Machine Learning-based interaction network recovery in dynamical systems

A Thesis

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Pawan Ramprasad Bhure

Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

April, 2023

Supervisor: Prof. M.S. Santhanam

Certificate

This is to certify that this dissertation entitled **Machine Learning-based interaction network recovery in dynamical systems** towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Pawan Ramprasad Bhure at Indian Institute of Science Education and Research, Pune under the supervision of Prof. M.S. Santhanam, Professor, Department of Physics , during the academic year 2022-2023.

Prof. M.S. Santhanam

Committee:

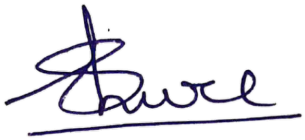
Prof. M.S. Santhanam

Prof. Apratim Chatterji

To my beloved parents for their unconditional love and support.

Declaration

I hereby declare that the matter embodied in the report entitled **Machine Learning-based interaction network recovery in dynamical systems** are the results of the work carried out by me at the Department of Physics, Indian Institute of Science Education and Research, Pune, under the supervision of Prof. M.S. Santhanam and the same has not been submitted elsewhere for any other degree.



Pawan Ramprasad Bhure

Acknowledgments

It is my great pleasure to express my deep gratitude to Prof. M.S. Santhanam for his exceptional mentorship and unwavering support. Without his guidance, this thesis would not have been possible. I am also immensely thankful to my TAC member, Prof. Apratim Chatterji, for his invaluable suggestions.

Furthermore, I extend my heartfelt thanks to IISER Pune for providing me access to the HPC facility (Param Brahma), which was crucial in the successful completion of my project. I am also grateful to the MHRD, Gov of India, for the financial support provided through the KVPY fellowship.

Finally, I want to thank my family and friends for their moral and emotional support.

Abstract

The study of interacting dynamical systems has been a topic of continuing research interest in various fields of science and engineering. In a collection of interacting agents, the interaction network contains information about which agent interacts with which other agents. In this thesis, given the information about the dynamics of agents, we use the graph neural network-based variational auto-encoder framework to recover the interaction network underlying the dynamical system and learn its dynamics. This is done entirely from observational data in a self-supervised manner.

We apply our model to two physical systems: the particles interacting via Hooke's law and the other interacting phase oscillators in the well-studied Kuramoto model. We also extend the applicability of this framework by applying it to the coupled system of financial instruments like stocks. It is well known that the log returns of several stocks are coupled with one another. Overall, we achieved an accuracy of greater than 89% in recovering the interaction matrix on all the tasks involving 5 interacting agents.

Contents

Introduction

Interacting dynamical systems are ubiquitous, and understanding them is one of the primary goals of Physics and other sciences. Generally, they consist of several agents that interact, i.e. one agent influences the state of the other agent.

The major challenge in studying and modelling such system (and their dynamics) is that we often do not have access to the precise physical interaction or laws governing the dynamics. But, only have access to the time series measurement of some of the dynamical variables recorded by the detectors.

In this thesis, we study such interacting systems by combining ideas from Network Science, Statistics and Machine Learning. Our primary focus is on explicitly inferring latent interaction network governing the dynamics from the observed data (refer: Figure 1).

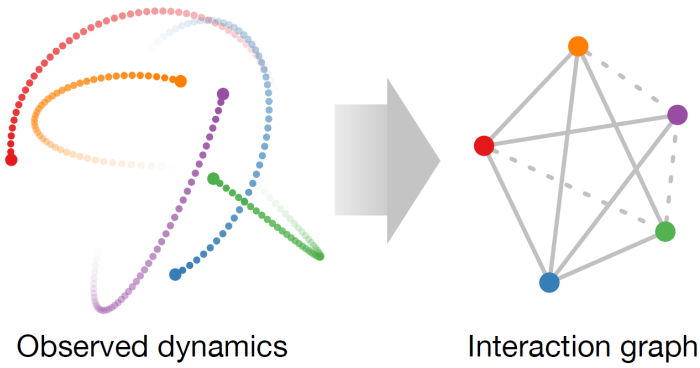


Figure 1: Trajectories of particles coupled via springs (Left) using an interaction network (right). A solid line in an interaction network denotes the particles are coupled through spring, and a dashed line denotes they are not. Source: *Kipf et al., 2018*[22]

In recent years, there has been an increase in interest in applying deep learning techniques to learn from simulated physical systems. Several studies have explored this idea and proposed var-

ious methods for breaking down complex physical systems into simpler parts and analyzing their interactions to learn the underlying dynamics ([5][3][13]). However, these previous works typically treat the interaction network recovery as the transient part of the learning process. However, in our case, we infer the interactions explicitly, which is one of our thesis's primary aims.

Chapter 1 introduces the idea of machine learning on networks highlighting various approaches. Chapter 2 discusses the model we used in our project. Chapter 3 describes the dynamical systems on which we applied the model. Chapter 4 discusses the results we obtained on different tasks associated with dynamical systems explored in Chapter 3. Lastly, Chapter 5 concludes the thesis.

Chapter 1

Machine Learning on Networks

1.1 Network

A network or graph is a mathematical structure used to represent and analyse interconnected entities. Networks can be classified into different categories based on their topology and size and with several other properties. However, in this thesis, we are primarily concerned with the Erdos Reyni Network.

1.1.1 Erdos Reyni Network

An ER (Erdos Reyni) network is a type of random network in which pair of nodes are connected with a probability 'p' (and not connected with probability $1-p$). It is denoted by $G(n,p)$, where n denotes the number of nodes in the network and p denotes the probability of having a link between two nodes.

The above definition can be easily extended to ER networks with multiple edge types. Suppose the ER network consists of 'L' types of links. In that case, apart from specifying n, we need to specify the $L-1$ probability parameters, where these $L-1$ parameters represent the probability of having a link of the first $L-1$ kind [1]. We will denote such ER networks as $G(n,p_1,p_2,...,p_{L-1})$, where p_r is the probability of having a link of type r.

[1] **Note**: The probability of having a link of L^{th} type is implicit

1.2 Machine Learning on Networks

Machine learning (ML) has significantly transformed various domains of science and engineering. Given that networks are omnipresent, it is of utmost importance to extend the application of ML to network-based problems. This will enable us to extract meaningful insights and facilitate deeper understanding of complex systems.

The application of machine learning techniques to network analysis involves a range of tasks that encompass but are not restricted to, the classification of nodes into distinct categories based on their relational structures, as well as the prediction of potential links within the network, particularly in the context of dynamical networks.

1.2.1 Challenges involved in network-based ML

The unique and complex nature of networks poses a significant challenge to the application of machine learning techniques on them. The state-of-the-art ML models like Convolutional Neural Networks (CNN)[12] and Recurrent Neural Networks (RNN)[18], which are very effective in dealing with images and time series data, respectively, cannot be directly applied to the networks due to their non-Euclidean nature. By the non-Euclidean nature of networks, we mean that the geometric properties of networks are fundamentally different from those of Euclidean space (or, in general, any Hilbert space). The following are the significant challenges while dealing with Networks:

- The notion of distance is well defined in Euclidean space (through the inner product), but this is not so with networks.
- The notion of locality and direction (left/ right) is also not defined in networks.
- Networks do not have canonical node ordering, i.e. ordering of nodes is a matter of personal choice.

Furthermore, networks/ graphs are an information-rich data structure and integrating its structural information into the ML model is also challenging. Two primary ways to deal with the abovementioned problems are conventional (traditional) and representation learning approaches.

Conventional ML methods rely on metrics such as node degree, clustering coefficient, various centrality measures, and methods like Neighbourhood overlap to derive topological information from networks. Such handcrafted features are **inflexible** (cannot be adjusted during learning) and **non-versatile** (their choice depends on the problem at hand).

Recently, many methodologies have surfaced that endeavour to develop representations capable of capturing the underlying structural information of networks. These techniques are collectively called representation learning.

1.3 Representation Learning

The representation learning techniques aim to find a transformation that embeds the network or its constituent entities, like nodes and edges, as points in some low-dimensional vector space. The primary goal is then to fine-tune this transformation such that spatial relationships in the embedding space closely resemble the relational structure of the network. Once the embedding space has been fine-tuned, the learned representations (embeddings) can be used as inputs for subsequent ML tasks like link prediction and node classification.

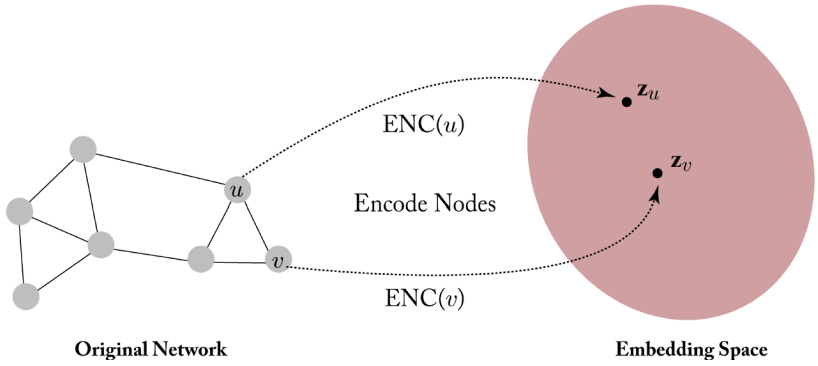


Figure 1.1: Embedding nodes by learning the transformation 'ENC'. $\mathbf{z}_u$ and $\mathbf{z}_v$ are node embeddings for nodes u and v, respectively. Source: *W.Hamilton, 2020* [15]

Compared to conventional learning methods, representation learning involves a different approach, whereby the task of feature engineering is not considered a distinct step in the learning process. Rather, in representation learning, features are learned directly from the data. Thus, feature engineering is transformed from a necessary pre-processing task to a machine-learning task in itself.

1.3.1 Formulating Node Embedding as an Optimisation Task

This section provides a mathematical framework for the node embedding problem. To begin, we consider an undirected and unweighted network $\mathscr{G}$= ($\mathscr{V}$, $\mathscr{E}$) for simplicity. The core of this framework revolves around two fundamental transformations: the Encoder (ENC) and the Decoder (DEC). The Encoder is a function that maps each node $v \in \mathscr{V}$ in the network to a low-dimensional vector representation $\mathbf{z}_v \in \mathbb{R}^d$ (where $\mathbf{z}_v$ is node v's embedding).

$$ENC : \mathscr{V} \rightarrow \mathbb{R}^d \tag{1.1}$$

In contrast, the decoder takes pair of node embeddings as an input and outputs the network attribute as required by the user from these node embeddings. For example, given the embeddings of a pair of nodes, the decoder may predict the possibility of a link between these nodes[23][1] or classify the node based on the community to which it belongs[16][24].

$$DEC : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^+ \tag{1.2}$$

The rationale behind the Encoder-Decoder framework is that if we can recover the complex high dimensional network information (like the global and local neighbourhood of the node in the network) from the low dimensional representations (embeddings) given by the Encoder, then in theory, embedding encapsulate all the essential information required for the subsequent Learning task. As a result, statistical/ ML algorithms can easily learn from and manipulate these low-dimensional embeddings. This approach is particularly advantageous because it enables the efficient handling and processing of complex network information.

If we denote the user-specified network attribute often called as a similarity metric on Network $\mathscr{G}$ by $S_{\mathscr{G}}$. The objective is to learn Encoder (ENC) and Decoder (DEC) functions such that:

$$DEC(ENC(u),\, ENC(v)) = DEC(\mathbf{z}_u, \mathbf{z}_v) \approx S_{\mathscr{G}}[u, v] \tag{1.3}$$

In the above expression, $S_{\mathscr{G}}[u, v]$ denotes the similarity metric between nodes u and v in the Network $\mathscr{G}$. If the similarity metric is defined as the existence of a link between nodes u and v in the network $\mathscr{G}$, then $S_{\mathscr{G}}[u, v]$ can be expressed as $A_{u,v}$, where $A_{u,v}$ is the (u, v) element of the adjacency matrix A of the Network $\mathscr{G}$.

The process of embedding a network can be mathematically expressed as the optimization of

the following objective function defined over the set of all possible node pairs $\mathscr{D}$:

$$\mathscr{L} = \sum_{(u,v)\in\mathscr{D}} \ell(DEC(\mathbf{z}_u,\, \mathbf{z}_v),\, S_{\mathscr{G}}[u,v]) \tag{1.4}$$

In this expression, ℓ represents the loss function that quantifies the discrepancy between the decoded value of the similarity metric, $DEC(\mathbf{z}_u, \mathbf{z}_v)$, and its true value, $S_{\mathscr{G}}[u,v]$.

1.3.2 Random Walk (RW) based Embeddings

The idea behind the RW-based embeddings is to simulate random walks on the network and use the resulting sequences of nodes visited during the walks to build a context for each node. This context can then be used to learn a representation (embedding) of the node that captures its structural relationships with other nodes in the network.

In this method, given some RW strategy R, we Calculate the probability that node v will be visited on a random walk that begins at node u. This information is utilised by optimising the embeddings.

The primary and distinct feature of RW based embedding approach is that instead of using the deterministic similarity metric, it quantifies the similarity between two nodes based on the Random walk (stochastic process). It is also efficient in the sense that here we do not have to consider each node pair but only those which co-occur on random walks.

Here our similarity metric will be the probability of a random walker visiting the node v on a random walk of a fixed number of steps starting at node u, $P(v|u)$. Decoder is generally quantified as a softmax of inner-product[2] of node embeddings in the following way:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) = \frac{exp(\langle \mathbf{z}_u | \mathbf{z}_v \rangle)}{\sum_{n\in\mathscr{V}} exp(\langle \mathbf{z}_u | \mathbf{z}_n \rangle)} \tag{1.5}$$

where $\mathbf{z}_u = \mathrm{ENC}(u)$ and our aim is $\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) \approx P(v|u)$. We can then formulate this problem as

[2]The notion of similarity in embedding space is chosen such that more similar the nodes u and v are higher will be $\langle \mathbf{z}_u | \mathbf{z}_v \rangle$ and vice-versa.

learning the encoder transformation *ENC* by minimising the cross entropy loss defined as follows:

$$\mathscr{L} = -\sum_{(u,v)\in\mathscr{D}} log[\text{DEC}(\mathbf{z}_u, \mathbf{z}_v)] \tag{1.6}$$

DeepWalk[28] and Node2Vec[11] are two widely used random walk-based approaches in network embedding. DeepWalk employs an unbiased random walk strategy, while Node2Vec utilises a biased random walk technique. The biasing mechanism in Node2Vec is designed to guide the random walker towards exploring the network in either depth-first or breadth-first order, as per the user's preference.

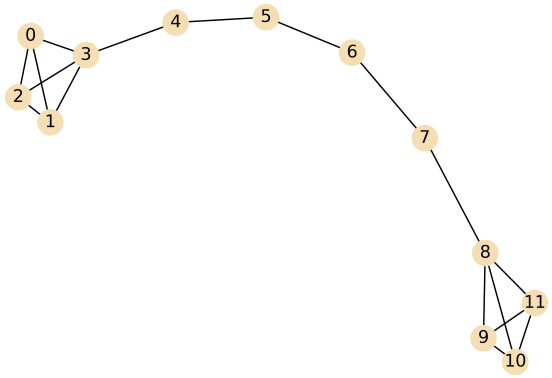


Figure 1.2: Two fully connected networks of 4 nodes joined via path graph of length 4/barbell graph.

Despite being computationally efficient and scalable, the RW-based methods do not capture structural similarities in the network. For example, in Figure 1.2, node 3 and node 8 are structurally similar both locally and globally, but their embeddings are very different (refer Figure: 1.3). Identifying topological similarities in a network is crucial in many physics problems, as structurally similar nodes may possess similar properties and may show similar dynamics.

Also, these methods do not capitalise on node or edge features, which may be an important source of information in certain problems. The other serious shortcoming of these methods is that we cannot get the embeddings for nodes that were not present in the network at the time of training. The solution to this problem is deep representation learning.

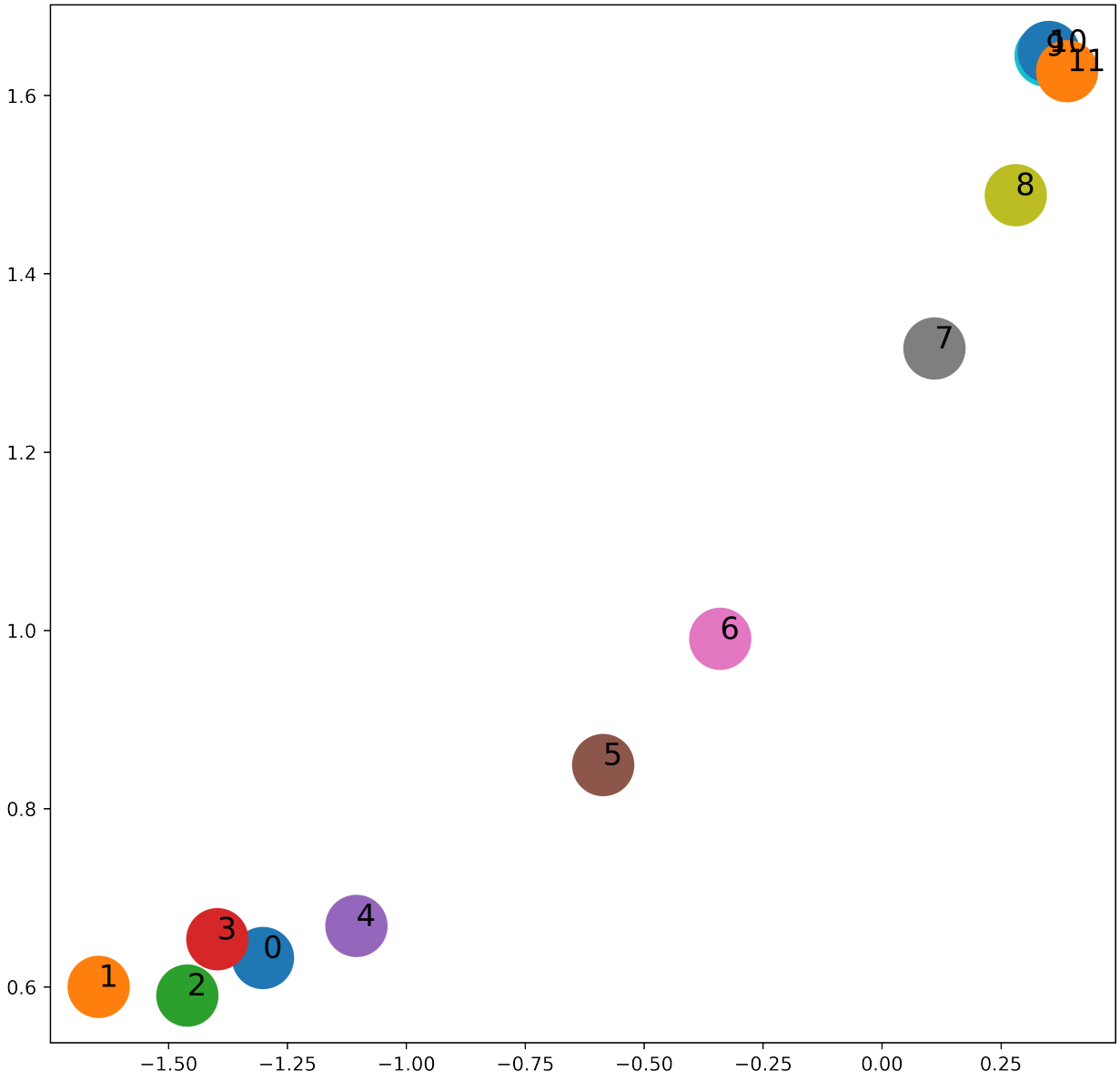


Figure 1.3: 2D embeddings of the network shown in Figure 1.2 obtained via Node2Vec with return parameter and inout parameter set to unity (This is the case where Node2Vec reduces to DeepWalk). Here 10 random walkers per node are used, and each random walker takes 80 steps.

1.4 Deep Representation Learning

Deep representation learning on networks is a technique used to learn hierarchical representations of network-structured data. The goal is to learn a set of features or representations that capture the structural and feature information associated with the Network or its components in a way that is useful for subsequent ML tasks such as node classification and edge prediction. Deep representation learning on networks typically involves the use of Graph Neural Networks (GNNs), which are specialised neural network architectures designed to process graph/ Network -structured data.

1.4.1 Graph Neural Network

The idea of using neural networks to process graph-structured data can be traced back to 2005, with the work of Gori et al[10]. However, the concept of Graph Neural Networks (GNNs) in its current form was first introduced in the year 2009 by Scarselli et al[30].

As previously discussed, the ordering of nodes within a network is subjective. This may pose a challenge if not considered during the design of a Graph Neural Network (GNN). To overcome this issue, it is imperative to ensure that GNNs are permutation invariant, or at the very least, permutation equivariant. In mathematical terms, any function G that accepts an adjacency matrix $\mathbf{A}$ as an input must satisfy either of the two properties:

$$\begin{aligned} G(\mathbf{PAP}^T) &= G(\mathbf{A}) \qquad \text{(Permutation Invariance)} \\ G(\mathbf{PAP}^T) &= \mathbf{P}G(\mathbf{A}) \qquad \text{(Permutation Equivariance)} \end{aligned} \tag{1.7}$$

here $\mathbf{P}$ is permutation matrix. Qualitatively, the Permutation invariance of GNN means GNNs' output is independent of node ordering. Permutation Equivariance means output depends only on the relative positions of the nodes in node ordering (not on their absolute position). In other words, equivariance implies that if the input network is transformed in some way, the output of the GNN should be transformed in the same way.

Fundamentally GNNs work by exchanging information between nodes in a network. This process is referred to as a message-passing operation. During this operation, each node sends its feature information to its neighbours, which then aggregate it to compute its new representa-

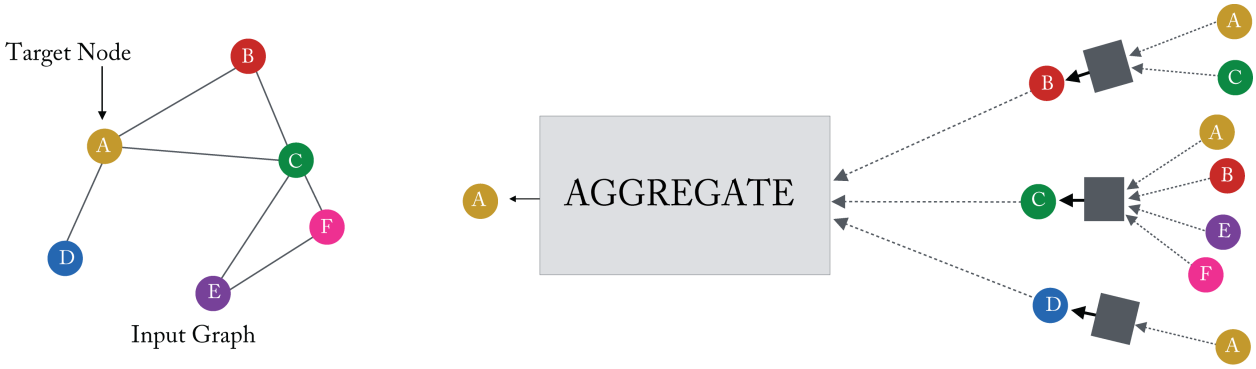


Figure 1.4: **2 layers of Message Passing in GNN**: Node A aggregates message from its neighbouring nodes, which in turn have already aggregated information from their respective neighbors. Source: *W.Hamilton, 2020* [15]

tion. This process is repeated over multiple iterations until the final representations are obtained. Typically a single GNN layer involve following operations:

$$\mathbf{h}_u^{(l+1)} = \sigma^{(l)}(\mathbf{h}_u^{(l)}, \text{AGGREGATE}^{(l)}(\{\mathbf{h}_v^{(l)}, \forall v \in \mathcal{N}_u\}) \tag{1.8}$$

where $h_u^{(l)}$ denotes the representation of node u at level l and $\mathcal{N}_u$ denotes the set of nodes in network neigbourhood of node u. Since we are considering the **set** of nodes, the operation (1.8) is permutation invariant by definition. AGGREGATE and σ are neural networks. σ can also simply be any non-linear function. Figure 1.4 shows two level of message passing operation in GNN.

Chapter 2

Methodology

Our approach (method) is greatly inspired by the Neural Relational Inference (NRI) model developed by Kipf et al[22]. Our experiments involved applying this method to various dynamical systems, each with unique challenges and nuances.

2.1 Message Passing Framework

The NRI model uses GNNs, which work by applying a message-passing algorithm on networks. Like [9], the one round of node-to-node message passing in GNN is defined in the following way. Given a graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ with vertices $v \in \mathscr{V}$ and edges $e = (v, v') \in \mathscr{E}$.

$$v \to e: \ \mathbf{h}^l_{(i,j)} = f^l_e([\mathbf{h}^l_i, \mathbf{h}^l_j, \mathbf{x}_{(i,j)}]) \tag{2.1}$$

$$e \to v: \ \mathbf{h}^{l+1}_j = f^l_v([\sum_{i \in \mathscr{N}_j} \mathbf{h}^l_{(i,j)}, \mathbf{x}_j]) \tag{2.2}$$

here $\mathbf{h}^l_i$ is the l^{th} layer node embedding of v_i and $\mathbf{h}^l_{(i,j)}$ is the edge embedding of $e_{(i,j)}$ at level l. Additionally, $\mathbf{x}_i$ and $\mathbf{x}_{(i,j)}$ refer to the node and edge features, respectively. The set of node ids of nodes neighboring node j is denoted by $\mathscr{N}_j$, while $[.,.]$ signifies vector concatenation. The functions f_v and f_e denote neural networks which are specific to node and link, respectively. Equations (2.1) and (2.2) provide insight into how nodes and edges within the network communicate with one another.

2.2 NRI Model

The NRI model is founded upon the architecture of the variational autoencoder, which was introduced by Kingma and Welling [21] and Rezende et al[29]. This model comprises two interrelated components that operate collaboratively: an encoder and a decoder. The encoder component predicts the latent interaction network by utilising given trajectories. In contrast, the decoder tries to reconstruct the trajectories based on the latent interaction network predicted by an encoder (and given initial condition). We will optimise the parameters of Encoder and Decoder in a self-supervised manner by minimising this reconstruction error[1].

The inputs to our model are the state vectors defining the time evolution of each of the N agents/nodes. We indicate the feature vector of the agent v_i at time t by $\mathbf{x}_i^t$. At time t the set of features of all N agents are denoted by $\mathbf{x}^t = \{\mathbf{x}_1^t, ..., \mathbf{x}_N^t\}$, similarly the trajectory of agent v_i is denoted by $\mathbf{x}_i = (\mathbf{x}_i^1, ..., \mathbf{x}_i^T)$. Finally, we collectively denote the set of trajectories of all the agents (objects) constituting the dynamical system by $\mathbf{x} = (\mathbf{x}^1, ..., \mathbf{x}^T)$. Given an unknown Network $\mathbf{z}$ where $\mathbf{z}_{ij}$ represents the edge type between two agents v_i and v_j, we aim to predict $\mathbf{z}_{ij}$ and learn the dynamical model.

Since we have cast this problem as a variational autoencoder (VAE), we will maximise ELBO (Evidence Lower BOund)[21]:

$$\mathscr{L} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[log p_\theta(\mathbf{x}|\mathbf{z})] - KL[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})] \tag{2.3}$$

Where $q_\phi(\mathbf{z}|\mathbf{x})$ is the output of an encoder, $p_\theta(\mathbf{x}|\mathbf{z})$ is the decoder's output , $p_\theta(\mathbf{z})$ is the prior probability distribution[2] over edge types (which we will take to be uniform distribution) and KL denotes the Kullback–Leibler divergence[31] between two probability distributions.

Maximising $\mathscr{L}$ means maximising the first term and minimising the second term in the LHS of equation (2.3). Maximising the first term is mathematically equivalent to minimising the reconstruction error. Minimising the second term ensures the encoder's output is close to the prior probability distribution[3].

[1] The discrepancy between actual trajectory and trajectory reconstructed by Decoder.
[2] Prior probability is chosen based on domain knowledge or assumptions about the data being modeled.
[3] Second term can be treated as a regularisation term.

2.2.1 Encoder

Encoder aims to infer the latent interaction network $\mathbf{z}$ given the observed trajectories $\mathbf{x} = (\mathbf{x}^1, ..., \mathbf{x}^T)$. The encoder returns a probability distribution of $\mathbf{z}_{ij}$[4] which we will denote by $q_\phi(\mathbf{z}|\mathbf{x})$. We do not know the interaction network priorly; hence we will use GNN on a complete (fully connected) network formed by given agents. Mathematically we formulate the encoder as $q_\phi(\mathbf{z}_{ij}|\mathbf{x}) = \text{softmax}(f_{enc,\phi}((\mathbf{x})_{ij,1:K}))$[5], where GNN acting on complete network is denoted as $f_{enc,\phi}(\mathbf{x})$.

The Encoder accepts trajectories $\mathbf{x}_1, ..., \mathbf{x}_K$ as input and computes messages based on the following message passing framework:

$$\mathbf{h}_j^1 = f_{emb}(\mathbf{x}_j) \tag{2.4}$$

$$v \to e: \ \mathbf{h}_{(i,j)}^1 = f_e^1([\mathbf{h}_i^1, \mathbf{h}_j^1]) \tag{2.5}$$

$$e \to v: \ \mathbf{h}_j^2 = f_v^1(\sum_{i \neq j} \mathbf{h}_{(i,j)}^1) \tag{2.6}$$

$$v \to e: \ \mathbf{h}_{(i,j)}^2 = f_e^2([\mathbf{h}_i^2, \mathbf{h}_j^2]) \tag{2.7}$$

We model the edge type probability as $q_\phi(\mathbf{z}_{ij}|\mathbf{x}) = \text{softmax}\,(\mathbf{h}_{(i,j)}^2)$. With a single pass through Eqs. (2.4)–(2.5), the embedding $\mathbf{h}_{(i,j)}^1$ is solely determined by the values of $\mathbf{x}_i$ and $\mathbf{x}_j$, disregarding any potential interplay with other nodes. Conversely, $\mathbf{h}_j^2$ draws on knowledge from the entire network to infer its embedding. This is the reason for using second order embeddings ($\mathbf{h}_{(i,j)}^2$). In above Eqs $f(...)$ is either MLP[6] in 'MLP Encoder' or 1D convolutional networks (CNNs) with attentive pooling like [26] in 'CNN Encoder'.

The next step is to do sampling from $q_\phi(\mathbf{z}_{ij}|\mathbf{x})$ which is a simple task but since $\mathbf{z}_{ij}$s are discrete variables therefore we cannot employ the reparametrization trick[21] to backpropagate errors through the sampling process. To overcome this problem we will approximate $\mathbf{z}_{ij}$ by a method called 'Continuous Relaxation of Discrete Random Variables'[27][19] in the following way:

$$\mathbf{z}_{ij} = \text{softmax}\left(\frac{\mathbf{h}_{(i,j)}^2 + \mathbf{g}}{\tau}\right) \tag{2.8}$$

[4]Here z_{ij} denotes a discrete categorical variable representing the type of link between agent v_i and v_j.
[5]K denotes net interaction (edge) types and ϕ denotes the parameters of encoder.
[6]MLP: Layers (2 in our case) of linear transformations, each followed by a nonlinearity

where vector $\mathbf{g} \in \mathbb{R}^K$ contains elements drawn from Gumbel(0, 1) distribution. The softmax temperature parameter τ controls the samples' "smoothness". As $\tau \to 0$, the distribution becomes discrete samples.

2.2.2 Decoder

The decoder's task is to forecast the future evolution of dynamics $p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^t,...,\mathbf{x}^1,\mathbf{z})$[7] based on the initial state(s) of the system and the interaction network anticipated by the encoder. Mathematically the decoder $p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^t,...,\mathbf{x}^1,\mathbf{z})$ models $p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^t,...,\mathbf{x}^1,\mathbf{z})$ with a GNN conditioned on $\mathbf{z}$. Following is the message-passing scheme employed by decoder:

$$v \to e: \ \mathbf{h}^t_{(i,j)} = \sum_k z_{ij,k} f^k_e([\mathbf{x}^t_i, \mathbf{x}^t_j]) \tag{2.9}$$

$$e \to v: \ \boldsymbol{\mu}^{t+1}_j = \mathbf{x}^t_j + f_v(\sum_{i \neq j} \mathbf{h}^t_{(i,j)}) \tag{2.10}$$

$$p(\mathbf{x}^{t+1}_j|\mathbf{x}^t,\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}^{t+1}_j, \sigma^2\mathbf{I}) \tag{2.11}$$

Here $z_{ij,k}$ denotes the probability of a link between node i and j belonging to class 'k' as predicted by an encoder. σ^2 is a fixed variance which we have taken to be 5×10^{-5} for all tasks. As present state $\mathbf{x}^t_j$ is added in equation (2.10), so our model actually learns change in the state $\Delta\mathbf{x}^t_j$.

The reconstruction loss term of equation (2.3) is of the form $\sum_{t=1}^{T} log[p(\mathbf{x}^t|\mathbf{x}^{t-1},\mathbf{z})]$ i.e. it involves only single step prediction. The problem with optimising such a loss function is that it ignores the fact that in short time interval the interactions can have negligible effect. Take, for instance, physics simulations where assuming a constant velocity can be a reasonable approximation for a short interval. In such scenarios, the decoder may not fully consider the interactions for predicting future state and will not be penalised much because of low reconstruction loss.

We will tackle this problem in two ways, First, instead of predicting a single time step into the future we will predict multiple steps. Here decoder which ignores interactions will perform significantly worse. Second, as opposed to Interaction Networks[3], where a single neural network was used to calculate the messages our model have a distinct MLP for each edge type. This makes the dependency on the edge type more evident and makes it more difficult for the model to

[7]For markovian dynamics $p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^t,...,\mathbf{x}^1,\mathbf{z}) = p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^t,\mathbf{z})$

disregard it.

To forecast multiple steps in future, we need to replace the actual input $\mathbf{x}^t$ with the estimated mean $\boldsymbol{\mu}^t$ for M[8] steps, then feed in the correct preceding step and repeat. Formally, if we write our decoder as $\boldsymbol{\mu}_j^{t+1} = f_{\text{dec}}(\mathbf{x}_j^t)$ then we get the following:

$$\begin{aligned}
\boldsymbol{\mu}_j^2 &= f_{\text{dec}}(\mathbf{x}_j^1) \\
\boldsymbol{\mu}_j^{t+1} &= f_{\text{dec}}(\boldsymbol{\mu}_j^t) \qquad\qquad t = 2, ..., M \\
\boldsymbol{\mu}_j^{M+2} &= f_{\text{dec}}(\mathbf{x}_j^{M+1}) \\
\boldsymbol{\mu}_j^{t+1} &= f_{\text{dec}}(\boldsymbol{\mu}_j^t) \qquad\qquad t = M+2, ..., 2M \\
&\dots
\end{aligned} \tag{2.12}$$

When we backpropagate through the entire process, the decoder ignoring the interactions is now highly inefficient since the errors compound for M steps.

Recurrent decoder

In some dynamical systems, there exist long-term temporal dependencies. In such situations, simple MLP-based decoders are less effective. RNNs are shown to be highly effective at modelling and generating sequential data and in capturing the long-term dependencies[17][7].

We model the recurrent decoder by modifying the original neural message passing scheme (as discussed in equations (2.9) to (2.11)) and by adding a Gated Recurrent Unit[6] to it in the following way:

$$v \to e: \quad \mathbf{h}_{(i,j)}^t = \sum_k z_{ij,k} f_e^k([\mathbf{h}_i^t, \mathbf{h}_j^t]) \tag{2.13}$$

$$e \to v: \quad \text{MSG}_j^t = \sum_{i \neq j} \mathbf{h}_{(i,j)}^t \tag{2.14}$$

$$\mathbf{h}_j^{t+1} = \text{GRU}([\text{MSG}_j^t, \mathbf{x}_j^t], \mathbf{h}_j^t) \tag{2.15}$$

$$\boldsymbol{\mu}_j^{t+1} = \mathbf{x}_j^t + f_{\text{out}}(\mathbf{h}_j^{t+1}) \tag{2.16}$$

$$p(\mathbf{x}^{t+1} | \mathbf{x}^t, \mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}^{t+1}, \sigma^2 \mathbf{I}) \tag{2.17}$$

[8] We have taken $M = 10$ for all experiments except "Interacting Stocks" Experiment where we used $M = 5$

In equation (2.13) edge embeddings are computed which are then aggregated at each node v_j to get an aggregated message MSG_j^t. This aggregated message is concatenated with feature vector of node v_j, $\mathbf{x}_j^t$ and passed to GRU along with the previous hidden state $\mathbf{h}_j^t$ (refer: equation (2.15)). The output of GRU is passed through f_{out} which is an MLP and combined with $\mathbf{x}_j^t$ to give predicted mean $\boldsymbol{\mu}_j^{t+1}$.

2.2.3 Training

We first pass our training features (trajectories) through an encoder which outputs $q_\phi(\mathbf{z}_{ij}|\mathbf{x})$. After this we use concrete distribution[27] as an approximation to $q_\phi(\mathbf{z}_{ij}|\mathbf{x})$ and sample $\mathbf{z}_{ij}$ from this approximate distribution. We then use the decoder to compute $\boldsymbol{\mu}^2, ..., \boldsymbol{\mu}^T$. The first term of our cost function equation (2.3) $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[log p_\theta(\mathbf{x}|\mathbf{z})]$ is a reconstruction error, which we will estimate by Negative Log Likelihood (NLL) of a Gaussian distribution[9]:

$$-\sum_i \sum_{t=2}^{T} \frac{\|\mathbf{x}_i^t - \boldsymbol{\mu}_i^t\|}{2\sigma^2} \tag{2.18}$$

The second term of our cost function (2.3) $KL[q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})]$ is a KL divergence between encoder probability $q_\phi(\mathbf{z}|\mathbf{x})$ and prior probability $p_\theta(\mathbf{z})$ which we have taken to be uniform. The KL divergence with a uniform prior is simply the sum of entropies [10]:

$$\sum_{i \neq j} H(q_\phi(\mathbf{z}_{ij}|\mathbf{x})) \tag{2.19}$$

We will iteratively change the model parameters by optimising the equation (2.3) and backpropagating the errors. We used Adam Optimiser[20] for optimisation.

[9] Note: We have set constant part of NLL as zero.

[10] Actually, the KL divergence with a uniform prior is the sum of entropies plus constant, but we have set the constant to zero.

Chapter 3

Applications to Dynamical Systems

We have experimented with two physical and one financial dynamical system. We will call our first experiment "Interacting Particles" second "Interacting Oscillators", and the third "Interacting Stocks". Instead of using natural data, we have simulated these dynamical systems to generate synthetic data. The reason for this approach is that in many natural (physical) settings, we have access to the time evolution of state variables, but the interaction network is inaccessible. This inaccessibility of the actual interaction network makes it difficult to assess the effectiveness of our model on natural data.

3.1 Interacting Particles - Hooke's Law

Our system consists of N particles[1] trapped within a 2D box. There is no external force on these particles except elastic collision with the walls of the box. We randomly couple (with probability 0.5) each pair of particles with a spring of spring constant $K = 1$. This results in the formation of an Erdos Reyni (ER) Network $G(n = N, p = 0.5)$[2], where particles serve as nodes and springs serve as edges. The particles attached by springs will interact via Hooke's law $F_{ij} = -K(s_i - s_j)$ where F_{ij} is the force applied by particle v_j on particle v_i. K denotes the spring constant of the spring and s_i is the position vector of particle v_i.

[1] Particles are assumed to be point masses

[2] p : probability of having link of type $K = 1$

Our approach can be extended to systems with multiple link types. For instance, to simulate a system with three link types, we added an additional link type with a spring constant of $K = 0.5$ and randomly sampled all three link types with equal probability. This can be viewed as generating an ER network $G\left(n = N, p_1 = \frac{1}{3}, p_2 = \frac{1}{3}\right)$. Where p_1 and p_2 are probabilities of coupling a node pair by a link of type $K = 1$ and $K = 0.5$, respectively.

We will sample the initial position of particles from the normal distribution $\mathcal{N}(\mu = 0, \sigma = 0.5)$. As for the initial velocity, we'll generate a random vector with a magnitude of 0.5. Using this information, we can then calculate the trajectory of the particles over time by solving Newton's equations of motion.

Since our system is conservative, we have integrated Newton's equations with the leapfrog integration[3] method with a step size of 0.001. The input features to the model will be generated by concatenating the 2D position (x, y) vector with the 2D velocity $(\dot{x}, \dot{y})$ vector. We then downsampled the features with a sampling frequency of 100 to get the final set of features (trajectories) for training and testing our model.

3.2 Interacting Oscillators - Kuramoto Model

The Kuramoto oscillator model[25] is a widely used mathematical framework to study the synchronisation of phase coupled oscillators in various physical systems. In this model, a group of N oscillators are interconnected through an interaction network, determining the strength and coupling pattern between them. The dynamical equation that governs the evolution of each oscillator's phase, θ_i, is given by:

$$\frac{d\theta_i}{dt} = \omega_i + \sum_{j \neq i} A_{ij} \sin(\theta_i - \theta_j) \tag{3.1}$$

where ω_i is the oscillator's intrinsic frequency and A_{ij} represents the coupling constant between oscillators v_i and v_j. For the purposes of simulation, we have used an undirected and unweighted ER network as the interaction network, allowing A_{ij} to be interpreted as the $(i, j)^{\text{th}}$ element of the adjacency matrix. Under different conditions, such as varying the coupling strength or distribution of intrinsic frequencies, the Kuramoto model can show a variety of intricate dynamical behaviours.

[3] Leapfrog integration is a very effective numerical integration method for solving dynamical equations of **conservative syatems**[14].

Our focus is not on exploring the complex behaviour exhibited by this dynamic system. Instead, we are trying to find, based on the time evolution of the phases of oscillators, how accurately we can predict the interaction network governing this time evolution.

In this experiment, we model a system of phase-coupled oscillators in 1D using two different N values, $N = 5$ and $N = 10$. Each oscillator has an intrinsic frequency ω_i, and an initial phase ϕ_i sampled uniformly from the intervals $[1, 10)$ and $[0, 2\pi)$, respectively. We randomly select pairs of oscillators v_i and v_j and connect them with a coupling constant A_{ij}, which has a value of 1 with a probability of 0.5 and 0 with the same probability.

We have used the Dormand-Prince 5(4) method[8], also known as DOPRI5, to solve the Kuramoto ODE (3.1) and obtain the time series of phases θ_i. DOPRI5 is a Runge-Kutta method of order 5 with an embedded fourth order method for stepsize control. We then downsampled θ_i by factor of 10 and use this to generate the final trajectories $\mathbf{x}_i$ by concatenating $\frac{d\theta_i}{dt}$, $sin\theta_i$ and ω_i $\forall$ oscillator $v_i \in \{v_i, ..., v_N\}$.

The idea of taking model input as a concatenation of $\frac{d\theta_i}{dt}$, $sin\theta_i$ and ω_i is inspired by Kipf et al., 2018[22]. However, including ω_i as one of the inputs is not physical since it is an intrinsic parameter of the dynamical system and is often inaccessible. In practice, the accessible information is limited to the time evolution of phases θ_i. We propose a creative solution to solve this problem by providing estimated ωs as an input instead of the actual ωs. We estimate ω_i as the DC component of the Fourier Transform of $\frac{d\theta_i}{dt}$. We then compare the performance of our model in both scenarios, i.e., using actual vs estimated ω as one of the inputs.

3.3 Interacting Stocks - Geometric Brownian Motion

In this experiment, we extend the application of our model to a dynamical system from the domain of quantitative finance. Our dynamical system will be the interacting financial instruments like stocks. Similar to the previous two experiments, we will use synthetic data instead of natural stock data; this is to test our model's performance on a known interaction network. We will use the Geometric Brownian Motion (GBM)[2][4] to simulate the time evolution of stock prices.

3.3.1 Geometric Brownian Motion

Here we will motivate the idea of GBM. Qualitatively the fundamental idea behind GBM is that the change in the price of a financial instrument (like stock) is determined by its current price, rate of return r and some random noises[32]. To begin with, we will focus on the deterministic aspects of stock price changes before accounting for the stochastic component. Mathematically the deterministic part can be easily modelled as follows:

$$\frac{ds_t}{dt} = rs_t \tag{3.2}$$

$$\implies ds_t = s_t r dt \tag{3.3}$$

Where s_t is the price of stock at time t. r denotes the rate of return. The eqaution (3.3) covers the deterministic part of the time evolution of stock price. Since most of the financial instruments have inherent randomness associated with them so to account for this we will add a stochastic component to equation (3.3).

$$ds_t = s_t(rdt + \sigma dB_t) \tag{3.4}$$

Where B_t denotes a stochastic process which is most commenly choosen to be a Brownian Motion (type of Wiener process). σ is used to scale the contribution of brownian motion. Brownian motion is a continuous-time stochastic process $B = \{B_t : t \geq 0\}$ which starts at zero. The increments of the process are independent and are normally distributed with mean 0 and variance equal to time interval.

The equation (3.4) is the dynamical equation for the GBM model and is a Stochastic Differential Equation (SDE). Like any other Ito process, GBM has both drift term $s_t r dt$ and diffusion term $\sigma s_t dB_t$. Before solving the equation (3.4), we will discuss the Ito's lemma.

Ito's Lemma

Ito's lemma is an extension of the chain rule in calculus, specifically for stochastic calculus. The chain rule in calculus is used to differentiate composite functions, where the output of one function becomes the input of another function. In stochastic calculus, Ito's lemma is used to differentiate stochastic processes that are functions of time and another stochastic process.

Let us consider an Ito process s_t satisfying following stochastic differential equation[4]:

$$ds_t = s_t(rdt + \sigma dB_t) \tag{3.5}$$

where r and σ are drift and diffusion coefficient respectively. Let $f(s_t,t)$ be any differentiable function of Ito process s_t and time. Starting with the Taylor expansion of $f(s_t + ds_t, t + dt)$, we have:

$$f(s_t + ds_t, t + dt) = f(s_t,t) + \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial S_t}ds_t + \frac{1}{2}\frac{\partial^2 f}{\partial s_t^2}ds_t^2 + \text{higher order terms}(\mathcal{O}((dt)^2)) \tag{3.6}$$

$$\implies df = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial S_t}ds_t + \frac{1}{2}\frac{\partial^2 f}{\partial s_t^2}ds_t^2 + \text{higher order terms}(\mathcal{O}((dt)^2)) \tag{3.7}$$

In the limit $dt \to 0$ we can ignore higher order terms$(\mathcal{O}((dt)^2))$ to get final form of Ito lemma.

Solving SDE

Let $f(s,t) = log(s_t)$ then using Ito lemma we have:

$$df = \frac{\partial f}{\partial t}dt + \frac{\partial f}{\partial S_t}dS_t + \frac{1}{2}\frac{\partial^2 f}{\partial S_t^2}dS_t^2 = \frac{1}{S_t}dS_t - \frac{1}{2S_t^2}dS_t^2 \tag{3.8}$$

From (3.4) we have

$$dS_t^2 = (rS_t dt + \sigma S_t dB_t)^2 = \sigma^2 S_t^2 dB_t^2 + 2r\sigma S_t^2 dB_t dt + r^2 S_t^2 dt^2 \tag{3.9}$$

In the limit $dt \to 0$, dt tends to zero faster than dB_t since we assume $dB_t^2 = \mathcal{O}(dt)$.

$$\implies ds_t^2 = \sigma^2 s_t^2 dt \tag{3.10}$$

Substituting ds_t and ds_t^2 in equation (3.8) we get:

$$df = rdt + \sigma dB_t - \frac{\sigma^2}{2}dt = \left(r - \frac{\sigma^2}{2}\right)dt + \sigma dB_t \tag{3.11}$$

[4]For notational simplicity we have taken this differential equation same as the SDE of GBM

Integrating the equation (3.11) and resubstituting $f(s,t) = log(s_t)$ we get

$$log(s_t) = log(s_0) + \left(r - \frac{\sigma^2}{2}\right)t + \sigma B_t \tag{3.12}$$

We finally arrive at the solution of GBM

$$s_t = s_0 \exp\left(\left(r - \frac{\sigma^2}{2}\right)t + \sigma B_t\right) \tag{3.13}$$

Interpreting the solution

Equation (3.13) is the final solution of SDE (3.4). Typically, SDEs do not have closed-form analytical solutions, but with the help of Ito's lemma, an analytical solution is possible in this case. In the equation (3.13) r denotes drift which controls the deterministic changes in the stock price and σ denotes the volatility which determines the stochastic changes in stock price. In the GBM model the performance of the stock is quantified using the log return. The log return for the time period Δt is given by:

$$log\left[\frac{s_{t+\Delta t}}{s_t}\right] = \left(r - \frac{\sigma^2}{2}\right)\Delta t + \sigma\varepsilon\sqrt{\Delta t} \tag{3.14}$$

Here we have replaced $B_t \sim \mathcal{N}(0,t)$ by $\varepsilon\sqrt{\Delta t}$, where ε is random number sampled from $\mathcal{N}(0,1)$.

Simulation description

We have simulated $N = 5$ stocks using the equation (3.13). Like previous experiments, we have coupled the stocks using an interaction (ER) network. The way we have coupled the stock is by coupling their log returns. In our simulation, the log return of a stock is influenced by the average log returns of its neighbouring stocks in the network, which in turn couple the time evolution of their prices. If the price and log return of stock i at time t is given by s_t^i and L_t^i[5] respectively, then our coupling principle translates to:

$$s_t^i = s_{t-1}^i \exp\left(L_t^i + \frac{1}{|\mathcal{N}_i|}\sum_{j\in\mathcal{N}_i} L_t^j\right) \tag{3.15}$$

[5]**Note**: Log returns L are calculated based on GBM model without any coupling.

Where $\mathcal{N}_i$ denotes the set of nodes (stocks) neighbouring node (stock) i. $|\mathcal{N}_i|$ denotes cardinality of the set $\mathcal{N}_i$.

For simulations we have sampled values of annual drift r, annual volatility σ and initial stock price s_0 uniformly at random from [1%, 10%),[1%, 30%) and [80, 90) respectively. For the purpose of training we have used 'CNN' Encoder and 'RNN' Decoder with hidden dimmension equal to 384. We have chosen prediction steps $M = 5$ and softmax temperature $\tau = 0.7$. The input to our model are feature vectors assosciated with each stock which are created by concatenating the stock price timeseries, log returns, drift and volitility.

We randomly couple the pair of stocks with a probability of 0.5. This can be viewed as creating an ER network $G(n, p = 0.5)$. Next, we calculated price time series and log returns based on the coupling principle (3.15). To create a comprehensive feature of each stock, we concatenated these price time series and log returns with drift and volatility values[6]. These resulting feature vectors were then used as our model input.

Motivation behind the coupling principle

The assumption that log returns of stocks are coupled is based on evidence of correlation or dependence among returns of different stocks in financial markets. This is because stocks in the same industry may be affected by common macroeconomic factors or similar news and geopolitical events, leading to correlated price movements.

This assumption is supported by empirical evidence that generally, when the returns of one stock are high (or low), the returns of other stocks (belonging to the same sector) tend to be high (or low) as well. This is often called a spillover effect in finance.

Log returns are preferred over simple returns for the assumption of coupling between stock returns due to their approximately normal distribution, additivity, and suitability for measuring percentage changes in asset prices. These properties make log returns more appropriate for statistical analysis, modelling, and forecasting.

[6]Drift and volatility here are not the same as used for generating stock data using equation (3.13) but are estimated from price time series and log returns.

Chapter 4

Results and Discussions

4.1 Results

We have evaluated our model on two different sets of tasks: interaction network recovery and trajectory forecasting. Performance on interaction network recovery tasks is quantified by accuracy in predicting interaction between the pair of agents, and performance in trajectory forecasting tasks is measured using the Mean Squared Error (MSE) in predicting the system's future state for T timesteps.

Our training set comprises 50,000 simulations for all tasks mentioned in tables 4.1, 4.2 and 4.3, and training is carried out for 500 epochs except in the task 8 (5 stocks) where performance seems to saturate before 100 epochs. Therefore, we trained this particular task for 140 epochs.

To assess the performance of our model, we conducted testing on a set of 10,000 simulations for each task except task 9 and 10 where we used 2000 test simulations. The metrics presented in Tables 4.1,4.2 and 4.3 were calculated based on the test performance. By using different initialisation[1], we get slightly different performances on the interaction network recovery tasks[2]; hence we have used three different initialisations to evaluate our model performance (3rd column of tables 4.1 and 4.2).

[1] Different seed to initialise the weights of our model

[2] performance does not vary much for trajectory forecasting tasks

Task 1 and task 2 consist of 5 and 10 interacting particles, respectively, where each pair of particles are either interacting, i.e. spring constant $K = 1$ or not interacting $K = 0$ with equal probability. Task 3 consists of 5 interacting particles where interactions can be of type $K = 0$, $K = 0.5$ or $K = 1$ with equal probability. Task 4 to 7 consists of a system of 5 or 10 interacting oscillators which are interacting or not interacting with equal probability. In tasks 4 and 6 actual value of intrinsic frequency (ω) is given as one of the model inputs. In tasks 5 and 7 estimated value of ω is given in place of the actual value. Task 8 consists of 5 stocks where the log return of each stock pair is either coupled or not coupled with equal probability.

Task No.	**Tasks**	**% Accuracy for Different Initialisations**	**Mean % Accuracy**
1	5 Particles, 2 link types	99.861	99.868 ± 0.010
		99.864	
		99.879	
2	10 Particles, 2 link types	98.399	98.434 ± 0.034
		98.467	
		98.435	
3	5 Particles, 3 link types	99.182	99.171 ± 0.014
		99.177	
		99.155	
4	5 Oscillators (Actual omega)	96.381	96.340 ± 0.041
		96.299	
		96.341	
5	5 Oscillators (Estimated omega)	95.173	95.270 ± 0.204
		95.504	
		95.132	
6	10 Oscillators (Actual omega)	75.111	74.648 ± 0.435
		74.248	
		74.585	
7	10 Oscillators (Estimated omega)	74.341	72.356 ± 2.678
		69.310	
		73.416	
8	5 stocks	89.209	89.489 ± 1.584
		91.195	
		88.063	

Table 4.1: Performance of our model on interaction network recovery tasks.

Task No.	Tasks	% Accuracy for Different Initialisations	Mean % Accuracy
9	5 Particles, 2 link types (Sparse)	99.175	97.911 ± 1.951
		95.664	
		98.895	
10	5 Particles, 2 link types (Dense)	99.800	99.785 ± 0.081
		99.858	
		99.698	

Table 4.2: Extension of task 1

Task No.	Tasks	MSE for 'T' Time-steps	
		T = 10	T = 20
1	5 particles, 2 link types	1.10×10^{-6}	4.58×10^{-6}
2	10 particles, 2 link types	1.65×10^{-5}	5.14×10^{-5}
3	5 particles, 3 link types	2.93×10^{-6}	1.06×10^{-5}
4	5 Oscillators (Actual omega)	1.84×10^{-3}	4.26×10^{-3}
5	5 Oscillators (Estimated omega)	1.99×10^{-3}	5.24×10^{-3}
6	10 Oscillators (Actual omega)	7.72×10^{-3}	2.22×10^{-2}
7	10 Oscillators (Estimated omega)	8.00×10^{-3}	2.89×10^{-2}
8	5 stocks	2.11×10^{-3}	2.12×10^{-3}

Table 4.3: Performance of our model on trajectory forecasting tasks.

4.2 Discussions

In task 1 we are getting accuracy very close to 100% with very low standard deviation. In task 2 which is similar to task 1 but with double the number of particles (10) we see a drop in accuracy by nearly 1%. In task 3 which involve one extra link type as compared to task 1 we see small decrease in accuracy.

Tasks 4-7 involve the system of phase-coupled oscillators (Kuramoto Model). In Task 4 and 6, we used the actual ω values as one of the model inputs, which allowed for a high degree of accuracy in our results. However, in Task 5 and 7, we explored a more practical approach by providing estimated ω values instead of the actual ones. This resulted in a slight drop in accuracy, but it is a more realistic scenario as the actual values of ω are often not accessible in real-world situations. Snce the drop in accuracy is not significant, It shows the **robustness** of our model.

In task 8, we applied our model to a financial dynamical system of interacting stocks. This is our novel extension to the applicability of our model. We see accuracy very close to 90%. Although the accuracy we achieved in Task 8 is slightly lower than other tasks involving 5 interacting agents, it is still impressive considering the inherent stochastic nature[3] of the dynamical system we are dealing with.

In table 4.2, we have evaluated our model trained in task 1 on some additional tasks. The test set used for evaluation consists of 2000 simulations. Here the model is evaluated at two extreme limits: first, when the interaction network governing the dynamics is sparse ($G(n=5,p=0)$)[4] and second, when the interaction network is dense ($G(n=5,p=1)$)[5]. In both regimes, we see a decrease in accuracy compared to task 1. As we have used a model trained on the simulations where the particles are interacting or not interacting with probability 0.5, i.e. the interaction network is of the form $G(n=5,p=0.5)$, our model may have developed some bias towards the set of simulations used for training. However, the bias towards the training set seems to be low as the drop in accuracy is relatively low; this shows the **generalizability** of our model. It is also worth noting that the drop in accuracy is more in the sparse regime than the dense one. It indicates that the model is more inclined towards predicting the presence of a link in the interaction network rather than its absence.

Table 4.3 shows our model's performance in predicting the dynamical systems' future state

[3]The system of interacting stock is governed by Stochastic Dynamical Equation (3.4)
[4]Each particle is independent/ uncoupled
[5]Every particle is coupled to every other particle with a spring of spring constant $K=1$

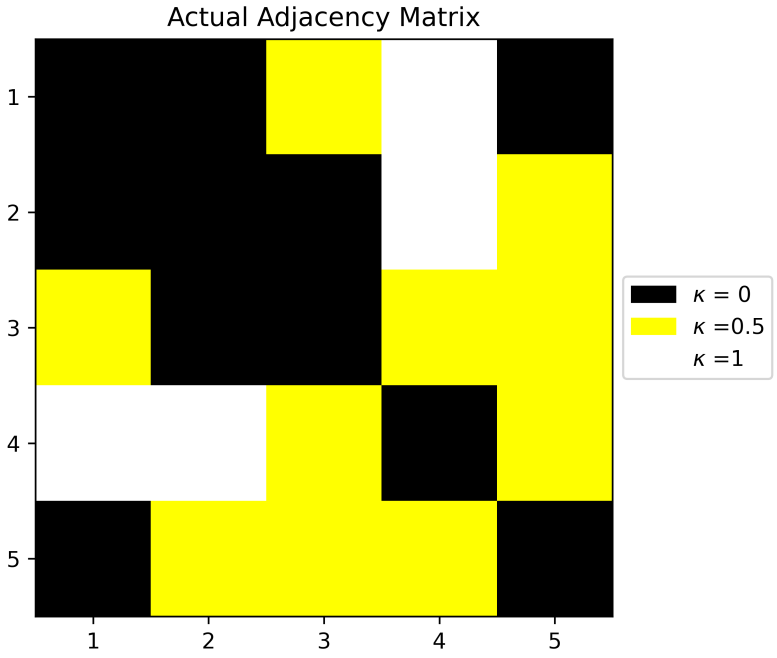
for 10 and 20 timesteps in future. It is very clear from the table that the MSE is relatively larger for $T = 20$ than $T = 10$. This is expected since predicting more steps in future will accumulate more errors and hence will have a larger MSE. Overall the MSE seems to be less for the tasks involving the system of interacting particles than the system of oscillators and stocks. This may be because the system of oscillators and stocks shows much more complex and rich dynamics than the particles interacting via Hooke's law.

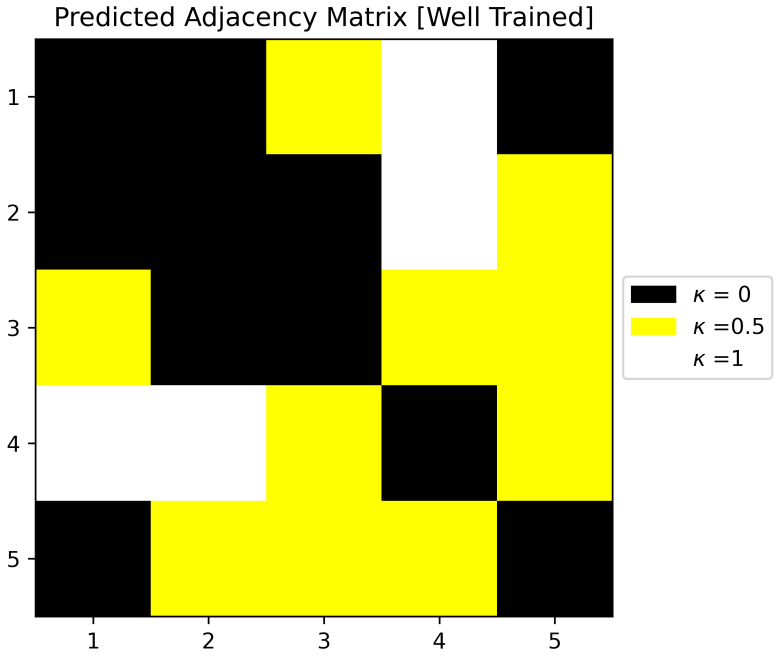
In the tasks involving the system of oscillators, the MSE is lower for the tasks where actual values of ω are given compared to the estimated one. This outcome was anticipated as having precise inputs facilitates better learning of the system's dynamics. It is also interesting to note that in the case of interacting stocks, the MSE for predicting 10 timesteps is almost similar to the MSE for predicting 20 timesteps. This finding can be attributed to the inherently stochastic nature of the system of stocks, which makes predicting 10 timesteps into the future as challenging as predicting 20 timesteps.

4.3 Visualisations

4.3.1 Model's performance with training

Here we will visualise the results of task 3 on a simulation from the test set to show how model performance improves with training on both fronts: interaction network recovery and trajectory forecasting. We will take two models, one which is trained for 10 epochs [suboptimally trained] and the other for all 500 epochs [well trained]. Both models have been trained on 50,000 training simulations. As expected, we can see performance improvement in both interaction network recovery (Figure 4.1) and trajectory forecasting tasks (Figures 4.3 and 4.2).


Actual Adjacency Matrix
1
2
3
4
5
1
2
3
4
5
κ = 0
κ =0.5
κ =1


Predicted Adjacency Matrix [Well Trained]
1
2
3
4
5
1
2
3
4
5
κ = 0
κ =0.5
κ =1

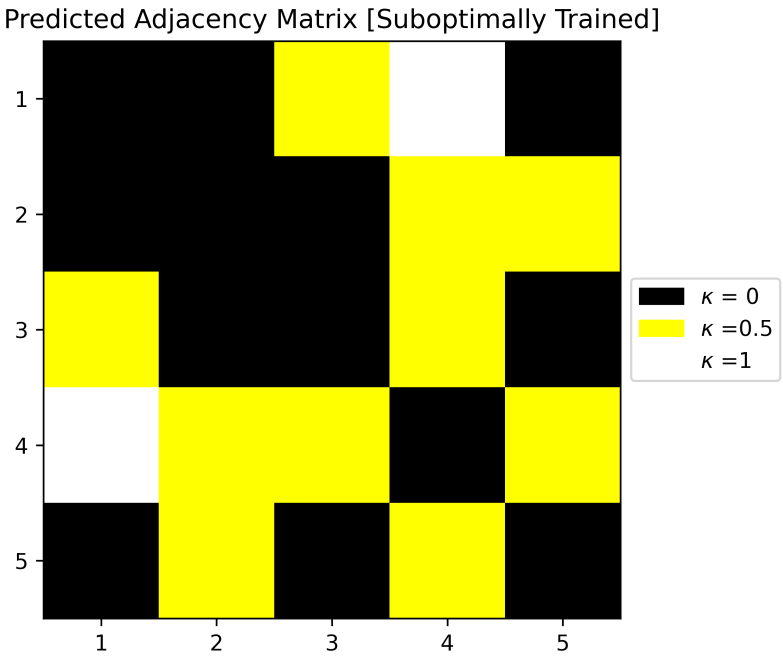


Figure 4.1: Performance comparison of **well trained** Vs **suboptimally trained** model in recovering the interaction network in task 3. The numbers displayed on the axes of the adjacency matrices denote the particle id. White, yellow and black indicates the particles are coupled by a link of type $K = 1$, $K = 0.5$ and $K = 0$ respectively.

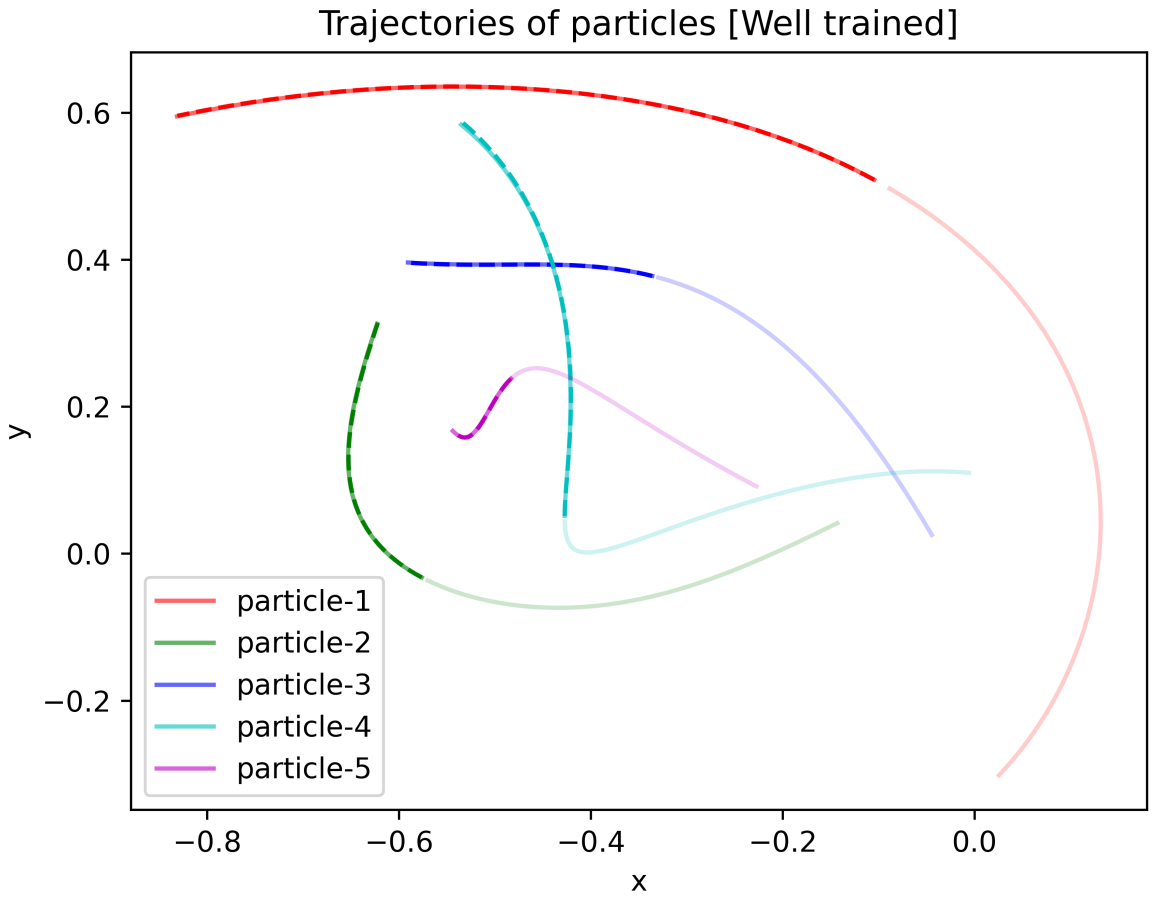


Figure 4.2: Performance of **well trained** model in forecasting the trajectory in task 3. The faint part of the trajectory corresponds to the first 49 timesteps given as input to the encoder, and the solid part of the trajectory denotes the 40 future timesteps predicted by the decoder. The dashed part indicates the actual trajectory for 40 future timesteps.

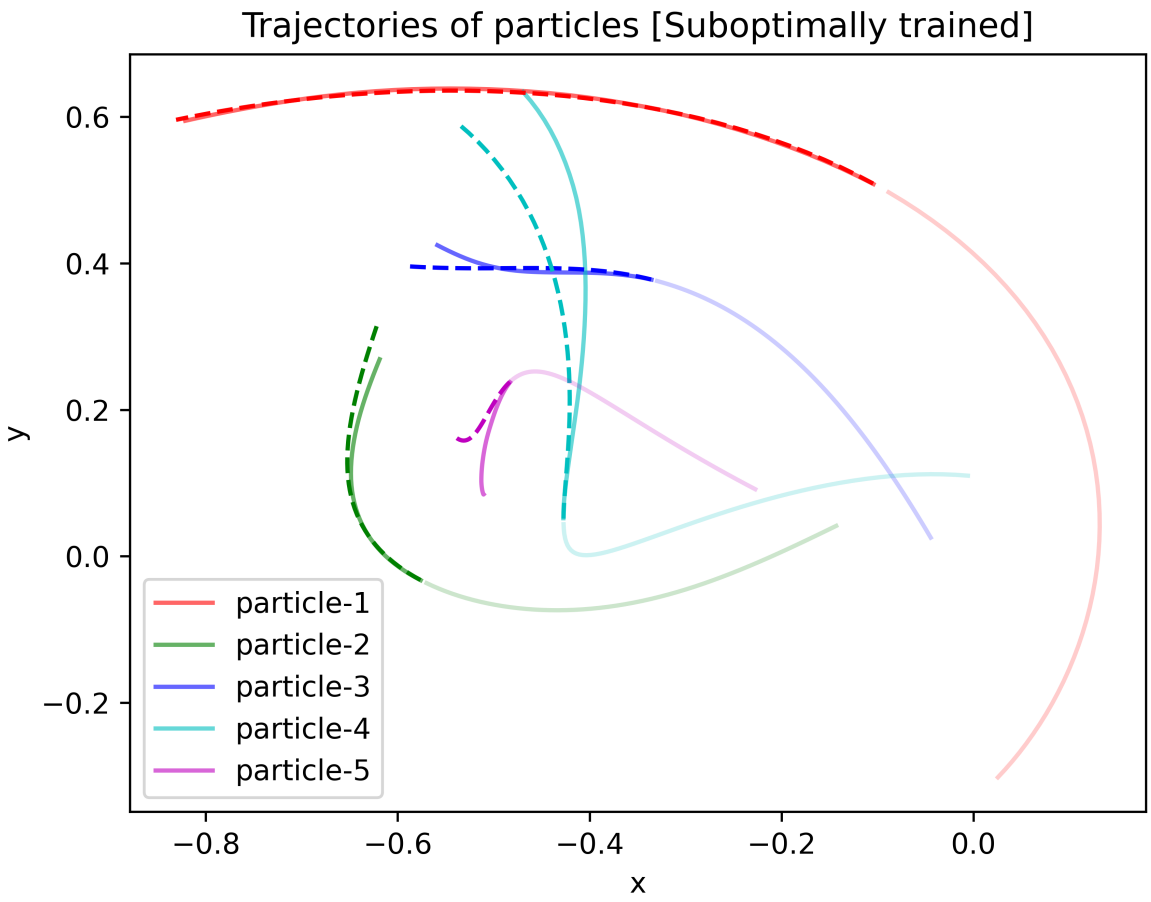
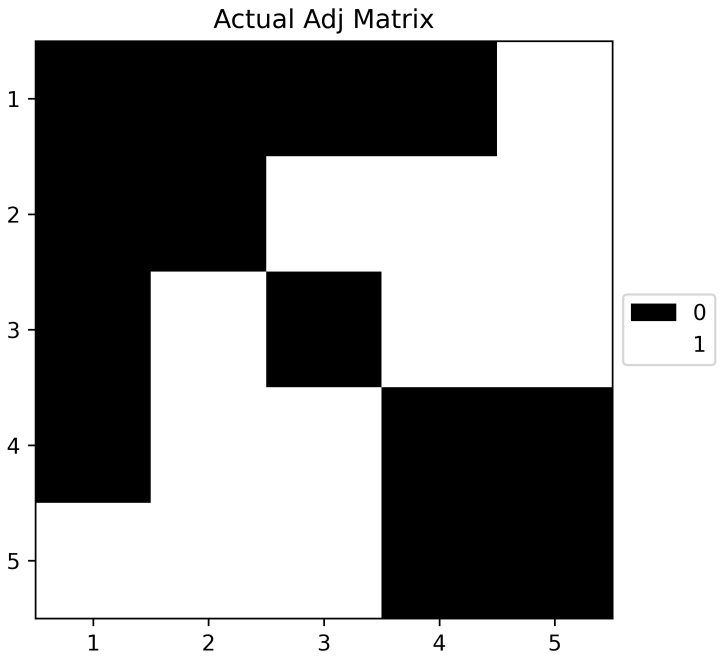
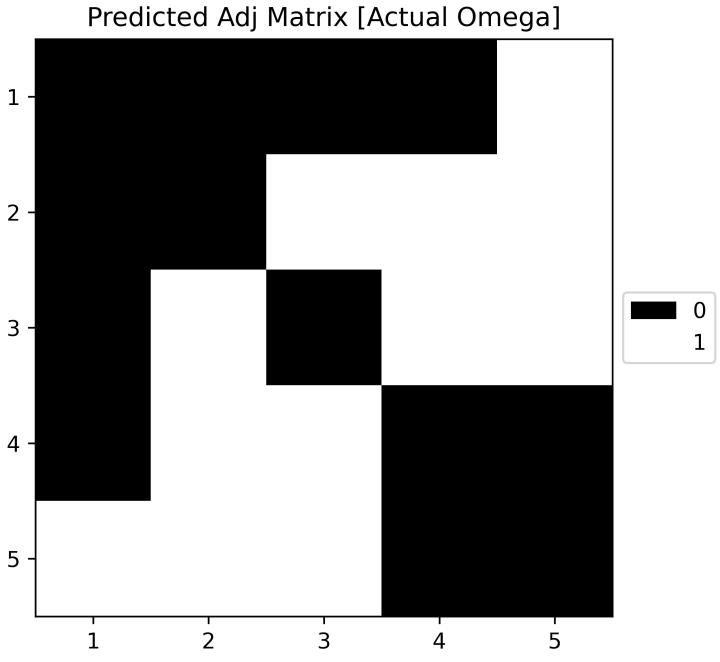


Figure 4.3: Performance of **suboptimally trained** model in forecasting the trajectory in task 3. The faint part of the trajectory corresponds to the first 49 timesteps given as input to the encoder, and the solid part of the trajectory denotes the 40 future timesteps predicted by the decoder. The dashed part indicates the actual trajectory for 40 future timesteps.

4.3.2 Interacting Oscillators - Kuramoto coupling

Here we will compare the performance of our model trained in task 4 [Actual Omega] and task 5 [Estimated omega] on a simulation from test set. Both models were able to infer the exact interaction network (refer Figure 4.4). In the trajectory forecasting task, both models were pretty close to the ground truth trajectory (refer Figure 4.5).


Actual Adj Matrix
1
2
3
4
5
1
2
3
4
5
0
1


Predicted Adj Matrix [Actual Omega]
1
2
3
4
5
1
2
3
4
5
0
1

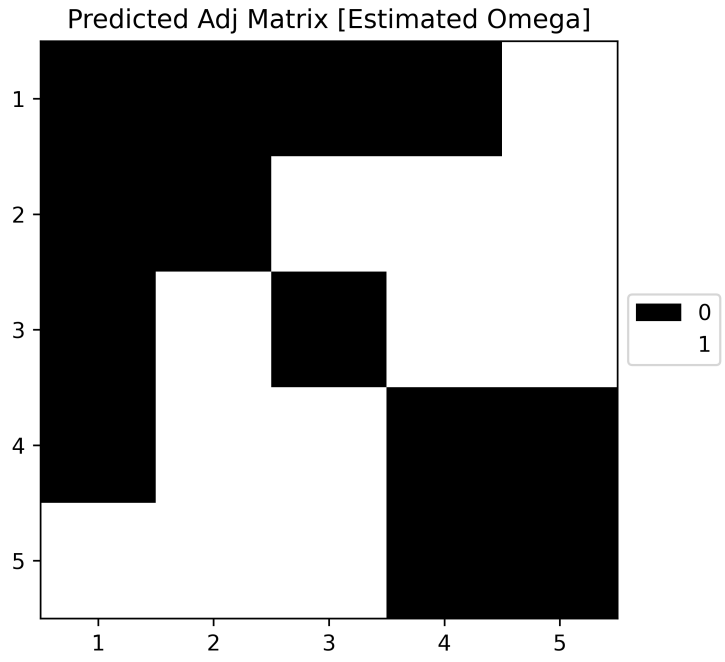


Figure 4.4: Performance comparison of the model trained in task 4 [**Actual Omega**] Vs model trained in task 5 [**Estimated omega**] in recovering the interaction network. The numbers displayed on the axes of the adjacency matrices denote the oscillator id. White indicates oscillators are coupled i.e. $A_{ij} = 1$, and black indicates they are not i.e. $A_{ij} = 0$.

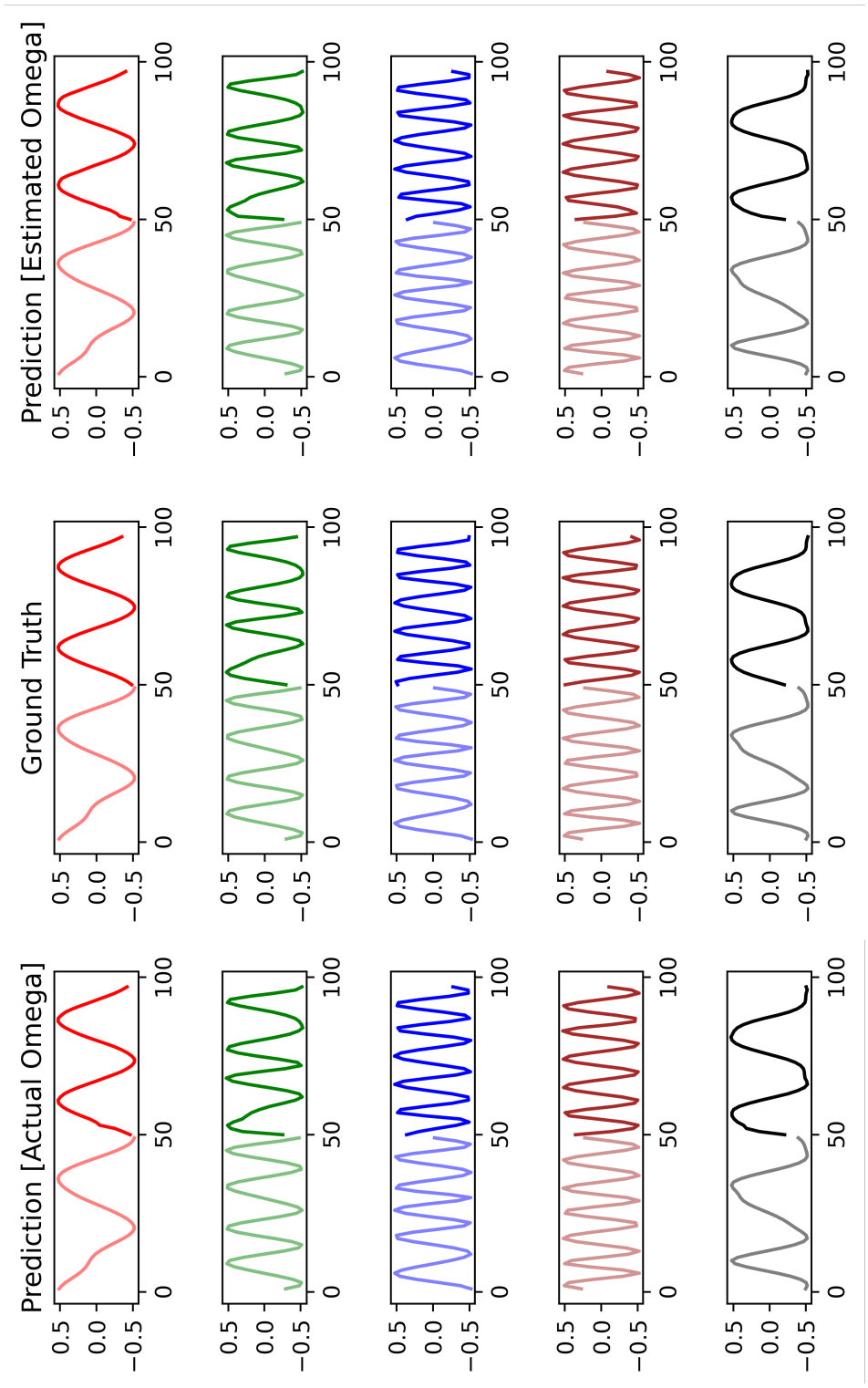


Figure 4.5: Performance comparison of the model trained in task 4 [**Actual omega**] Vs model trained in task 5 [**Estimated omega**] in forecasting the phase of oscillators. Faint trajectories denote the first 49 timesteps given as model input, while solid trajectories denote the model predictions for the next 49 timesteps.

4.3.3 Interacting stocks - GBM model

Here we visualise the performance of the model trained in task 8 in interaction network recovery (refer Figure 4.6) as well as trajectory (log returns) forecasting (refer Figures 4.7 to 4.11).

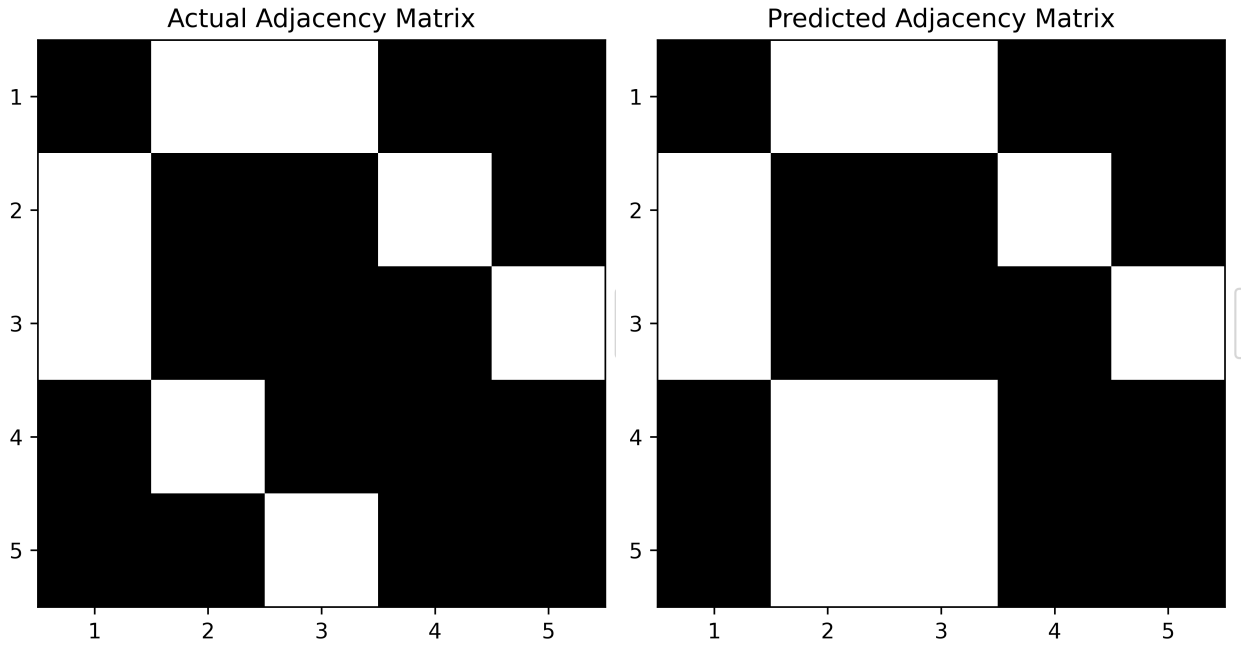


Figure 4.6: Performance of the model trained in task 8 in recovering the interaction network. The numbers displayed on the axes of the adjacency matrices denote the stock id. White indicates stocks are coupled, and black indicates they are not.

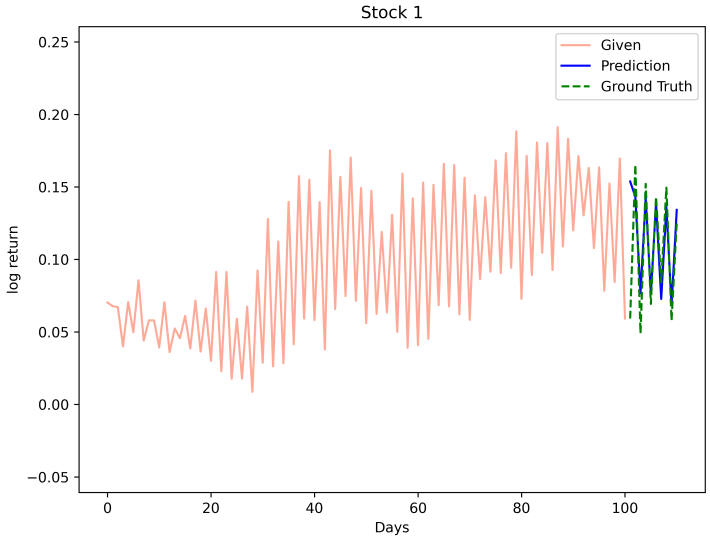


Figure 4.7: Performance of model trained in task 8 in predicting future log returns of stock 1. The red time series (corresponding to the first 99 days) is given to the model, and the blue is the time series (corresponding to the next 10 days) predicted by the model. The dashed green time series (corresponding to the next 10 days) is the actual continuation of the red time series.

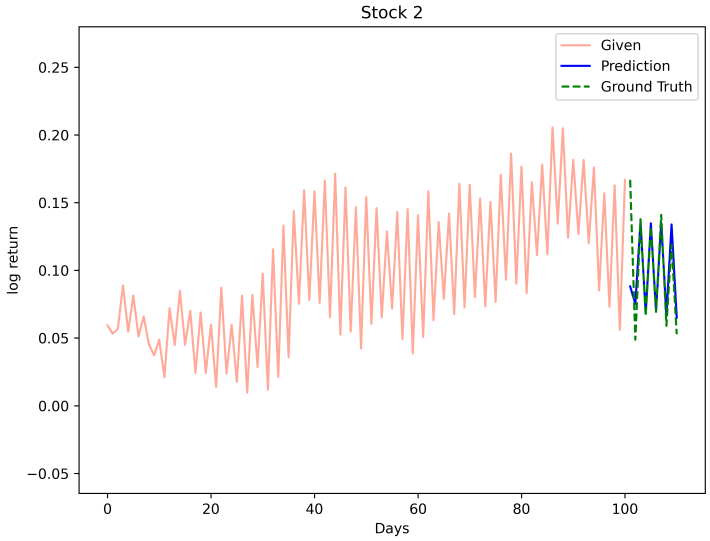


Figure 4.8: Performance of model trained in task 8 in predicting future log returns of stock 2. The red time series (corresponding to the first 99 days) is given to the model, and the blue is the time series (corresponding to the next 10 days) predicted by the model. The dashed green time series (corresponding to the next 10 days) is the actual continuation of the red time series.

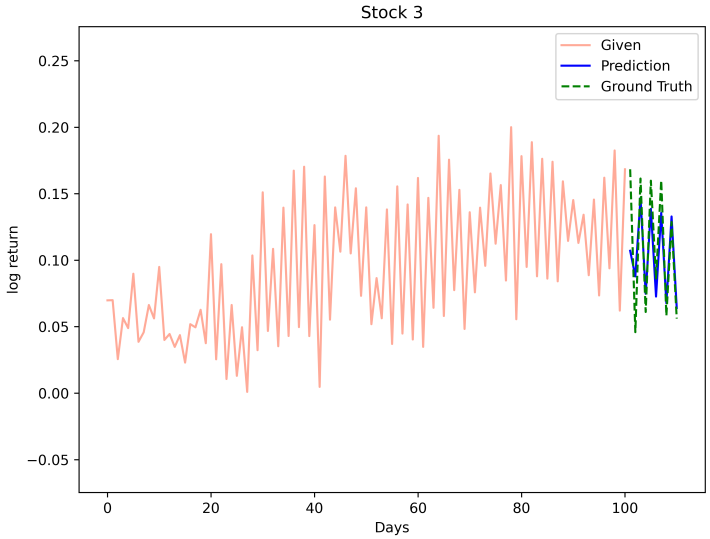


Figure 4.9: Performance of model trained in task 8 in predicting future log returns of stock 3. The red time series (corresponding to the first 99 days) is given to the model, and the blue is the time series (corresponding to the next 10 days) predicted by the model. The dashed green time series (corresponding to the next 10 days) is the actual continuation of the red time series.

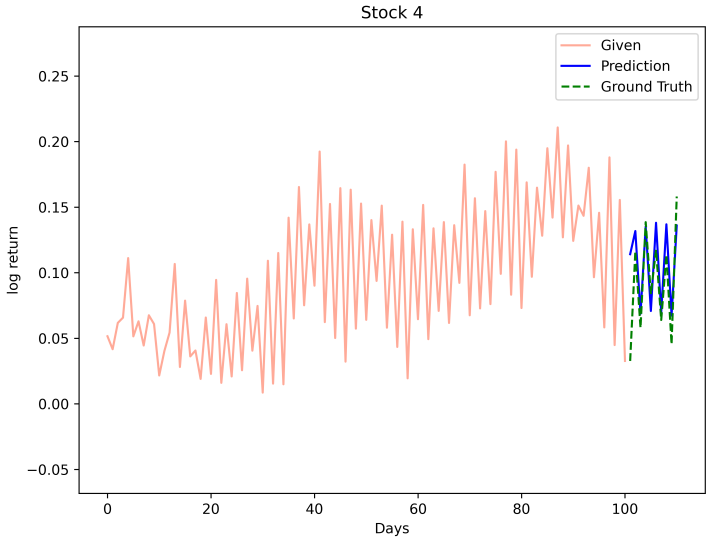


Figure 4.10: Performance of model trained in task 8 in predicting future log returns of stock 4. The red time series (corresponding to the first 99 days) is given to the model, and the blue is the time series (corresponding to the next 10 days) predicted by the model. The dashed green time series (corresponding to the next 10 days) is the actual continuation of the red time series.

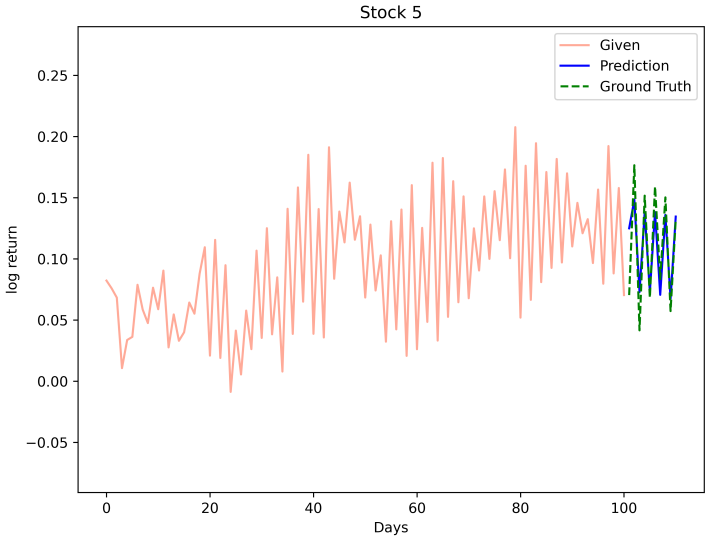


Figure 4.11: Performance of model trained in task 8 in predicting future log returns of stock 5. The red time series (corresponding to the first 99 days) is given to the model, and the blue is the time series (corresponding to the next 10 days) predicted by the model. The dashed green time series (corresponding to the next 10 days) is the actual continuation of the red time series.

Chapter 5

Conclusion and Outlook

In this thesis, we used Neural Relational Inference, a graph neural network-based self-supervised learning model, to infer various dynamical systems' relational structures and dynamics. Our model has performed very well in various tasks pertaining to different dynamical systems. Since our model does not make any assumptions regarding the dynamical system on which it is applied, this shows our model's versatility.

We have also shown that instead of giving actual omega (which is often inaccessible) as one of the model inputs in tasks related to the system of Kuramoto oscillators, we can give the estimated omega values without compromising much on the accuracy. In this way, we increased the practicality of our model. We also extended the applicability of the NRI model by successfully applying it to the inherently stochastic dynamical system of interacting stocks.

There is room for significant improvement as Interaction Network Inference is still in its early stage of development. The NRI model is Less effective for large Networks since the resource and time complexity of the model is $\mathcal{O}(|\mathcal{V}|^2)$[1]. The NRI model starts with a fully connected network and discovers the sparsity gradually. This is a bit unnecessary since most real-world networks are sparse.

There is further scope for improvements in the NRI model, like extending it to the cases where the interaction network itself is time-dependent. Applying it to the extensive stochastic dynamical system will also be a challenge that can be explored in future works.

[1] $\mathcal{V}$ denotes the set of nodes in the interaction network.

Bibliography

[1] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, page 37–48, New York, NY, USA, 2013. Association for Computing Machinery.

[2] Louis Bachelier. The theory of speculation [théorie de la spéculation]. In Paul Cootner, editor, *The Random Character of Stock Market Prices*, pages 17–78. MIT Press, 1964.

[3] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics, 2016.

[4] Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–654, 1973.

[5] Michael B. Chang, Tomer Ullman, Antonio Torralba, and Joshua B. Tenenbaum. A compositional object-based approach to learning physical dynamics, 2017.

[6] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[7] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[8] J.R. Dormand and P.J. Prince. A family of embedded runge-kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.

[9] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry, 2017.

[10] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 2:729–734 vol. 2, 2005.

[11] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks, 2016.

[12] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Li Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks, 2017.

[13] Nicholas Guttenberg, Nathaniel Virgo, Olaf Witkowski, Hidetoshi Aoki, and Ryota Kanai. Permutation-equivariant neural networks applied to dynamics prediction, 2016.

[14] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric numerical integration*, volume 31 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, second edition, 2006. Structure-preserving algorithms for ordinary differential equations.

[15] William L. Hamilton. *Graph Representation Learning*. Morgan & Claypool Publishers, 2020.

[16] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[18] L. C. Jain and L. R. Medsker. *Recurrent Neural Networks: Design and Applications*. CRC Press, Inc., USA, 1st edition, 1999.

[19] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2017.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[21] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[22] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems, 2018.

[23] Thomas N. Kipf and Max Welling. Variational graph auto-encoders, 2016.

[24] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[25] Yoshiki Kuramoto. Self-entrainment of a population of coupled non-linear oscillators. In Huzihiro Araki, editor, *International Symposium on Mathematical Problems in Theoretical Physics*, pages 420–422, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg.

[26] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding, 2017.

[27] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables, 2017.

[28] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, aug 2014.

[29] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models, 2014.

[30] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[31] Jonathon Shlens. Notes on kullback-leibler divergence and likelihood, 2014.

[32] Steven E Shreve. *Stochastic Calculus for Finance I: The Binomial Asset Pricing Model*. Springer Science & Business Media, 2004.