. (2020) inserts this matrix into the equation:

$$F_k = 2(U^\top C_1 U)(U^\top C_{k-1} U) - U \top C_{k-2} U \tag{75}$$

And finally, since $F_k = U^\top C_k U$ for all k:

$$F_k(\boldsymbol{\lambda}) = 2F_1(\boldsymbol{\lambda})F_{k-1}(\boldsymbol{\lambda}) - F_{k-2}(\boldsymbol{\lambda}) \tag{76}$$

□

The Chebyshhev expansion is only one of the possible filters that can be used to build neural layers. Muhammet et al. (2020) provides similar analysis on Cayley expansion neural layers, first introduced in Levie et al. (2019).

E.3 The GCN layer's frequency profile

Since the renormalisation trick changes the frequency profile of the GCN layer, Muhammet et al. (2020) provides a different proof, which assumes that all nodes have the same degree $d \in \mathbb{N}$. Let $\tilde{A} = A + I_N$ the adjacency matrix of the graph with self loops and let $\tilde{D} = dI_N$ be its degree matrix. By definition of the GCN layer:

$$\begin{aligned} C_{\text{GCN}} &= \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2} \\ &= (D + I_N)^{-1/2}(A + I_N)(D + I_N)^{-1/2} \\ &= ((d+1)I_N)^{-1/2}\,(A + I_N)\,((d+1)I_N)^{-1/2} \\ &= \frac{A + I_N}{d+1} \end{aligned} \tag{77}$$

Which we can use in the frequency profile formula. We then have:

$$F_{\text{GCN}} = \frac{1}{d+1}\left(U^\top A U + I_N\right) \tag{78}$$

Then, since U is the eigenvector matrix of L the normalised Laplacian:

$$\left(I_N - D^{-1/2}AD^{-1/2}\right)U = U\text{diag}(\boldsymbol{\lambda}) \tag{79}$$

Which can be transformed, since $D^{-1/2}AD^{-1/2} = A/d$, into:

$$AU = dU - dU\text{diag}(\boldsymbol{\lambda}) \tag{80}$$

There remains to replace AU by this expression to get the following expression:

$$\begin{aligned} F_{\text{GCN}} &= \frac{1}{d+1}U^\top\left(dU - dU\text{diag}(\boldsymbol{\lambda})\right) + \frac{1}{d+1}I_N \\ &= \frac{(d+1)I_N - \text{diag}(\boldsymbol{\lambda})d}{d+1} \\ &= I_N - \frac{d}{d+1}\text{diag}(\boldsymbol{\lambda}) \end{aligned} \tag{81}$$

We have the frequency profile of the GCN layer, both before and after the renormalisation trick. Since the GAT layer relies on a more complex mechanism than the GCN layer, an analytical spectral profile is impossible to obtain. This is why Muhammet et al. (2020) only provides an empirical profile computed through Monte Carlo simulations.

F Tables

Table 6: Annual backtest of the long 5% strategies on the MSCI World index

		Perf (in %)		Vol (in %)		SR		MDD (in %)	
	model	MSE	BCE	MSE	BCE	MSE	BCE	MSE	BCE
2016	MSCI	8.15	-	12.69	-	0.64	-	-11.47	-
	LSTM	48.69±18.20	59.32±11.43	16.41±1.09	16.51±1.55	**2.94±1.01**	**3.57±0.53**	-7.33±1.98	-7.43±1.31
	GCN	36.41±15.13	37.54±13.28	16.24±0.77	16.65±1.39	2.24±0.95	2.29±0.90	-8.84±1.58	-9.68±1.89
	GCN*	43.05±11.99	41.66±10.35	15.65±1.12	15.65±1.03	2.72±0.68	2.69±0.71	-7.96±1.42	-7.89±0.95
	GAT	35.86± 9.99	33.10±11.44	15.73±1.14	16.01±1.40	2.29±0.65	2.09±0.76	-8.68±1.40	-9.07±1.66
	GAT*	27.27± 5.24	30.58± 9.53	15.52±1.62	15.67±0.96	1.77±0.32	1.97±0.65	-10.02±1.48	-9.12±2.29
	HGCN	43.98±12.64	38.13± 9.61	17.06±1.01	16.68±1.32	2.58±0.72	2.33±0.71	-9.69±1.72	-8.29±1.60
	HGCN*	37.38±10.16	36.04± 8.37	16.02±1.27	15.30±1.29	2.36±0.71	2.38±0.64	-8.96±1.73	-8.70±1.87
2017	MSCI	23.07	-	5.71	-	4.04	-	-1.95	-
	LSTM	36.11±8.15	34.88± 6.25	8.87±0.47	9.23±0.50	4.07±0.87	3.79±0.68	-3.54±0.64	-4.08±0.81
	GCN	31.51±6.93	32.62± 6.40	8.36±0.44	8.57±0.69	3.78±0.86	3.79±0.63	-3.76±0.95	-3.71±0.57
	GCN*	32.61±5.89	28.55± 6.14	8.65±0.61	8.67±0.88	3.83±0.89	3.36±0.96	-3.22±0.48	-3.52±0.74
	GAT	30.42±4.57	26.61±13.51	8.36±0.43	8.43±0.34	3.63±0.46	3.19±1.61	-3.74±0.56	-4.15±1.43
	GAT*	29.50±4.16	29.02± 4.94	8.24±0.36	8.36±0.39	3.58±0.43	3.46±0.54	-3.34±0.50	-4.00±0.71
	HGCN	30.60±7.02	31.74± 8.14	9.24±0.82	8.80±0.88	3.38±0.95	3.72±1.21	-4.59±1.67	-4.16±1.21
	HGCN*	37.61±7.16	37.05± 5.80	8.80±0.36	15.30±1.29	**4.29±0.89**	**4.17±0.70**	-4.18±1.17	-4.55±1.051
2018	MSCI	-8.85	-	12.59	-	-0.70	-	-18.28	-
	LSTM	1.00±8.43	6.41±5.14	16.19±1.26	18.02±0.87	0.04±0.55	0.36±0.30	-17.49±3.74	-17.14±2.67
	GCN	-2.33±5.65	1.92±4.12	15.13±0.82	14.86±1.08	-0.16±0.39	0.12±0.29	-19.94±2.05	-9.97±1.95
	GCN*	0.43±7.25	0.21±3.30	14.50±2.36	15.04±1.56	0.10±0.47	0.03±0.22	-15.94±4.77	-14.78±3.69
	GAT	-1.14±9.44	0.36±7.48	14.25±1.09	15.02±1.07	-0.09±0.68	0.01±0.53	-16.40±3.46	-17.00±1.93
	GAT*	1.48±5.90	1.46±3.04	13.84±0.98	15.48±1.02	0.09±0.43	0.10±0.20	-14.68±2.65	-16.81±1.68
	HGCN	-2.59±4.77	8.97±8.21	16.41±1.62	15.55±1.27	-0.16±0.30	**0.56±0.52**	-18.91±4.46	-15.35±2.74
	HGCN*	2.91±6.83	-0.65±4.93	15.15±1.90	16.91±1.18	**0.21±0.45**	-0.03±0.29	-15.61±4.50	-18.39±2.15
2019	MSCI	28.40	-	10.09	-	2.81	-	-5.91	-
	LSTM	37.34±10.42	42.92±5.76	12.98±0.74	13.6±0.53	2.86±0.72	**3.16±0.41**	-7.73±0.80	-8.79±1.08
	GCN	29.59± 8.46	28.85±7.37	12.51±0.78	12.66±0.38	2.39±0.74	2.28±0.59	-9.34±2.23	-9.97±1.95
	GCN*	33.09± 5.28	31.91±5.56	12.18±1.42	11.78±1.12	2.71±0.25	2.70±0.31	-8.62±2.06	-8.09±1.35
	GAT	25.82± 5.96	27.74±5.12	12.17±0.62	11.94±0.54	2.13±0.50	2.33±0.44	-9.90±2.11	-8.56±1.17
	GAT*	37.12± 7.86	33.04±4.16	11.8±0.79	12.54±0.43	**3.16±0.68**	2.64±0.33	-7.44±1.83	-9.21±1.05
	HGCN	34.28± 9.96	37.8±7.33	13.20±0.52	13.60±0.50	2.61±0.80	2.78±0.55	-8.16±1.75	-9.45±1.12
	HGCN*	30.06± 8.20	37.06±8.84	12.10±0.94	13.42±0.80	2.46±0.61	2.75±0.62	-8.90±1.82	-9.17±0.85
2020	MSCI	16.50	-	28.60	-	0.58	-	-33.99	-
	LSTM	22.14±14.92	53.17±16.93	33.38±6.66	39.00±5.19	0.65±0.41	**1.36±0.39**	-36.95±5.96	-40.31±4.36
	GCN	14.49± 0.99	21.16±11.90	30.42±3.14	30.90±2.49	0.49±0.25	0.70±0.43	-30.85±4.07	-32.07±3.86
	GCN*	23.34±11.46	37.27± 4.70	33.67±1.75	31.40±2.49	0.70±0.36	1.19±0.18	-35.45±3.38	-30.63±3.47
	GAT	26.43± 8.40	21.56± 7.15	29.42±2.67	31.73±3.51	**0.90±0.29**	0.70±0.27	-29.00±4.38	-34.19±5.22
	GAT*	20.50±15.54	36.68±12.48	29.64±3.20	31.34±3.02	0.68±0.50	1.20±0.46	-32.50±4.82	-31.94±4.98
	HGCN	6.39±10.99	31.08±10.86	32.51±3.40	34.13±3.70	0.20±0.35	0.93±0.37	-36.67±4.76	-35.21±5.46
	HGCN*	20.19±16.60	36.28±19.89	31.67±4.25	30.84±3.28	0.66±0.54	1.15±0.62	-35.17±4.75	-32.46±2.88
2021	MSCI	36.50	-	11.27	-	3.24	-	-4.21	-
	LSTM	35.40±15.22	30.17± 3.82	15.05±1.23	16.48±0.48	2.34±0.91	1.83±0.25	-7.60±1.59	-8.72±0.79
	GCN	31.16±25.80	25.64±19.23	13.81±1.10	14.15±1.39	2.18±1.66	1.75±1.12	-6.22±1.26	-7.86±1.46
	GCN*	40.20±16.21	31.23±11.23	13.86±1.16	14.70±1.47	**2.89±1.15**	**2.10±0.70**	-5.90±1.08	-6.41±1.41
	GAT	32.86±13.09	23.12±12.12	13.88±1.57	13.51±1.13	2.38±1.01	1.71±0.85	-6.91±1.05	-7.24±1.47
	GAT*	32.82±20.66	25.28±12.84	13.81±1.64	14.24±1.03	2.35±1.46	1.74±0.83	-6.67±1.50	-7.44±0.83
	HGCN	31.32±20.71	19.86± 9.14	14.34±2.29	14.87±1.35	2.16±1.22	1.34±0.65	-6.61±1.35	-7.08±1.30
	HGCN*	30.27± 9.45	29.69± 9.35	14.52±1.84	15.72±1.05	2.11±0.67	1.91±0.71	-7.41±1.65	-8.74±0.88

Table 7: Annual backtest of the long 10% strategies on the MSCI World index

		Perf (in %)		Vol (in %)		SR		MDD (in %)		MDD/Vol	
	Model	MSE	BCE	MSE	BCE	MSE	BCE	MSE	BCE	MSE	BCE
2016	MCSI	8.15		12.69		0.64		11.47		0.9	
	LSTM	**70.68**	48.84	18.05	15.99	**3.92**	3.05	-6.62	-9.15	-0.37	-0.57
	GCN	53.65	37.15	17.45	15.95	3.07	2.33	-10.01	-13.19	-0.57	-0.83
	GCN*	47.13	38.12	14.72	17.71	3.20	2.15	**-5.21**	-10.76	**-0.35**	-0.61
	GAT	29.08	36.77	16.84	16.21	1.73	2.27	-9.68	-9.14	-0.57	-0.56
	GAT*	37.64	40.48	18.66	14.93	2.02	2.71	-10.97	-7.65	-0.59	-0.51
	HGCN	27.86	42.34	15.81	**14.54**	1.76	2.91	-11.81	-8.06	-0.75	-0.55
	HGCN*	33.80	35.94	18.96	15.29	1.78	2.35	-8.66	-10.36	-0.46	-0.68
2017	MCSI	23.07		5.71		4.04		1.95		0.34	
	LSTM	26.24	27.53	8.41	9.81	3.12	2.81	-3.95	-3.91	-0.47	-0.40
	GCN	27.49	34.17	7.84	7.82	3.50	4.37	-3.50	-3.31	-0.45	-0.42
	GCN*	30.42	17.03	**7.38**	10.08	4.12	1.69	-3.77	-4.85	-0.51	-0.48
	GAT	28.46	36.78	8.38	8.73	3.40	4.21	-3.94	-3.57	-0.47	-0.41
	GAT*	42.00	30.15	8.60	7.83	4.88	3.85	-3.43	-3.08	-0.40	-0.39
	HGCN	**46.04**	28.48	8.03	8.85	**5.73**	3.22	**-2.16**	-5.05	**-0.27**	-0.57
	HGCN*	25.51	36.47	9.20	9.32	2.77	3.91	-4.26	-4.76	-0.46	-0.51
2018	MCSI	-8.85		12.59		-0.70		18.28		1.45	
	LSTM	4.60	11.26	18.36	15.83	0.25	0.71	-15.47	-12.33	-0.84	**-0.78**
	GCN	-10.71	9.01	14.29	15.39	-0.75	0.59	-21.00	-14.54	-1.47	-0.94
	GCN*	5.33	-11.29	**11.58**	17.45	0.46	-0.65	**-9.51**	-20.58	-0.82	-1.18
	GAT	-2.59	-6.81	15.53	15.35	-0.17	-0.44	-21.31	-19.21	-1.37	-1.25
	GAT*	-7.28	-0.78	14.39	13.65	-0.51	-0.06	-19.87	-18.37	-1.38	-1.35
	HGCN	-3.17	-1.89	13.49	17.49	-0.24	-0.11	-18.13	-22.56	-1.34	-1.29
	HGCN*	**11.44**	-4.58	15.56	14.89	**0.74**	-0.31	-14.40	-14.33	-0.93	-0.96
2019	MCSI	28.4		10.09		2.81		5.91		0.59	
	LSTM	**53.54**	36.90	13.58	12.80	**3.94**	2.88	-7.82	-9.70	-0.58	-0.76
	GCN	33.13	33.26	11.96	12.20	2.77	2.73	-9.72	-10.70	-0.81	-0.88
	GCN*	25.97	27.02	**9.93**	14.24	2.61	1.90	-8.59	-12.09	-0.86	-0.85
	GAT	33.69	42.22	12.06	11.13	2.79	3.79	-8.15	**-5.16**	-0.68	**-0.46**
	GAT*	26.31	27.61	12.93	10.95	2.03	2.52	-10.68	-6.54	-0.83	-0.60
	HGCN	32.41	40.77	12.85	13.85	2.52	2.94	-10.26	-7.66	-0.80	-0.55
	HGCN*	47.63	31.45	13.03	13.71	3.66	2.29	-7.92	-8.41	-0.61	-0.61
2020	MCSI	16.5		28.6		0.58		33.99		1.19	
	LSTM	36.61	**77.53**	44.49	36.57	0.82	**2.12**	-45.58	-39.04	-1.02	-1.07
	GCN	21.85	18.29	25.40	29.96	0.86	0.61	-32.79	-32.93	-1.29	-1.10
	GCN*	14.33	38.80	33.00	36.85	0.43	1.05	-39.06	-35.19	-1.18	**-0.96**
	GAT	52.72	-5.13	30.60	**25.05**	1.72	-0.20	-30.02	-32.77	-0.98	-1.31
	GAT*	21.16	42.40	35.15	30.53	0.60	1.39	-40.40	-31.50	-1.15	-1.03
	HGCN	22.72	9.32	27.60	31.75	0.82	0.29	**-27.45**	-33.50	-0.99	-1.06
	HGCN*	12.14	47.49	38.95	31.07	0.31	1.53	-42.86	-34.74	-1.10	-1.12
2021	MCSI	36.5		11.27		3.24		4.21		0.37	
	LSTM	20.83	13.73	7.27	7.80	2.87	1.76	-2.02	-3.30	-0.28	-0.42
	GCN	20.81	15.06	9.90	10.04	2.10	1.50	-3.26	-3.56	-0.33	-0.35
	GCN*	16.70	7.39	7.96	**7.23**	2.10	1.02	-2.37	-4.78	-0.30	-0.66
	GAT	14.21	11.51	7.40	12.98	1.92	0.89	-3.62	-6.97	-0.49	-0.54
	GAT*	4.14	4.86	11.05	7.26	0.37	0.67	-2.97	**-1.83**	-0.27	-0.25
	HGCN	3.27	**50.12**	8.33	10.28	0.39	**4.87**	-3.03	-2.02	-0.36	**-0.20**
	HGCN*	16.85	8.92	7.79	8.07	2.16	1.11	-2.59	-5.31	-0.33	-0.66

Table 8: Annual backtest of the long 20% strategies on the MSCI World index

		Perf (in %)		Vol (in %)		SR		MDD (in %)		MDD/Vol	
	Model	MSE	BCE	MSE	BCE	MSE	BCE	MSE	BCE	MSE	BCE
2016	MCSI	8.15		12.69		0.64		11.47		0.9	
	LSTM	**48.99**	38.49	15.54	14.57	**3.15**	2.64	-7.08	-7.53	-0.46	-0.52
	GCN	41.86	34.24	14.74	14.73	2.84	2.32	-8.96	-9.94	-0.61	-0.67
	GCN*	41.38	29.88	13.29	15.12	3.11	1.98	**-5.03**	-9.20	**-0.38**	-0.61
	GAT	23.18	36.19	14.58	13.78	1.59	2.63	-10.16	-7.60	-0.70	-0.55
	GAT*	33.36	34.28	16.20	**12.64**	2.06	2.71	-9.83	-6.54	-0.61	-0.52
	HGCN	28.84	38.49	13.79	14.54	2.09	2.65	-9.66	-8.21	-0.70	-0.56
	HGCN*	34.52	31.18	16.16	12.78	2.14	2.44	-8.14	-7.24	-0.50	-0.57
2017	MCSI	23.07		5.71		4.04		1.95		0.34	
	LSTM	29.94	30.10	7.22	8.08	4.15	3.73	-2.57	-2.55	-0.36	-0.32
	GCN	26.68	30.58	6.82	6.86	3.91	4.46	-2.62	-2.24	-0.38	-0.33
	GCN*	34.32	21.73	**6.08**	7.86	5.65	2.77	-1.93	-3.25	-0.32	-0.41
	GAT	29.25	30.51	7.25	7.33	4.04	4.16	-2.94	-3.44	-0.41	-0.47
	GAT*	34.02	29.88	7.59	7.05	4.49	4.24	-2.38	-3.06	-0.31	-0.43
	HGCN	**43.86**	26.23	7.15	7.93	**6.14**	3.31	**-1.82**	-3.91	**-0.26**	-0.49
	HGCN*	28.35	32.36	8.03	8.01	3.53	4.04	-3.21	-4.09	-0.40	-0.51
2018	MCSI	-8.85		12.59		-0.70		18.28		1.45	
	LSTM	**6.31**	0.23	16.07	14.47	0.39	0.02	-14.00	-13.66	-0.87	-0.94
	GCN	-8.30	5.30	12.81	14.23	-0.65	0.37	-17.52	**-12.10**	-1.37	**-0.85**
	GCN*	1.18	-8.29	**11.76**	15.60	0.10	-0.53	-13.14	-19.75	-1.12	-1.27
	GAT	0.40	-2.99	15.27	15.21	0.03	-0.20	-19.77	-18.34	-1.29	-1.21
	GAT*	2.89	0.24	14.13	13.42	0.20	0.02	-14.59	-19.32	-1.03	-1.44
	HGCN	-4.70	-3.12	12.95	14.77	-0.36	-0.21	-17.24	-19.56	-1.33	-1.32
	HGCN*	6.14	-0.15	14.61	14.13	**0.42**	-0.01	-14.35	-13.71	-0.98	-0.97
2019	MCSI	28.4		10.09		2.81		5.91		0.59	
	LSTM	**45.16**	31.71	12.29	11.43	**3.67**	2.77	-7.78	-7.11	-0.63	-0.62
	GCN	30.62	30.44	11.03	11.50	2.78	2.65	-8.98	-9.56	-0.81	-0.83
	GCN*	27.14	32.83	**8.88**	13.21	3.06	2.48	-6.38	-9.21	-0.72	-0.70
	GAT	37.69	38.14	11.90	11.11	3.17	3.43	-8.74	**-5.29**	-0.73	**-0.48**
	GAT*	27.18	39.83	12.56	11.75	2.16	3.39	-11.17	-7.32	-0.89	-0.62
	HGCN	40.74	32.49	12.06	11.73	3.38	2.77	-8.34	-5.93	-0.69	-0.51
	HGCN*	35.47	39.44	12.33	12.16	2.88	3.24	-7.94	-7.25	-0.64	-0.60
2020	MCSI	16.5		28.6		0.58		33.99		1.19	
	LSTM	17.70	**62.22**	41.01	34.87	0.43	**1.78**	-41.89	-35.96	-1.02	-1.03
	GCN	20.25	17.60	27.54	30.47	0.74	0.58	-32.66	-32.25	-1.19	-1.06
	GCN*	21.68	31.24	31.25	31.86	0.69	0.98	-36.92	-33.50	-1.18	-1.05
	GAT	50.00	-5.05	31.33	**25.73**	1.60	-0.20	-29.07	-31.27	**-0.93**	-1.22
	GAT*	17.14	33.08	35.32	30.92	0.49	1.07	-39.85	-30.94	-1.13	-1.00
	HGCN	21.30	8.24	26.77	31.11	0.80	0.26	**-26.76**	-31.37	-1.00	-1.01
	HGCN*	10.82	53.21	38.58	30.60	0.28	1.74	-43.91	-29.86	-1.14	-0.98
2021	MCSI	36.5		11.27		3.24		4.21		0.37	
	LSTM	16.71	9.93	7.50	6.85	2.23	1.45	**-1.99**	-2.31	-0.26	-0.34
	GCN	16.65	14.80	10.88	10.51	1.53	1.41	-4.15	-4.14	-0.38	-0.39
	GCN*	15.27	7.32	7.49	7.08	2.04	1.03	-2.29	-2.87	-0.31	-0.40
	GAT	12.38	17.27	9.23	11.96	1.34	1.44	-2.96	-4.61	-0.32	-0.39
	GAT*	6.89	6.35	11.20	7.02	0.62	0.90	-3.26	-2.01	-0.29	-0.29
	HGCN	9.85	**32.55**	7.84	9.48	1.26	**3.43**	-2.11	-2.39	-0.27	**-0.25**
	HGCN*	17.74	12.69	9.24	**6.73**	1.92	1.88	-3.15	-2.73	-0.34	-0.41

Table 9: Annual backtest of the long 5% strategies on the S&P index

		Perf (in %)		Vol (in %)		SR		MDD (in %)		MDD/Vol	
	model	MSE	BCE	MSE	BCE	MSE	BCE	MSE	BCE	MSE	BCE
	SP500	12.32	-	17.74	-	0.69	-	15.99	-	0.90	-
	GCN*	23.46	24.01	27.80	25.72	0.84	0.93	22.62	21.98	0.81	0.85
2010	GAT*	15.45	**24.91**	28.77	17.80	0.54	**1.4**	27.49	**8.12**	0.96	**0.46**
	HGCN*	9.49	11.19	**13.31**	23.20	0.71	0.48	10.68	19.39	0.80	0.83
	SP500	0.00	-	22.91	-	0.00	-	19.39	-	0.85	-
	GCN*	-10.68	-6.92	39.05	33.33	-0.27	-0.21	40.49	31.54	1.04	0.95
2011	GAT*	-15.38	2.76	37.48	20.70	-0.41	0.13	38.40	20.00	1.02	1.00
	HGCN*	**11.22**	7.12	**18.59**	25.45	**0.60**	0.28	**15.39**	25.59	**0.83**	1.01
	SP500	12.91	-	12.49	-	1.03	-	9.94	-	0.80	-
	GCN*	18.03	**20.94**	21.36	18.40	0.84	1.14	21.49	15.59	1.01	0.85
2012	GAT*	17.23	17.47	21.58	13.04	0.8	**1.34**	23.01	9.64	1.07	**0.74**
	HGCN*	11.73	13.80	**9.77**	17.52	1.2	0.79	**7.35**	14.81	0.75	0.84
	SP500	28.45	-	10.88	-	2.61	-	5.76	-	0.53	-
	GCN*	**38.76**	25.30	15.17	10.61	2.56	2.39	6.57	5.79	0.43	0.55
2013	GAT*	29.50	29.84	12.13	**10.37**	2.43	**2.88**	6.75	**4.53**	0.56	**0.44**
	HGCN*	25.85	30.13	11.38	11.63	2.27	2.59	6.97	6.40	0.61	0.55
	SP500	10.98	-	11.17	-	0.98	-	7.40	-	0.66	-
	GCN*	12.82	28.09	16.77	10.68	0.76	2.63	13.06	5.57	0.78	0.52
2014	GAT*	5.12	**29.49**	18.28	10.36	0.28	**2.85**	15.32	**5.31**	0.84	**0.51**
	HGCN*	25.71	-8.68	**10.07**	15.56	2.55	0.56	**5.31**	17.79	0.53	1.14
	SP500	-0.70	-	15.22	-	-0.05	-	12.35	-	0.81	-
	GCN*	-10.83	-14.09	22.02	18.87	-0.49	-0.75	28.41	24.00	1.29	1.27
2015	GAT*	-13.09	-6.79	23.16	19.49	-0..56	-0.35	30.29	21.72	1.31	1.11
	HGCN*	**1.71**	1.58	**13.32**	17.21	**0.13**	0.09	**9.87**	18.94	1.74	**1.1**
	SP500	9.19	-	12.87	-	0.71	-	10.51	-	0.82	-
	GCN*	**43.04**	1.64	28.09	14.54	**1.53**	0.11	16.19	14.86	**0.58**	1.02
2016	GAT*	33.95	19.04	28.24	16.02	1.2	1.19	19.90	10.55	0.7	0.66
	HGCN*	12.26	8.59	**13.23**	13.91	0.93	0.62	**9.56**	10.43	0.72	0.75
	SP500	18.77	-	6.57	-	2.86	-	2.80	-	0.43	-
	GCN*	15.60	16.19	10.70	**8.49**	1.46	**1.91**	6.59	**4.58**	0.62	0.54
2017	GAT*	8.74	1.62	11.47	11.57	0.76	0.14	8.58	14.89	0.75	1.29
	HGCN*	**16.71**	10.92	11.26	8.92	1.48	1.22	8.84	4.74	0.79	**0.53**
	SP500	-6.03	-	16.72	-	-0.36	-	19.78	-	1.18	-
	GCN*	-5.06	**-3.29**	20.50	19.99	-0.25	**-0.16**	22.54	25.12	**1.1**	1.26
2018	GAT*	-4.97	-21.07	19.48	**15.72**	-0.26	-1.34	23.88	31.41	1.23	2
	HGCN*	-3.91	-14.85	18.67	16.75	-0.21	-0.89	**22.25**	26.00	1.19	1.55
	SP500	27.76	-	12.25	-	2.27	-	6.84	-	0.56	-
	GCN*	38.72	37.41	18.75	18.55	2.06	2.02	12.33	11.93	0.66	0.64
2019	GAT*	**43.74**	34.11	18.29	**12.62**	2.39	**2.7**	12.13	**5.52**	0.66	**0.44**
	HGCN*	29.24	28.63	16.37	14.40	1.79	1.99	11.09	10.14	0.68	0.7
	SP500	15.59	-	33.83	-	0.46	-	33.92	-	1.00	-
	GCN*	14.49	11.48	55.74	35.56	0.26	0.32	46.37	39.70	**0.83**	1.12
2020	GAT*	6.50	-10.38	57.56	**30.78**	0.11	-0.34	54.06	**39.57**	0.94	1.29
	HGCN*	**18.36**	3.50	43.19	42.26	**0.43**	0.08	41.54	46.43	0.96	1.1
	SP500	32.26	-	13.16	-	2.45	-	4.23	-	0.32	-
	GCN*	21.14	**64.14**	22.16	19.41	0.95	**3.3**	9.84	**5.59**	0.44	**0.29**
2021	GAT*	37.81	26.46	19.42	**15.99**	1.95	1.65	7.26	7.42	0.37	0.46
	HGCN*	22.76	51.10	16.97	18.04	1.34	2.83	6.91	5.93	0.41	0.33

Table 10: Annual backtest of the long 10% strategies on the S&P index

		Perf (in %)		Vol (in %)		SR		MDD (in %)		MDD/Vol	
	model	MSE	BCE	MSE	BCE	MSE	BCE	MSE	BCE	MSE	BCE
2010	SP500	12.32	-	17.74	-	0.69	-	15.99	-	0.90	-
	GCN*	17.22	**31.26**	26.60	22.29	0.65	**1.4**	22.49	18.97	0.85	0.85
	GAT*	14.04	21.52	27.20	17.55	0.52	1.23	24.88	9.17	0.91	**0.52**
	HGCN*	13.10	12.72	**13.82**	23.11	0.95	0.55	**9.12**	19.83	0.66	0.86
2011	SP500	0.00	-	22.91	-	0.00	-	19.39	-	0.85	-
	GCN*	-14.05	-1.75	35.87	27.69	-0.39	-0.06	38.71	30.37	1.08	1.1
	GAT*	-11.99	7.68	35.81	20.50	-0.33	0.37	34.72	19.53	0.97	0.95
	HGCN*	10.93	**17.32**	**17.97**	25.61	0.61	**0.68**	**14.19**	20.62	**0.79**	0.81
2012	SP500	12.91	-	12.49	-	1.03	-	9.94	-	0.80	-
	GCN*	17.11	16.34	20.67	17.61	0.83	0.93	22.59	18.94	1.09	1.08
	GAT*	14.02	**18.78**	19.89	12.79	0.7	**1.47**	23.61	8.63	1.19	0.67
	HGCN*	9.22	18.06	**9.30**	17.55	0.99	1.03	**6.16**	15.56	**0.66**	0.89
2013	SP500	28.45	-	10.88	-	2.61	-	5.76	-	0.53	-
	GCN*	33.50	**36.88**	14.22	12.45	2.36	2.96	6.17	7.65	**0.43**	0.61
	GAT*	30.19	31.44	11.97	**10.45**	2.52	3.01	5.39	**4.83**	0.45	0.46
	HGCN*	28.90	33.97	11.01	11.06	2.62	**3.07**	6.70	5.85	0.61	0.53
2014	SP500	10.98	-	11.17	-	0.98	-	7.40	-	0.66	-
	GCN*	10.12	12.91	15.81	13.12	0.64	0.98	13.29	11.17	0.84	0.85
	GAT*	5.28	21.57	16.08	10.35	0.33	2.08	12.49	5.55	0.78	0.54
	HGCN*	**25.21**	-11.98	**9.76**	16.66	**2.58**	-0.72	**5.09**	22.99	**0.52**	1.38
2015	SP500	-0.70	-	15.22	-	-0.05	-	12.35	-	0.81	-
	GCN*	-16.92	-12.18	20.14	18.66	-0.84	-0.65	27.84	23.06	1.34	1.24
	GAT*	-10.68	-7.19	20.80	18.19	-0.51	-0.4	24.83	19.63	1.19	1.08
	HGCN*	-1.63	**6.46**	**13.88**	16.92	-0.12	**0.38**	**13.05**	15.72	0.94	**0.93**
2016	SP500	9.19	-	12.87	-	0.71	-	10.51	-	0.82	-
	GCN*	**30.31**	6.47	24.17	14.60	**1.25**	0.44	15.97	14.88	0.66	1.02
	GAT*	25.64	17.74	24.05	14.72	1.07	1.21	16.43	11.29	0.68	0.77
	HGCN*	9.01	7.84	**12.96**	13.36	0.7	0.59	10.20	**6.46**	0.79	**0.63**
2017	SP500	18.77	-	6.57	-	2.86	-	2.80	-	0.43	-
	GCN*	18.92	9.85	10.24	10.20	1.85	0.97	6.63	8.34	0.65	0.82
	GAT*	7.76	1.61	10.40	10.55	0.75	0.15	9.89	12.34	0.95	1.17
	HGCN*	**45.44**	15.70	10.59	**8.41**	1.46	**1.87**	9.05	**4.52**	0.85	**0.54**
2018	SP500	-6.03	-	16.72	-	-0.36	-	19.78	-	1.18	-
	GCN*	-9.68	-7.24	19.54	18.06	-0.5	-0.4	23.54	23.25	1.2	1.29
	GAT*	-12.00	-16.88	18.66	**16.21**	-0.64	-1.04	25.36	27.14	1.36	1.67
	HGCN*	**-4.52**	-6.58	19.55	16.35	**-0.23**	-0.4	22.48	**21.89**	**1.15**	1.34
2019	SP500	27.76	-	12.25	-	2.27	-	6.84	-	0.56	-
	GCN*	**43.29**	30.33	17.92	15.65	2.42	1.94	12.53	9.70	0.7	0.62
	GAT*	37.39	30.49	17.82	12.16	2.1	**2.51**	12.25	**5.41**	0.69	**0.44**
	HGCN*	20.17	31.48	16.98	**14.20**	1.19	2.22	12.37	9.05	0.73	0.64
2020	SP500	15.59	-	33.83	-	0.46	-	33.92	-	1.00	-
	GCN*	16.10	3.35	54.10	**29.78**	0.3	0.11	46.94	**36.69**	**0.87**	1.23
	GAT*	7.78	-10.83	53.88	32.24	0.14	-0.34	51.39	40.61	0.95	1.26
	HGCN*	**18.70**	15.75	40.87	41.06	**0.46**	0.38	41.47	44.07	1.01	1.07
2021	SP500	32.26	-	13.16	-	2.45	-	4.23	-	0.32	-
	GCN*	43.58	**58.94**	20.45	16.71	2.13	**3.53**	7.22	**4.85**	0.35	**0.29**
	GAT*	49.18	38.24	19.19	**15.11**	2.56	2.53	7.49	5.36	0.39	0.35
	HGCN*	47.01	52.09	16.69	17.72	2.82	2.94	5.48	5.94	0.33	0.34

Table 11: Annual backtest of the long 20% strategies on the S&P index

		Perf (in %)		Vol (in %)		SR		MDD (in %)		MDD/Vol	
	model	MSE	BCE	MSE	BCE	MSE	BCE	MSE	BCE	MSE	BCE
2010	SP500	12.32	-	17.74	-	0.69	-	15.99	-	0.90	-
	GCN*	19.95	**33.30**	25.97	23.05	0.77	**1.44**	21.34	16.79	0.82	0.73
	GAT*	16.68	19.82	25.70	18.04	0.65	1.10	22.40	12.56	0.87	0.7
	HGCN*	10.62	15.89	**14.20**	22.39	0.75	0.71	**9.47**	20.18	**0.67**	0.9
2011	SP500	0.00	-	22.91	-	0.00	-	19.39	-	0.85	-
	GCN*	-8.77	2.97	25.97	28.89	-0.26	0.1	33.23	24.21	0.98	0.84
	GAT*	-6.55	3.93	33.73	21.26	-0.19	0.18	30.53	19.70	0.91	0.93
	HGCN*	4.76	**10.21**	**20.00**	25.61	0.24	**0.4**	**16.59**	23.27	0.**83**	0.91
2012	SP500	12.91	-	12.49	-	1.03	-	9.94	-	0.80	-
	GCN*	**20.20**	15.54	19.19	17.27	1.05	0.9	19.28	17.67	1	1.02
	GAT*	16.93	17.39	18.12	12.86	0.93	1.35	19.82	9.08	1.09	**0.71**
	HGCN*	13.11	17.17	**9.90**	15.82	**1.32**	1.09	**6.73**	13.02	0.83	0.91
2013	SP500	28.45	-	10.88	-	2.61	-	5.76	-	0.53	-
	GCN*	32.12	**40.25**	13.95	12.50	2.3	3.22	7.54	6.02	0.54	0.48
	GAT*	32.24	36.02	11.96	10.90	2.69	**3.3**	5.88	**4.46**	0.49	**0.41**
	HGCN*	31.44	32.60	**10.69**	10.88	2.94	3	6.06	6.11	0.57	0.56
2014	SP500	10.98	-	11.17	-	0.98	-	7.40	-	0.66	-
	GCN*	6.63	12.36	14.93	12.77	0.44	0.97	12.55	10.08	0.84	0.79
	GAT*	8.10	16.99	14.16	10.79	0.57	1.57	10.17	7.05	0.72	0.65
	HGCN*	**23.03**	3.46	**10.00**	13.88	**2.3**	0.25	**4.42**	13.28	**0.44**	0.96
2015	SP500	-0.70	-	15.22	-	-0.05	-	12.35	-	0.81	-
	GCN*	-12.40	-6.36	18.69	17.78	-0.66	-0.36	23.00	18.70	1.23	1.05
	GAT*	-10.41	-7.73	18.70	17.12	-0.56	-0.45	21.39	17.84	1.14	1.04
	HGCN*	2.18	**2.94**	**14.03**	14.91	0.16	**0.2**	**10.55**	13.88	**0.75**	0.93
2016	SP500	9.19	-	12.87	-	0.71	-	10.51	-	0.82	-
	GCN*	19.85	7.77	21.25	13.31	0.93	0.58	14.93	12.86	0.7	0.97
	GAT*	**20.48**	11.27	20.62	13.56	0.99	0.83	13.82	11.23	0.67	0.83
	HGCN*	13.79	5.50	**12.43**	14.17	**1.11**	0.39	**6.62**	9.62	**0.53**	0.68
2017	SP500	18.77	-	6.57	-	2.86	-	2.80	-	0.43	-
	GCN*	13.39	11.17	9.19	9.59	1.46	1.16	6.08	6.15	0.66	0.64
	GAT*	9.52	7.55	9.01	9.64	1.06	0.78	7.79	8.33	0.86	0.86
	HGCN*	**28.48**	15.84	9.37	**7.65**	**3.04**	2.07	4.67	**3.44**	0.5	**0.45**
2018	SP500	-6.03	-	16.72	-	-0.36	-	19.78	-	1.18	-
	GCN*	-11.48	-8.12	18.48	16.82	-0.62	-0.48	23.49	23.30	1.27	1.39
	GAT*	-11.22	-12.64	17.76	16.00	-0.63	-0.79	24.34	24.98	1.37	1.56
	HGCN*	**-4.52**	-7.88	17.74	**15.47**	**-0.25**	-0.51	**19.57**	21.46	**1.1**	1.39
2019	SP500	27.76	-	12.25	-	2.27	-	6.84	-	0.56	-
	GCN*	**39.53**	28.70	17.02	15.13	2.32	1.9	11.99	10.29	0.7	0.68
	GAT*	37.81	28.62	16.79	**12.00**	2.25	2.38	11.19	5.20	0.67	**0.43**
	HGCN*	30.11	36.37	15.47	13.38	1.95	**2.72**	9.34	**7.86**	0.6	0.59
2020	SP500	15.59	-	33.83	-	0.46	-	33.92	-	1.00	-
	GCN*	**17.20**	6.96	51.67	**30.03**	0.33	0.23	46.04	**37.15**	**0.89**	1.24
	GAT*	17.03	-0.37	49.77	33.44	**0.34**	-0.01	46.95	37.96	0.94	1.14
	HGCN*	11.40	5.85	39.66	38.13	0.29	0.15	40.63	42.27	1.02	1.11
2021	SP500	32.26	-	13.16	-	2.45	-	4.23	-	0.32	-
	GCN*	53.63	40.07	19.03	**14.46**	2.82	2.84	5.60	**4.91**	**0.29**	0.34
	GAT*	**61.06**	41.82	18.02	14.52	**3.39**	2.88	5.90	5.22	0.33	0.36
	HGCN*	27.46	52.75	15.18	15.89	1.81	3.32	6.14	5.12	0.4	0.32

Table 12: Ablation study of the three adjacency matrices taken in isolation

		Perf (in %)	Vol (in %)	SR	MDD (in %)	MDD/Vol
	MSCI	8.15	12.69	0.64	-11.47	0.90
	GCN sector	38.38±10.19	15.40±1.21	2.53±0.76	-9.19±1.84	0.59±0.09
	GAT sector	39.39±12.51	15.00±1.09	2.62±0.80	-8.00±1.93	0.54±0.13
2016	GCN corr	44.33±12.38	15.31±2.18	2.92±0.85	-7.22±2.19	0.46±0.09
	GAT corr	56.75±10.61	15.51±1.27	3.64±0.51	-6.63±1.21	0.43±0.07
	GCN supply chain	33.59±6.97	13.61±0.90	2.50±0.60	-7.13±1.36	0.52±0.09
	GAT supply chain	32.11±8.30	14.13±0.91	2.27±0.57	-7.73±1.18	0.55±0.08
	MSCI	23.07	5.71	4.04	-1.95	0.34
	GCN sector	35.22±6.60	8.49±0.43	4.15±0.80	-3.84±0.64	0.45±0.07
	GAT sector	29.36±6.26	8.11±0.64	3.64±0.81	-3.69±0.81	0.45±0.10
2017	GCN corr	31.3±10.85	9.08±1.34	3.55±1.40	-3.89±0.93	0.42±0.07
	GAT corr	41.78±8.31	8.48±1.20	5.07±1.33	-3.30±0.85	0.38±0.05
	GCN supply chain	32.73±7.29	8.02±0.65	4.09±0.86	-3.52±0.53	0.44±0.05
	GAT supply chain	31.77±7.83	8.27±0.29	3.83±0.87	-3.58±0.77	0.43±0.09
	MSCI	-8.85	12.59	-0.70	-18.28	1.45
	GCN sector	3.17±4.85	16.36±1.06	0.21±0.31	-18.92±1.84	1.16±0.07
	GAT sector	0.63±7.13	15.40±1.28	0.04±0.45	-17.95±2.69	1.16±0.14
2018	GCN corr	3.92±7.84	17.49±1.50	0.23±0.44	-16.93±3.48	0.97±0.20
	GAT corr	12.32±9.94	17.08±1.93	0.77±0.64	-16.06±4.69	0.92±0.18
	GCN supply chain	-1.66±5.03	15.95±0.63	-0.11±0.32	-18.62±4.05	1.17±0.25
	GAT supply chain	-0.71±3.00	15.19±0.55	-0.05±0.20	-16.22±2.20	1.06±0.13
	MSCI	28.40	10.09	2.81	-5.91	0.59
	GCN sector	35.11±5.34	14.13±0.39	2.48±0.36	-9.52±0.89	0.67±0.06
	GAT sector	33.57±8.72	13.39±0.93	2.52±0.68	-9.31±0.94	0.69±0.06
2019	GCN corr	35.66±3.98	13.88±1.22	2.58±0.30	-8.11±2.40	0.58±0.13
	GAT corr	38.78±8.73	13.38±1.58	2.91±0.64	-7.80±1.51	0.58±0.08
	GCN supply chain	35.17±7.47	13.04±0.74	2.70±0.61	-9.04±1.96	0.69±0.12
	GAT supply chain	35.89±7.41	12.78±0.79	2.80±0.53	-8.90±1.89	0.69±0.13
	MSCI	16.50	28.60	0.58	-33.99	1.19
	GCN sector	35.02±14.28	32.12±2.49	1.09±0.43	-30.30±3.21	0.95±0.12
	GAT sector	42.73±12.64	30.55±2.62	1.39±0.33	-28.69±2.90	0.95±0.14
2020	GCN corr	28.58±14.39	37.40±3.41	0.79±0.46	-37.17±5.29	0.99±0.10
	GAT corr	40.31±9.55	30.35±2.26	1.32±0.29	-31.02±2.24	1.02±0.06
	GCN supply chain	37.54±8.94	30.03±2.39	1.24±0.23	-31.41±1.40	1.05±0.08
	GAT supply chain	29.64±11.51	31.58±4.08	0.96±0.36	-31.68±3.50	1.01±0.11
	MSCI	36.50	11.27	3.24	-4.21	0.37
	GCN sector	15.18±9.40	14.18±0.72	1.07±0.67	-7.86±1.20	0.56±0.08
	GAT sector	16.53±6.30	13.94±0.51	1.19±0.46	-6.81±0.97	0.49±0.06
2021	GCN corr	33.41±8.87	15.05±1.27	2.20±0.44	-7.71±1.13	0.51±0.07
	GAT corr	33.41±15.49	14.54±1.76	2.33±1.07	-7.66±1.81	0.52±0.08
	GCN supply chain	34.24±7.85	13.73±1.75	2.54±0.69	-6.28±1.64	0.45±0.08
	GAT supply chain	43.02±11.43	15.14±0.96	2.81±0.61	-7.31±1.39	0.48±0.09