Saarland University
Department of Computer Science
Data Science and Artificial Intelligence

Master Thesis

Network Reconstruction Using Deep Learning and Sensitivity Analysis

Author Joshgun Guliyev

Supervisor Univ.-Prof. Dr. Verena Wolf

Advisor Dr. Gerrit Großmann

1. Reviewer Univ.-Prof. Dr. Verena Wolf
Department of Computer Science
Saarland University

2. Reviewer Univ.-Prof. Dr. Holger Hermanns
Department of Computer Science
Saarland University

Submitted on May 4, 2023

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I consent to both versions of my thesis being added to the library of the Computer Science Department, making them accessible to the public.

Saarbrücken, _____ _____

Abstract

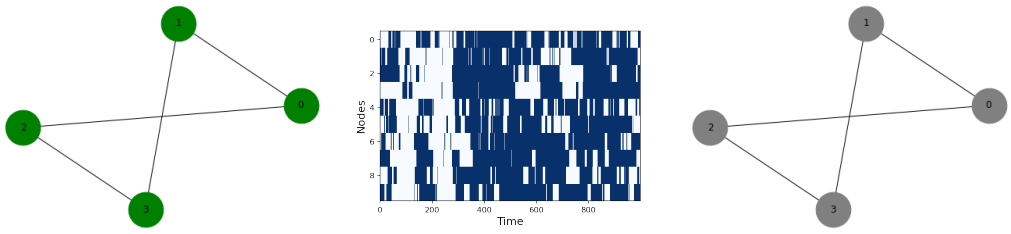
There are many natural processes that can be represented as dynamic systems, and the components of these systems are connected to each other with some underlying network structure. Understanding the behavior of such systems requires knowledge of the underlying interactions and network structure, which is often difficult to observe directly. In this thesis, we use deep learning to address the network reconstruction problem. Without relying on many parametric assumptions about the network structure and dynamics. We train node-specific MLP (multi-layer perceptron) models to learn the behavior of each node. Assuming that the direct neighbors of each node have the most impact on its future behavior. Using sensitivity analysis, we identify the most important nodes for each node and reconstruct the network without any ground truth information. Unlike the many traditional (statistical) network reconstruction methods, our approach can reconstruct networks with non-linear dynamics, find edges between heterogenous nodes (nodes do not behave similarly), can distinguish direct and indirect effects in sufficient amounts. We tested our method on various dynamical models and networks, with high accuracy. Despite the fact that our approach successfully reconstructs networks with high accuracy in small and medium-sized networks, scalability and identifying much more complex interactions in larger networks is the main limitation. Overall, this work aims to present a suitable alternative to traditional statistical methods for network reconstruction by focusing on their main limitations.

Contents

1 Introduction

1.1 Network reconstruction problem

There are many intricate systems around the world that consist of multiple components which work together according to underlying dynamical rules. These components are often connected or related to each other through unknown structural connectivity, creating an underlying network structure of the system. Therefore, any system can be thought of as a network of its interacting components. For instance, a network of brain cells in the human body, a social media network of individuals, a network of atmospheric components, and networks of industrial operations can serve as examples of this topic. In most cases, the dynamic rules governing these systems are neither known nor observable. This represents one of the most significant obstacles when it comes to predicting future system behaviors, and solving or avoiding potential problems. Examining the dynamical rules and structures of "black box" systems can provide valuable opportunities to resolve potential problems, interpret systems, or even develop similar systems automatically [55]. For instance, this may include limiting the spread of viruses in networks, identifying the source of fake news on social media, predicting weather patterns, controlling the spread of epidemics, optimizing industrial operations, and so forth. Despite the numerous benefits of understanding the dynamic and structural principles of these systems, in many real-world scenarios, only data generated by the system during a specific time window is observable. Inferring the dynamical rules and structural connectivity of system components based on their historical behaviors is a far more critical topic [55], which is called network reconstruction.



(a) Ground truth graph **(b)** Observed time series **(c)** Reconstructed graph

Fig. 1.1.: Example of sample NR process

1.1.1 Applications

As mentioned above, NR (network reconstruction) is the process of inferring the underlying network structure from observed data. This can be applied to a wide range of systems, including biological, social, and technological networks. By using techniques from graph theory and statistical analysis, or machine learning methods, network reconstruction can help researchers better understand the dynamics of complex systems and make predictions about their behavior. And by this way, network reconstruction can provide valuable insights into the organization and function of complex systems. As mentioned in [55], many researchers in this field have focused on how the topology of a graph can be used to infer the dynamical behaviors of a system, but in real cases, we have only access to the observed time-series data as mentioned before. Consequently, inferring network structure and underlying relations based on observed data is a quite important and efficient process to solve many complex problems. As an example of the applications of network reconstruction in biology, researchers have proposed a method that allows for the study of the way genetic networks work and for finding the molecular targets of drugs in a way that can be easily tested and simulated using computers or reconstruction of gene regulatory networks by using expression data [26]. Also in climate science, the importance of understanding the patterns and mechanisms behind atmospheric teleconnections (long-distance connections between different parts of the Earth's atmosphere) for predicting extreme weather events, particularly in the context of human-caused climate change is very crucial for weather forecasting in general, but especially for predicting extreme events, which may be impacted by climate change [9]. In social science, network reconstruction is used to study social and communication networks, such as friendship networks, email networks, and social media networks. By reconstructing the network structure, patterns of communication, and social interaction within the network and how these patterns influence the behavior of individuals within the network can be understood in a better and more accurate way. Overall, network reconstruction is a powerful tool for understanding and predicting the behavior of complex systems, and it has a wide range of applications in various fields.

1.1.2 Objective, hypothesis and methodical outline

We have presented several applications and benefits of network reconstruction in diverse fields, from biology to social sciences, which have highlighted the significance of network inference. In this thesis, we endeavor to address these challenges and

reveal the underlying network structures without any parametric assumptions and prior knowledge about the ground truth function that governs node evolution. To accomplish this goal, we propose to utilize the functional relationships among the components of a system, which we will derive from time series data generated by an unknown network with unknown rules over a specific time period. We plan to identify crucial relationships between nodes (components) based on node-specific MLPs (multi-layer perceptron), which will be the key element of our methodology in this work. Our work aims to be independent of the specific dynamical rules that generate the data, making it applicable to various data types, including binary, discrete, and continuous state spaces, and to be a convenient alternative to traditional methods used in network reconstruction problems. In the related works section, we will present some examples of both model-dependent and model-free approaches that have been applied in this field. In addition to network inference, we also endeavor to learn the dynamic behaviors of the agents in a system, enabling us to make predictions about their future states. While many network inference methods assume that all components of a system behave similarly, our approach analyzes each agent individually to infer possible differences between them, which is often observed in real-world scenarios such as atmospheric components or stock markets. Specifically, we hypothesize that a node's direct neighbors have more influence on its future behaviors than other components without direct structural connections. We will reconstruct networks relying on this assumption by interpreting the effects of the network's components on each other and classifying them as either a neighbor or not based on the measured effects. We will train deep learning models to predict the state of a specific component of the system at the next time step using the historical information (states) of other nodes at the previous time step. Then, we will identify the most important features of the trained model that contribute significantly more to the prediction than others by sensitivity analysis methods. These features will be used to determine which nodes are the predicted node's neighbors. After accurate analysis and decision-making, we will reconstruct the network. We will train an independent model for each agent, allowing us to learn the agent-specific effects of the dynamical rules accurately. This will also help us to identify the pure impacts of other agents on a particular node more clearly and accurately. Briefly, we will combine different methods for network inference from the prediction of node states with deep learning, interpretation of deep learning models, and classifying those interpreted effects with unsupervised learning methods to learn the functional connectivity and infer structural connectivity based on learned functional relationships or interactions. Although our method will depend on dynamics prediction, we will mainly focus on the quality of the reconstructed network structure rather than the prediction accuracy of the agents' future states. For instance, we will not use the

agent's own previous information in the prediction to quantify other agents' effects on the target agent more clearly, leading to weaker dynamics prediction accuracy than using the agent's own information. Instead, we will try to make as accurate a prediction as possible without using the agent's own information and reconstruct the network based on this prediction. This will be explained in more detail in the following sections. For the identification of the most significant features from trained predictors, we will employ three sensitivity analysis methods to interpret and extract the importance of each agent. These methods will be explained after the related works section.

1.1.3 Structure of the thesis

The next sections of the thesis and the overall structure will be as shown below:

- In Chapter 2, we will provide the necessary background on the topic and present an overview of the problem statement. We will review related works that have been conducted in the domain of network reconstruction, as well as discuss information about the dynamical models that have been used for data generation and experiments.
- In Chapter 3, we will present our proposed methodology, which includes our overall approach, prediction methods, and model architectures. We will also discuss the sensitivity analysis methods used to extract future importance, as well as techniques for converting resulting weighted and directed adjacency matrices to symmetric and undirected matrices. Additionally, we will cover the process of neighbor selection based on importance scores and the network reconstruction process.
- In Chapter 4, we will present and discuss the experimental results of our method. We will demonstrate the effectiveness of our approach on various settings, such as different graph types, varying numbers of nodes and edges, and diverse dynamical models. Additionally, we will compare our method with various statistical baselines, and present the results of these comparisons.
- In Chapter 5, we will provide a thorough discussion of our method, results, and their advantages and disadvantages. We will highlight the strengths and limitations of our approach and compare it with other existing methods. Finally, we will suggest some future research directions in the related topic to further enhance the performance and applicability of our proposed method.
- Chapter 6 is the final chapter of the thesis with short summary and conclusion.

2 Background

2.1 Notations

In this work, we represent networks as graphs that consist of a different number of components (nodes), and those nodes are connected with edges. We will notate the graph as $G = (V, E)$ where $V = \{1, \ldots, N\}$ is a set of nodes (vertices), where N is the number of present nodes in the given graph G and $E \subseteq V \times V$ is a set of edges. We will use undirected and binary graphs in our experiments which means that the adjacency matrix of these graphs is symmetric and the notation of a true adjacency matrix of the given graph will be $A^* \in \{0, 1\}^{N \times N}$. The predicted symmetric adjacency matrix will be notated as A, and the weighted adjacency matrix will be notated as $\hat{A}$. We should find a A that satisfies $A = A^T$ condition which means it is symmetric. 0 valued entries of the A means there is no direct connection (edge) between node i and j as $A_{ij} = 1$, and 1 means there is which can be written as $A_{ij} = 1$. And the nodes which are connected are called direct neighbors of each other and $N(i) := \{j \in V \mid (i, j) \in E\}$ is called a direct neighborhood of node i. As we mentioned in the above sections, we will use time series data of nodes' states in our prediction and we define these time series as $X := \left(X^t, \ldots, X^T\right)$ which contains T time steps for each node from i to j where $X^t := (X_1^t, \ldots, X_N^t)$ is 't' -th time step of N nodes, and $X_i := \left(X_i^t, \ldots, X_i^T\right)$ is T time steps of node i. X_{-i} indicates the time series data without node i and we notate a single observation of X_{-i} as x while explaining sensitivity analysis methods for simplicity. Values of this time series can be binary, discrete, or continuous, depending on the respective dynamical model's characteristics. Additionally, we denote the random variable in the case of the stochastic process as $\mathbf{X}_i^t$ for node i at time step t. Those are the main notations used in the thesis, but specific case-related notations for specific parameters and functions will be clarified when it is mentioned in the next sections.

2.2 Related works

Researchers have recently applied various approaches to the network reconstruction problem domain. In this section, we will discuss some of these approaches to provide more comprehensive information about the problem domain and solutions and related works to our approach. These approaches can be categorized based on their application methods, advantages, and disadvantages.

Network reconstruction methods can be divided into two groups based on their application methods and requirements: model-based and model-free methods. Both methods have specific rules and assumptions for the network reconstruction problem. However, model-free approaches have fewer assumptions, such as linearity, which will be discussed in more detail after model-based approaches in the following sections.

2.2.1 Model-based approaches

In network reconstruction, model-based methods refer to approaches that rely on a mathematical model or set of assumptions about the underlying network to infer the network structure. These methods typically depend on the specific dynamical model or assume a function that generates the data, and by defining the parameters of this function, relationships between nodes are identified [6]. The general framework of the model-based approaches in network inference in many fields, from climate science to neuroscience, is defining a generative function and fitting this function to the data by finding the optimal parameters of the function and reconstructing the network structure using the information in the fitted function. While model-based methods can be successful in theory, they do not always reveal the same results in real-world problems, as in many real-world problems interactions between the components of the system are not simple(such as linear) or a single function is not enough to represent the whole interactions where there are a lot of other factors that affect the interactions of nodes, and these factors fall outside of the assumed generative function [51].

The most popular kinds of model-based inference methods are auto-regressive models which assume a linear function to fit the time series data. These methods are not able to cover non-linear relationships and are applicable for continuous states generally [24, 39]. Extension of the auto-regressive methods for the binary variables is generalized linear models which are used for modeling the data containing binary states and assume that the data is coming from a Bernoulli distribution [16, 6].

These methods also share similar disadvantages with AR (auto-regressive) methods. Dynamic Bayesian Networks (DBNs) which are a generalization of Bayesian networks are also used to infer network structure based on the functional connectivity (statistical relationships of the series) which is sufficient for discrete and continuous variables. However, one of the disadvantages of DBNs is the requirement for large sample sizes to accurately estimate the network interactions [31, 39].

Briefly, model-based approaches for network reconstruction aim to reconstruct the network based on the functional connectivities of the nodes by making assumptions about the generative function of the data. While they can detect the linear and pair-wise effect of nodes on each other, most of them fail to capture non-linear relationships or interaction effects where some nodes can affect the specific node together but they are not so important individually, which creates the limitation of distinguishing direct and indirect interactions. More information about the model-based methods, especially on neural recordings, can be found in [6].

2.2.2 Model-free approaches

On the other hand, model-free techniques do not depend on any specific assumptions about the underlying network structure or dynamical model and do not assume any generative function for the data. Instead, these methods aim to directly infer the network structure from the data without any or with as few parametric assumptions as possible. These methods can be more flexible but may also be more susceptible to noise in the data. Approaches that rely on statistical dependencies such as a correlation between nodes to infer the structural connectivity of the underlying network components can be considered model-free methods. These methods do not fit any generative model to the data but rather use the information contained in the data to explore the connectivity [6]. For instance, correlation-based methods measure the correlation coefficients between nodes' time series, and highly correlated pairs are counted as direct neighbors [40], which is determined by various thresholding techniques. But the assumptions about the linearity of the interactions can lead to failures in exploring the non-linear interactions between nodes. Moreover, correlation-based methods are not able to identify the direction of the interaction and cannot distinguish between direct and indirect effects. Partial correlation methods can tackle some of these problems as they can distinguish between direct and indirect effects by computing the connectivity measure between nodes while considering other nodes as well [49]. However, these methods cannot detect non-linear relationships and do not use the time information, which contains crucial knowledge about the nodes' relationships, as in correlation-based methods.

Transfer entropy [52] is one of the model-free methods that can capture non-linear relationships but suffers from the high variance and curse of dimensionality problem, which can cause failures when there are a large number of components (nodes) [44, 51]. It also requires a large amount of data to return accurate estimations. Another approach to measuring the functional dependencies between the time series of nodes is the Granger causality method [20], which measures the predictive ability of a given series to forecast another series. While Granger causality can detect the relationships between two pairs, it can give misleading results when the actual relationship involves more than two nodes or when dependencies are highly non-linear. Supervised approaches for network inference have also been proposed, such as using Convolutional Neural Networks (CNNs) to predict the possibility of edges between node pairs which uses labeled data as supervision [42]. In summary, model-free methods are much more general than model-based approaches and can be applied to data without assumptions about the generative function. Most of them are computationally efficient and can provide satisfactory results. However, they may fail when relationships are non-linear or involve more than two pairs of interactions during the time. While they can give better approximations about structural connectivity based on functional connectivity, most of the statistical or information-theoretic model-free methods [29] fail to identify the direction of the effects or distinguish between direct and indirect effects.

2.2.3 Prediction-based methods

Recent improvements in the field of machine learning, especially in deep learning, have affected network inference methodologies as well. As in many fields like NLP, computer vision, and other areas, applications of machine/deep learning have proposed insightful results in network reconstruction problems. Based on the predictive power of deep learning models in any context, in the NR domain, the main objective of related works is to use predictive models to predict the future states of the nodes based on their time series and reconstruct the network according to the quality of the prediction. Especially after the revealing of graph neural networks [56] it is possible to make predictions on graph-structured data successfully, which provides an advantage of learning the structural connectivity of the networks, not only the functional connectivity as in other methods mentioned above. The general formulation depends on the predicted adjacency matrix of the underlying network, where the predicted matrix should provide better prediction of the next states or current states of the nodes as it converges to the true adjacency matrix, where the distance between two graphs gradually vanishes during training. One of the

first works in this area belongs to Kipf et al. [32], where they use a variational autoencoder and decoder framework to learn the dynamical rules of the network and reconstruct the network structure. While their work has been tested on continuous dynamical models, networks with dynamics that have a discrete state space have not been covered. In 2017, Zhang et al. proposed a general deep-learning framework that applies to all kinds of state spaces, such as continuous, binary, or discrete [55]. This framework is called the Gumbel Graph Network (GGN) and consists of two modules, the dynamics learner and the network generator. By providing time-series data of the network as input to the model, it tries to generate a network structure that minimizes the dynamics prediction's error by joint training of the dynamics learner and network generator without the limitation of specific state space. These methods use time-series data as input, and time information contained in the data provides a lot of information about the nodes' interaction and evolution during some time, but it is not always accessible in real-world scenarios. The related work proposed by Gerrit Großmann et al. [22], which is called GINA, introduced a deep learning framework that considers independent "snapshots" instead of time-series data. These snapshots are independent and can be taken from any arbitrary point in time. Instead of trying to infer network structure that enables the best prediction about the next states of nodes as in the mentioned works above, they focus on finding the best graph structure that enables the prediction of a node's state on a given snapshot, not for the future time steps, by using the states of the nodes at the same time steps. To summarize, prediction-based network reconstruction methods are very successful frameworks to determine the structure of the underlying network based on time-series data or just snapshots. They can detect non-linear relationships, apply to any state space, and can generally distinguish direct and indirect relationships as they consider the structural connectivity, not only the functional connectivity, with the help of graph neural networks. However, they are mainly limited to undirected networks and are more computationally expensive than statistical approaches. They do not make assumptions about the dynamical rules of the system (except the fitted parameters of the dynamics learners) and are model-free approaches.

2.2.4 Neural Granger causality

In this part, we will introduce a proposed related work, "Neural Granger Causality" [51], which is very similar to our work in this thesis. As we mentioned before, Granger causality is one of the statistical methods to measure the predictive ability of time series on each other but has some limitations such as linearity. NGC (Neural Granger causality) is trying to tackle this problem by applying neural networks

instead of auto-regressive models to identify the Granger causality of the nodes' time series. More specifically, they train MLP or LSTM (long-short-term memory) models on the given time-series data to capture the interactions of the series in non-linear scenarios, which was the main limitation of most statistical approaches.

One of the interesting parts of this method is that there is a different prediction model for each series, and try to infer "non-causalities" from those models during the training. They call it Granger "non-causalities" because instead of detecting the causalities, they focus on removing the non-important variables from models which are non-causal with the target output series. This is done by adapting a regularization to the weights of the first layer of the models as below:

$$\min_{\mathbf{W}} \sum_{t=K}^{T} \left(x_{it} - g_i \left(x_{(t-1):(t-K)} \right) \right)^2 + \lambda \sum_{j=1}^{p} \left(\mathbf{\Omega} W_{:j}^1 \right) \tag{2.1}$$

Where x_{it} is the actual value of predicted series i at time step t, and g_i is the prediction model of i which takes input as previous steps (defined by K) of all variables. The term $\mathbf{\Omega}$ represents a penalty applied to the entire set of first layer weights for the input series j, denoted as $\mathbf{W}_{:j}^1$. This penalty term is a group lasso which is defined by $\mathbf{\Omega} \left(W_{:j}^1 \right) = ||W_{:j}^1||_F$ and shrinks the weights toward zero, where $||.||_F$ is the Froebenius matrix norm. Additionally, authors are using variations of the group lasso penalty such as *group sparse group lasso* and *hierarchical group lasso* to detect the casualties more effectively, which can be found in more detail in [51]. They call it the series of j is non-causal with the series of i if its weight is zero in the prediction model of series i. By forcing non-important series to have zero weight, they identify the non-causalities between the series. This method helps to capture non-linear relationships effectively based on a defined number of time steps and tackles the dimensionality issues, which is one of the limitations of some statistical model-free approaches.

Although they lift some limitations, their work is based on continuous time series and discrete or binary state spaces are not covered. Also, they train all models on the average loss of all models, which may lead to difficulties to learn node-specific behaviors. Choosing an appropriate penalty parameter is one of the challenging parts of the method too.

In our work, we have formulated our way of reconstructing the network based on this idea, where we try to identify the possibility of edges between nodes according to their functional relationships. To be able to observe these relationships in non-linear cases together with the linearities, we use an independent MLP model for each node's series, as in NGC, and try to interpret the models to identify the important

nodes. However, authors use regularization on the models to identify causalities, we will use different sensitivity analysis methods on trained models to interpret them, to capture the overall effect of node i on node j, which is its individual effect and interaction effects, instead of just looking at the weights penalized by regularization. This will lead our method to be suitable for any kind of state space and will provide more information about nodes' interactions than the weights and we will not have to choose a penalty parameter. As a difference, we will train each model independently in sequential order as model i will start to train after model j finishes training, where they will be optimized based on their own losses and will be able to determine node-specific dynamics.

Summary

In this work, our goal is to detect both linear and non-linear relationships between nodes, as well as interaction effects among a subset of nodes that may be more important for a specific node than their individual interactions. This is a limitation of pair-wise statistical model-free and model-based approaches. By adopting MLPs, we aim to tackle dimensionality issues and offer an alternative to prediction-based methods that require less computational time. Our method reconstructs networks based on predictions, making it applicable to any kind of state space. However, we are still using models (MLPs) to fit the data, and if consider the parameters of the fitted models, we call our approach prediction-based and "almost" model-free which can be placed between model-free and model-based approaches.

2.3 Dynamical models

In this section, we will introduce the dynamical models which we have used for data generation and experiments to measure the performance of our method. In our work, we have generated data by using various dynamical models which can be called *spreading processes* on the 4 types of undirected networks. Before going into details about the characteristics of each dynamical model, we would like to give the general method that we have applied for most of them. Firstly, it is known that there are discrete and continuous time models. Discrete-time models are kind of dynamical processes in which the process can be formulated in a way that it generates states at discrete time intervals where the generative function of the dynamical model is returning the state of a network at equally spaced time intervals which means that the amount of a passing time between time step t and $t + 1$ is always same. On the

other hand, continuous time models do not assume that the time between the state transitions is equal. Instead in continuous time models evolving of the dynamical process occurs continuously between any two points of time and the passing time between the states of a network is not equal as in discrete time models [35]. In most natural processes the components of the systems evolve continuously during the passing time, and using continuous time models for spreading processes on the networks would be a better fit than discrete time models as they explain the state changes more generally. Additionally, it may be more convenient to collect more accurate information about the underlying dynamical rules [23]. That is why we have decided to use continuous time models to run simulations with different dynamical rules over the experimented networks. Specifically, the dynamical models in our work have been formulated based on continuous-time Markov chain semantics [30, 21]. Moreover, most of the dynamical rules in our work are stochastic rules, which means that nodes of the network change their state according to the state space of the dynamical process stochastically based on its neighborhood region. If we notate the dynamical model as μ where the set of possible node states is S and the set of rules of the model is R, the state transition of each node occurs with the rule $r_i \in R$ as below:

$$s_j \longrightarrow s'_j \quad \text{with} \quad f(m) \tag{2.2}$$

Where $s_j, s'_j \in S$ and $s_j \neq s'_j$. It means that a particular node which is at the source state s_j can change its state to target state s'_j if the rule r_i is applied (fired) on the node with a rate that is determined by the rate function which depends on its neighborhood's states at a particular time point. Each node has the waiting time drawn from an exponential distribution with rate $f(m)$, and higher rates refer to the shorter waiting time for the changing of node i's state from s_j to s'_j, and only the node that has the shortest waiting time changes its state in a particular transition. $m \in \mathbb{Z}_{\geq 0}^{|S|}$ is the vector of the node's neighborhood that contains the number of neighbors at each possible state and $m[s_j]$ denotes the number of neighbors in state $s_j \in S$ at the current time step. Note that $f : M \to \mathbb{R}_{\geq 0}$ is a rate function that receives the vector of the node's neighborhood's states (m) and returns a rate for that specific node, where M is the set of possible neighborhood vectors in the network. By applying this general formulation to the different dynamical models, we simulate the process on given networks that are undirected and unweighted, we collect the nodes' states during some periods. As there is no specific time interval between changes, we run each simulation until the desired number of transitions occurred in the network, and reinitialize the initial state of the network to run the simulation again. This process continues until the desired amount of data is collected. Reinitializing the states after some time is a crucial and suitable way to

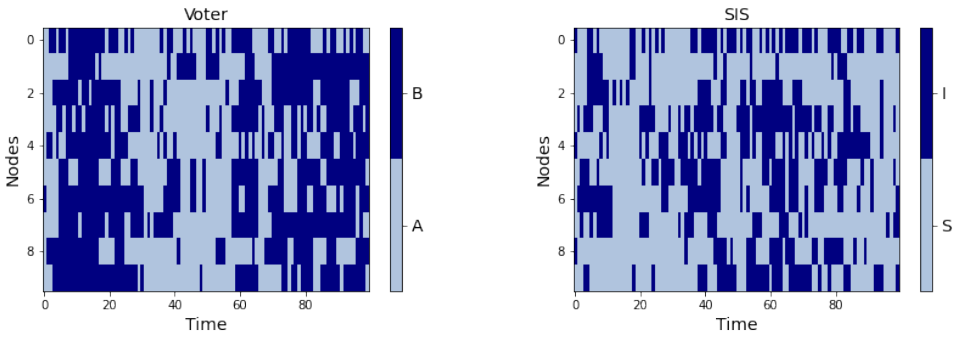
gather more information about the dynamic rules of the network.
We have applied the Voter, SIS, Game of ife, RPS, Forest fire, and CML dynamical models on the networks. Except for CML, the dynamical models have discrete or binary state space. Formulation and the implementation of the continuous dynamical models in our work are based on [21, 22]. We will introduce each dynamical model, its rules, and formulation below.

2.3.1 Voter

The voter dynamical model is a stochastic model which is introduced in 1975 [25]. It is modeling the effect of interacting participants of the network on each other's opinion. This is one of the simplest dynamical models with a binary state space $S = \{A, B\}$, and it models the spreading of an opinion among the nodes, especially in their neighborhood. More precisely, nodes change their opinions based on the opinions of their neighbors at time point t. However, the node's state or opinion at time point $t + 1$ is dependent on its neighbor's state at time t, it is independent of its own state at time t. In our work, the nodes change their opinions continuously over time as mentioned above, and nodes try to maximize the agreement with their neighborhood. Node i may change its state from A to B or from B to A with the following formulation below:

$$\begin{aligned} B &\longrightarrow A \quad \text{with} \quad f(m) = m[A] + \epsilon \\ A &\longrightarrow B \quad \text{with} \quad f(m) = m[B] + \epsilon \end{aligned} \tag{2.3}$$

Where $m[A]$ is the number of neighbors of node i at state A at time t, ϵ is additional noise subjected to all nodes with $\epsilon = 0.01$, and the waiting time for the node changes its state is drawn from an exponential distribution with rate $f(m)$. While the rate increases, the likelihood of the change is increasing. As shown in the equation, nodes tend to change their opinion to maximize the agreement in the network. In the Voter model, it can be possible that the network reaches its consensus state, where all nodes in the network are in the same opinion (A or B), and for this reason, gathering data with several simulations is a better idea to collect more useful information about interactions instead of using single simulation. You can see the example of the spreading of opinion over time on the network in Figure 2.1



(a) Voter **(b)** SIS

Fig. 2.1.: Example of Voter and SIS dynamics on 10 nodes graph over time

2.3.2 Susceptible-infected-susceptible (SIS)

Another dynamical model which is used for experiments and data generation in our work is the SIS dynamical model which describes the spreading of infection in the network [34]. SIS model is a stochastic dynamical model like Voter and has a binary state space where I indicates the node is infected, and S means susceptible. Nodes in the network are getting infected by their infected neighbors with the infection rate β, and infected nodes can get recover and be susceptible again with the recovery rate μ. It is possible that infection can vanish during this time, which means that there is no chance of any node getting infected. It is crucial to utilize diverse simulations and establish adequate infection and recovery rates to ensure the persistence of infections within the network. Nodes can be infected by their neighbors in the network as shown below:

$$S \longrightarrow I \quad \text{with} \quad f(m) = \beta m[I] + \epsilon \tag{2.4}$$

Likewise, infected nodes can recover again:

$$I \longrightarrow S \quad \text{with} \quad f(m) = \mu + \epsilon \tag{2.5}$$

You can see an example in Figure 2.1.

2.3.3 Game of life

Game of life is a zero-player game [27] and its evolution depends on the initial state of the game. In the original version of the game of life, evolving of the process occurs on an infinite grid surface. And each cell has binary state space $S = \{A, D\}$ where

A indicates the alive cells and D indicates the dead. Cells are born and die during the time based on the rules of the game, such as overpopulation, underpopulation, and other specific rules. In our work, we have formulated this game which applies to the finite networks [33], but as a continuous time model as in Voter and SIS. The transition of the node from alive to dead, and from dead to alive is formulated as below:

$$\begin{aligned} A &\longrightarrow D \quad \text{with} \quad f(m) = (m[A] + m[D]) - |m[A] - m[D]| + \epsilon \\ D &\longrightarrow A \quad \text{with} \quad f(m) = |m[A] + m[D]| + \epsilon \end{aligned} \tag{2.6}$$

The game of life ends when there is no life, or it can repeat itself after some time. To learn underlying interactions better, it is needed to run various simulations from different initialization as in SIS and Voter. We use $\epsilon = 0.01$

2.3.4 Rock-paper-scissors (RPS)

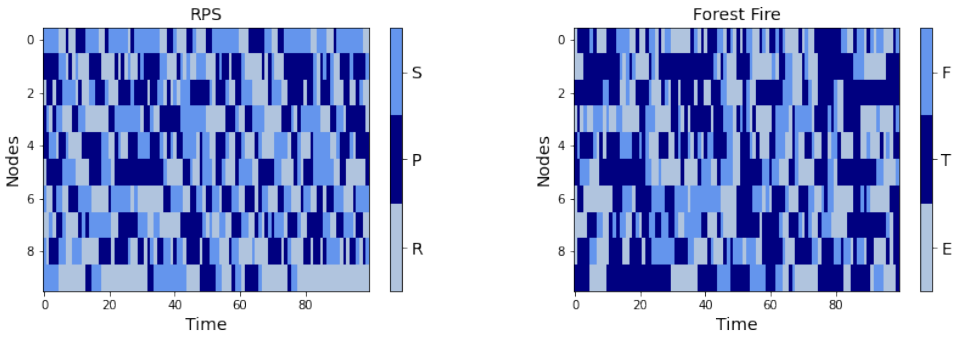
RPS is a very famous two-player game, which has state space of $S = \{R, P, S\}$ where R - Rock, P - paper, and S - Scissor. In contrast to the previously mentioned dynamical models, it has more than 2 states and helps us to learn and measure our method's performance on more complicated state spaces. RPS is an evolutionary game and its dynamics can be useful to study many natural phenomena by applying it to graphs [50]. We have applied RPS on different networks with different numbers of edges and nodes in our experiments similar to other dynamical models previously mentioned. In a network, nodes affect their neighbors with a changing rate σ to change their current states from R to P, P to S, and S to R as formulated below:

$$\begin{aligned} R &\longrightarrow P \quad \text{with} \quad f(m) = \sigma m[P] + \epsilon \\ P &\longrightarrow S \quad \text{with} \quad f(m) = \sigma m[S] + \epsilon \\ S &\longrightarrow R \quad \text{with} \quad f(m) = \sigma m[R] + \epsilon \end{aligned} \tag{2.7}$$

Where $\epsilon = 0.01$. You can see the evaluation of the game over time on 10 nodes graph in Figure 2.2

2.3.5 Forest fire

The forest fire dynamical model is another evolutionary model with three possible states [11]. This model is also defined as a cellular automaton like Game of life on a grid surface, and cells of the grid can be empty, occupied by trees, or fired. It is



(a) RPS **(b)** Forest Fire

Fig. 2.2.: Example of RPS and Forest Fire dynamics on 10 nodes graph over time.

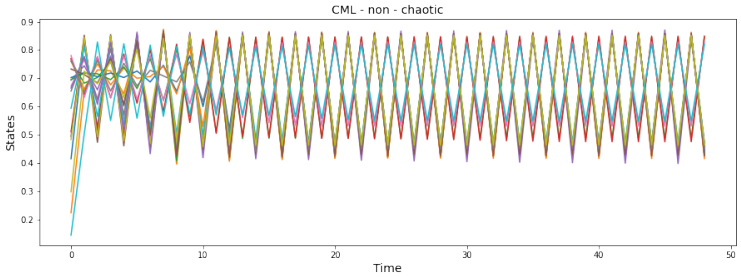
again a stochastic spreading process as other dynamical models in our work, and we have applied the Forest fire model on the networks with parameters growth rate g, spreading rate s, lightning rate l, and extinct rate e. In the initial state, there can be randomly chosen states for the nodes of the graph from state space $S = \{E, T, F\}$, and empty nodes can be occupied by the tree with rate g, occupied nodes can start to fire with the random lightning rate l and fire can spread to other nodes with spreading rate s, or fire can end with the extinct rate e. The model is formulated as shown below where nodes change their states in continuous time:

$$\begin{aligned} T &\longrightarrow F \quad \text{with} \quad f(m) = l + m[F]s + \epsilon \\ F &\longrightarrow E \quad \text{with} \quad f(m) = e + \epsilon \\ E &\longrightarrow T \quad \text{with} \quad f(m) = g + \epsilon \end{aligned} \tag{2.8}$$

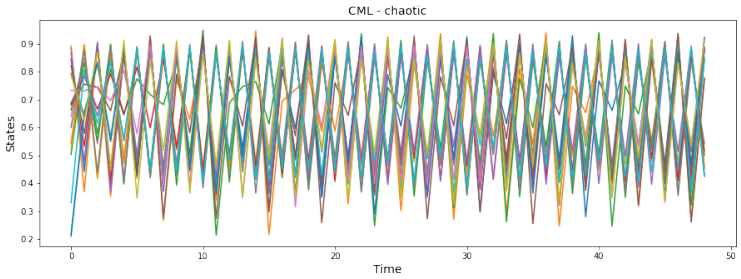
You can see the example of the spreading of fire over 10 nodes graph in Figure 2.2.

2.3.6 Coupled map lattice (CML)

Unlikely the previous dynamical models, coupled map models are deterministic and have continuous state space instead of discrete which we have used to measure the performance of our method for continuous states too, where the state space is much more complex than binary and discrete versions. Coupled map models are generally used for measuring the chaotic properties of physical systems [53] and are discrete-time models, which is very suitable to see the reconstruction quality of our approach where the nodes of the network behave in chaotic and non-chaotic ways. There are various types of coupled maps and one of them is coupled lattice map which is used in our work to generate time series on networks with continuous



(a) Non-Chaotic $\lambda = 3.5$



(b) Chaotic $\lambda = 3.8$

Fig. 2.3.: Example of chaotic and non-chaotic versions of CML dynamics on 10 nodes graph over time

states in range $[0, 1]$ in chaotic and non-chaotic settings which will be explained below. Nodes change their states based on the formulation below in CML:

$$X_i^{t+1} = (1 - s)f(X_i^t) + \frac{s}{deg(i)} \sum_{j \in m_i} f(X_j^t) \tag{2.9}$$

Where s is the coupling strength, and f is the mapping function which is the logistic map in our work:

$$f(x) = \lambda x(1 - x) \tag{2.10}$$

The chaotic level of the system is determined by the parameter λ which can get a value from the range $[0, 4]$. According to the [38, 55], $\lambda \approx 3.56$ is the boundary between chaotic and non-chaotic behaviors of the nodes of the network. In our work, we use $\lambda = 3.5$ for non-chaotic, and $\lambda = 3.8$ for chaotic cases to experiment with both situations. Coupling strength s is 0.2 is used in our case. You can review the difference between chaotic and non-chaotic cases on the 10 nodes graph in Figure 2.3.

3 Method

In previous sections, we introduced the objective of our method, which is to reconstruct networks using time-series data generated by unknown dynamics, without parametric assumptions about the dynamical rules that generate the data we have in hand. The method will mainly rely on the paradigm, which indicates that "the direct neighbors of a node are the nodes that affect its future behavior the most". However, this hypothesis can be analyzed by identifying the nodes' effects on each other in different ways, such as with traditional descriptive statistics methods. It will bring previously mentioned limitations, such as non-linearity failures, failures in more than pair-wise interactions, and so on. We build a deep-learning prediction framework instead of using statistical measures to tackle those limitations. We assume that the importance of a node on a specific node's future state prediction decreases as the distance between these nodes increases, and the nodes with the shortest distance in an interaction graph to the target node must be more important features when predicting the target node's next state. Surely, the subset of the most important nodes is not always the subset of nodes that enables the best prediction of a target node's future state, but checking all possible combinations of nodes is not a feasible solution; instead, we will try to approximate it based on the prediction we made by using all nodes' previous states except the target node by extracting the contained information about node contribution with SA methods. By analyzing the importance of the nodes on each other the whole interaction graph will be approximated without the limitation of the kind of possible states (binary, continuous, discrete). Our method consists of the following steps: the prediction stage, where we will try to predict each node's future state based on the previous state of the other nodes; the main part of our method, where we will apply sensitivity analysis to each trained model to interpret these models and find important features that contribute to the decision made by the model; a step in which we will try to classify nodes as neighbors or non-neighbors based on their importance weights and finally, a step in which we will convert the resulted weighted and directed adjacency matrix to a binary and symmetric matrix. To summarize, there will be 3 steps: **(1) Prediction**, **(2) Sensitivity analysis**, **(3) Neighbor choosing and reconstruction**

3.1 Prediction

Our method's input is time-series data (X) of the nodes, and the prediction part, which is the first "layer" of the process, is where we learn the dynamical rules of the system based on time-series. In recent years, the power of machine learning and deep learning algorithms has significantly increased, making them a suitable alternative to traditional methods in various fields, including time series prediction. Deep learning approaches have advantages in handling large amounts of data and automatically extracting relevant features from it, which can improve prediction accuracy, especially in modeling non-linear and complex relationships, which is crucial for accurate time series prediction. In particular, the development of recurrent neural networks (RNNs) and their variant, long short-term memory (LSTM) algorithms [47], has led to significant progress in time-series forecasting, which involves predicting future events over some time. While these algorithms are highly effective in time-series prediction, this work uses a multi-layer perceptron (MLP) model, which is a type of deep learning architecture, for the prediction step. The reason for that is that MLP models are much more computationally efficient than RNNs; secondly, they are more interpretable, which will be more convenient for us in the SA part; and lastly, as we will use the one previous step for the prediction, there will no need to use sequential information and MLP will be sufficient. But we will also see the effects of using more than one time step in the experiments section.

Prediction of next states

Our prediction layer consists of independent MLP models, which are node-specific dynamics learners. The main purpose here is to learn how a dynamical model's rules affect a specific node, even when other nodes in the network do not behave similarly. When the input data is received, it is prepared for each model where the model of node i is fed with previous states of other nodes and learns parameters with the supervision of target node i's time series. In this way, during the training process, our node-specific models will learn the underlying interactions between nodes and will predict the target node's next state, and they will be able to learn the dynamics of a system, but not for a whole network, only for a specific node. But surely, by combining those independent models, we can make predictions about the whole

system. Our prediction step focuses on two assumptions:
1) Markov property in time:

$$P(\mathbf{X}^{t+k}|\mathbf{X}^t, ..., \mathbf{X}^1) \approx P(\mathbf{X}^{t+k}|\mathbf{X}^t) \tag{3.1}$$

Which means that the current state of the stochastic process can be enough to give a reasonable prediction about its future state after k jumps, where k is the arbitrary sampling rate from the underlying stochastic process and
2) Markov property on the network:

$$P(\mathbf{X}_i^{t+k}|\{\mathbf{X}_j^t|j \in V) \approx P(\mathbf{X}_i^{t+k}|\{\mathbf{X}_j^t|j \in N(i)\} \cup \{\mathbf{X}_i^t\}) \tag{3.2}$$

Which means that the direct neighborhood of the node and the node itself contain all the information needed to predict the future state of the node [1]. However, those assumptions hold for the stochastic processes in continuous time, we assume that not using the node's previous information in its own prediction model will make us identify other nodes' effects more clearly. But consequently, this will lead to poorer prediction results compared to using the node itself in the prediction. We will see the effects of sampling rate k and the effects of using the node itself or not in the experiments section in more detail. If there are N nodes in the data and T time steps, the size of the data will be $T \times N$. It means that we will have a N separate predictor model, as each node has its own predictor. We can think of other nodes as predictor variables and the target node as a dependent variable for a specific model. So, for node i, there is model f_i, which takes as input the X_{-i}^t vector of states of other nodes with size $N-1$ at time step t, except node i, and outputs X_i^{t+k}. This process can be formulated as $X_i^{t+k} = f_i(X_{-i}^t)$ for a single observation of time. And here the process becomes a simple supervised learning process, and the aim is to optimize the parameters (weights and biases) of the model f_i that enable sufficient prediction about node i. In this way, we train our model f_i on whole training observations and learn the dynamical rules of a system for node i. It is an iterative training process, and models are not trained at the same time. The training process starts for the first node, and after it finishes, the model of the second node starts for training, and this process continues till all models are trained. We can define a set $F := \{f_1, \ldots, f_N\}$ that contains N models as the number of nodes, where each of them will be trained with their own training data that does not contain the information of the predicted node.

MLP

We mentioned the importance and benefits of using deep learning algorithms in prediction problems, especially in time-series forecasting. In this section, we will introduce the properties of our main predictor model, which is the MLP (multi-layer perceptron).

MLPs are feedforward neural networks that can be used for classification and regression tasks in deep learning. They consist of multiple layers of artificial neurons, including the input, hidden, and output layers. Activation functions like sigmoid, ReLU, and tanh are used on top of the hidden layers to identify non-linear relationships in data, but choosing an activation function is a very crucial step in model building [28]. The weights and biases of MLPs are learned through training using an optimization algorithm like stochastic gradient descent, which is our choice for our task, to minimize the error between predicted and true labels determined by the appropriate loss function for the problem, by iteratively adjusting the weights and biases. As mentioned before, one of the key advantages of MLPs is their ability to model complex relationships between the input and output data. They can learn to make accurate predictions even when the data is highly nonlinear, and it helps us a lot while trying to predict the state of a node in the next time step if the dynamical model of the network is non-linear. However, they can be sensitive to the choice of hyperparameters, such as the number of hidden layers and the number of nodes in each layer, and can require a large amount of data to learn patterns accurately. You can review the model properties, use cases, and architectures more broadly at [19]. We use MLP as a predictor of the nodes in two different structures for classification and regression tasks on discrete (categorical) and continuous states, respectively, which are chosen depending on the use case as an argument. All MLP models have the same structure in terms of the number of layers, units, or activation functions. The activation function that is applied on the outputs of the respective hidden layers of the models is ReLU in our case for the classification tasks, and the softmax function applied on the output layer to arrange the results in the $[0, 1]$ range for each possible state to indicate the probabilities of each observation belonging to each class that is available in the training data. If we notate the model of node i as f_i, we try to approximate the true function that has been applied to each node during the dynamical process in a given time window with f_i. So if the true function is f_i^*, we try to minimize the difference $f_i - f_i^*$ by optimizing the parameters of f_i with the optimization algorithm SGD [10]. In classification problems, where the node states are discrete classes or binary classes, we use a categorical cross-entropy loss function in our models, to measure the difference between actual values and

predicted values. The formulation of the categorical cross-entropy loss function can be shown below:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{m} y_{i(j)} \log(f(x_i)_{(j)}) \tag{3.3}$$

Where m is the number of classes (states) and N is the number of observations in the data. $y_{i(j)}$ is a binary indicator that indicates the true class of the i-th observation is class j or not, and $f(x_i)_{(j)}$ is the predicted probability by the model f for the observation i belonging to class j. The loss function will be calculated based on these values to measure the error and will be optimized with backpropagation iteratively during the predefined epochs, which is the number of iterations. One of the main and most famous problems in machine learning is overfitting [45], where the validation loss starts to increase after some epochs while the training loss is decreasing, and it means that the model starts to memorize the training observations and perform very well at the training set but fails at the validation data, so we have to choose an optimal set of parameters and several epochs to keep our model's training and validation as low as possible at the same time. Because overfitting is the main problem when we want to extract nodes' effects, as the model will not be an accurate estimator of the real interactions of nodes. To deal with this problem, we use 'early-stopping', which prevents the model from overfitting by assigning penalties when the validation loss of the model is greater than the previous epoch, and when the number of penalties reaches a predefined threshold, the model stops training. The complexity of the model (number of hidden layers, neurons, learning rate, number of inputs, etc.) is a very crucial factor in the sense of overfitting, and we should keep the complexity at an optimal rate based on the bias-variance trade-off [13]. Overfitting issues in trained models have a large impact on sensitivity analysis results, resulting in misleading connections in reconstructed networks.
In continuous models, where the states of the nodes at each time step are continuous, we use a different model structure that has more hidden layers and hidden units than the classification model to be able to capture the more complex interactions as in CML. In this case, we do not use the softmax activation function at the output layer and instead use linear activation and the loss function for the model is a mean-squared error to measure the differences between actual and predicted continuous values, which is formulated below:

$$L = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{3.4}$$

These are the main models we have used to predict node values and learn underlying dynamical rules for the data. For each respective case, the model of each node went

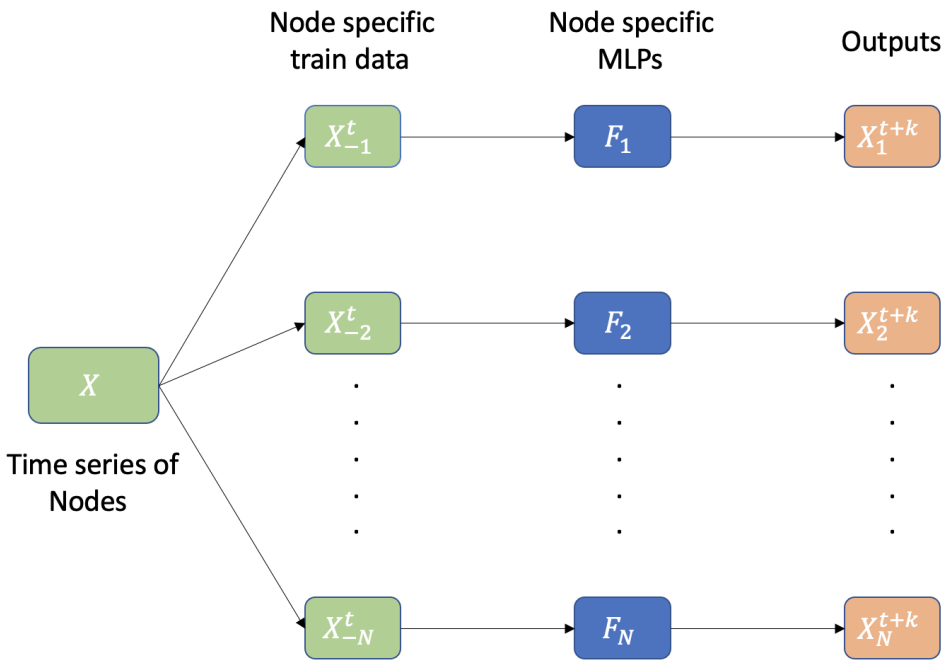


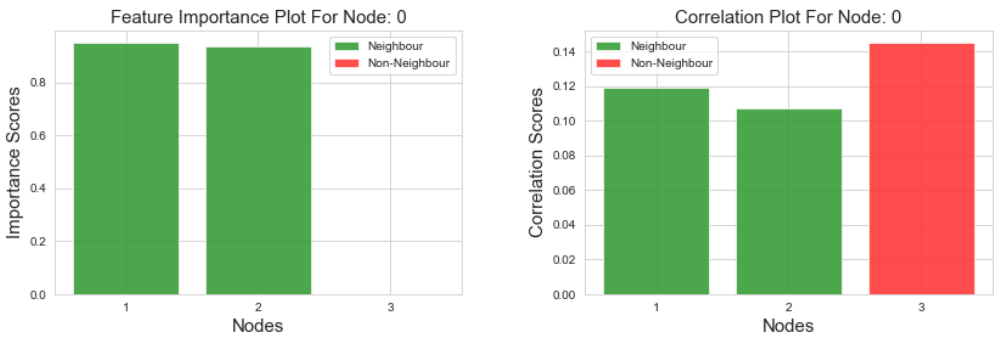
Fig. 3.1.: Figure shows the example of a training process for all models for a sample graph with N nodes.

through the training process explained above, and at the end, we got N trained models that are capable of predicting a specific node's future behavior after the prediction step of our method. However, while our default model is MLP, we will also experiment with the effects of different models such as LSTM in terms of dynamics prediction and network reconstruction, especially to see the effect of using more than one previous piece of information from other nodes on the prediction of a target node's future state and compare results with our default models. You can see the overview of the prediction process in Figure 3.1

3.2 Node importance

In this section, we will describe the methods that we have used to extract useful information from the trained models for network reconstruction, which means that we will try to weight each node based on its importance in another node's prediction, and these weight scores will be an approximation of the possibility of an edge between these two nodes. As mentioned before, we are working on

undirected graphs, which means that the edge between two nodes will not have any directionality. However, while we will find important nodes for each predicted node, and we will assume those nodes are neighbors with each other, it is not always guaranteed that the assumed important nodes are the true neighbors of a target node. These scores are assumed to be the best possible approximation of the structural connectivity that we will gain based on the functional connectivity of nodes from their prediction models, but their true structural connectivity may be different. For example, there can be three nodes: node (1), node (2), and node (3), where node (1) is connected to node (2) and node (2) is connected to node (3), but there is no connection between node (3) and node (1). Still, there can be a correlation between node (3) and node (1), and these correlations can affect our analysis results [1]. As we talked about before, it is one of the main problems in traditional network reconstruction approaches, but we assume that nodes' neighbors are the nodes that affect the prediction most, which means that other undirect effects will probably be lower than actual neighbors' effects, so these effects will not impact our results too much unless they are the same as or even greater than direct neighbors' effects. As we will consider the other nodes' interactions too while measuring the importance of node i on the node j's model, it is more likely that its importance will not be significant if it is indirect and can be explained by other nodes, which is not the case in pair-wise calculations as in correlation-based methods. Also, we apply sensitivity analysis to all nodes, and there can be one node that seems important on another node, but this may not be the same on the opposite side, and this will also reduce the risk of false positive edges in some amount. Analyzing the sensitivities of the nodes on each other's future behavior given all nodes will help us determine the interaction effects too, where a node does not affect the target node individually but it is important for the prediction together with some subset of nodes, which cannot be identified with pair-wise dependence measurements. It may occur in a specific dynamical model of the system, where all neighbors together determine the next state of the target node, but there is not such a strong correlation or any other individual relationship, which creates a highly non-linear decision boundary. If we think of the XOR example, we can understand that while linear models or statistical models fail to identify these kinds of relationships, neural networks can easily solve these patterns, and using sensitivity analysis, we can also gain access to these underlying interactions.



(a) Importance scores on prediction

(b) Correlation between nodes

Fig. 3.2.: Figure shows the example of a 4-node graph that has a dynamical model based on XOR rules, and we can see that while there are no clear relationships based on their pair-wise correlation, we can determine the real effect by analyzing our prediction model.

3.2.1 Sensitivity analysis

Sensitivity Analysis is one of the most-used methods for interpreting and revealing the important relationships between variables and their impact on the model's output. Especially, it is one of the most used techniques to interpret deep learning models and explain their decision-making processes. As is known, deep learning models are kind of a black box and very complex in terms of interpretability compared to other traditional machine learning models. But still, using different SA methods, we can explain the model's outcome and identify important features. There are very different techniques of SA applied to artificial neural network models [12]. Overall, sensitivity analysis is an important tool for understanding how different input parameters influence the output of a model or system. There are two types of sensitivity analysis: local and global. Local sensitivity analysis focuses on the impact of a change in a single input value on a respective output, while global sensitivity analysis analyzes the effect of the entire range of input space on the model's output. Global sensitivity analysis is more comprehensive, but it can be computationally intensive, particularly for large systems. In contrast, local sensitivity analysis requires less computational power and is often used when the global analysis is not feasible [5]. In our work, we will focus on the individual node's total importance on another node's prediction and for this reason, will try to approximate the global sensitivity of nodes. Our main analysis method will be based on the permuting input features and approximate their global importance on the output values. Additionally, we will use 2 other local SA methods which are more computationally efficient in larger networks, for comparison with our main SA method. Their performances on different graphs and dynamical models will be discussed in the experiments section too. Mentioned

SA methods in our work are : **(1) Input permutation SA**, **(2) Partial derivatives SA**, **(3) DeepLift**.

Input permutation

One of the most common use-cases of sensitivity analysis on artificial neural networks is conducted by changing or reformulating the specific input parameter (adding noise, masking, etc.) to check the sensitivity of the outputs to these minor changes in the input feature and measure the importance of the relation between the respective input and the target values. Many methods in the literature have been developed based on the input-output relationship [17], which is about input perturbation and tries to add additional noise to change the input values for sensitivity analysis. In our work, we have used the Feature-based Sensitivity of Posterior Probabilities (FSPP) method [54], which uses input permutation instead of adding additional noise to input and leads to a better approximation of the input's effect on the output by completely 'removing' it from the feature set of the model in an efficient way. The FSPP method for selecting features involves considering how sensitive the probability of a certain outcome y_i is to the presence or absence of a particular feature j. This sensitivity or effect is measured by calculating the absolute difference in the probability of the outcome with and without the feature and summing these differences across the entire feature space, which yields a global sensitivity approximation. In general, calculating the feature's importance by measuring the model's performance based on the feature's presence and not in the model requires training the same model N times if there are N features. It is computationally very expensive, especially in our case: if we have N nodes in a graph, we have to train N models, and each model has to be retrained $N-1$ times for sensitivity analysis, which yields $N(N-1)$ training processes. To tackle this problem, instead of removing the feature from the feature set, they use an approximation of it, which we also use as an approximation in our work. Firstly, we can introduce the calculation of the importance score for feature j by the equation proposed by the FSPP method:

$$S^P(j) = \sum_{k=1}^{c} \int |p(y_k|x) - p(y_k|x_{-j})| p(x) dx \tag{3.5}$$

Where $S^P(j)$ is the sensitivity score of feature j, x_{-j} is the sample derived from the data excluding node j in the prediction model of the target node, $p(x)$ is the probability density function of x, which is the original time observation including node j, and y is the state of the target node if there are c possible states. It can be understood from the equation that the importance of the node j depends on

the absolute difference between the model's output probabilities with and without node j over the whole future space. As mentioned above, the issue here is that the process is computationally expensive because we train the model without each feature (node). In this step, we use an approximation of the process. We do this by random permutation. Instead of removing feature j from the training data and re-training the model, we apply random permutation to feature j, which is swapping the feature values randomly. By this method, we can say that the data without feature j and with randomly permuted feature j are approximately the same, so we will not need to retrain the model again for each feature. The sensitivity score of the feature (node) j in 3.5 can be approximated over time series X_{-l} (which is containing n time steps of all nodes except the target node l) for a model of l-th node:

$$\hat{S}^P(j) = \frac{1}{n} \sum_{k=1}^{c} \sum_{i=1}^{n} |p(y_k|x_i) - p(y_k|x_{(j),i})| \tag{3.6}$$

where $x_{(j),i}$ is a single input observation with permuted value of feature j instead of excluding it. The average absolute difference between posterior probabilities of the outputs is counted as the sensitivity score for node j. To prove the approximation $p(y_k|x_{-j}) \approx p(y_k|x_{(j)})$, authors note that distribution of $p(x^j)$ will not change after uniformly random permutation and can be written as $p(x^j_{(j)}) = p(x^j)$, where x^j is the sample with original feature j, and $x^j_{(j)}$ is the sample with uniformly randomly permuted feature j. So,

$$p(x_{(j)}) = p(x^j_{(j)}, x_{-j}) = p(x^j_{(j)})p(x_{-j}) = p(x^j)p(x_{-j}) \tag{3.7}$$

And

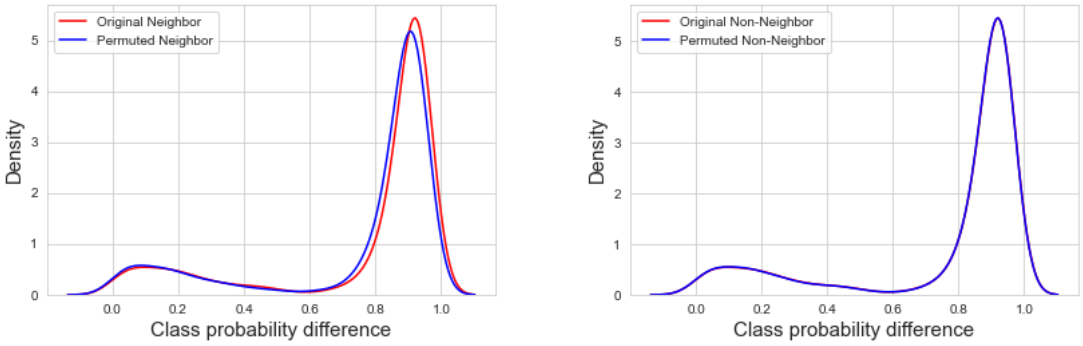
$$p(x_{(j)}, y_k) = p(x^j_{(j)})p(x_{-j}, y_k) = p(x^j)p(x_{-j}, y_k) \tag{3.8}$$

Hence,

$$p(y_k|x_{(j)}) = \frac{p(x_{(j)}, y_k)}{p(x_{(j)})} = \frac{p(x^j)p(x_{-j}, y_k)}{p(x^j)p(x_{-j})} = p(y_k|x_{-j}). \tag{3.9}$$

The proof is taken from FSPP paper [54].

The input permutation method or FSSP algorithm is applied to the probabilistic MLP models generally, which is the classification problem, but we mentioned before that based on the dynamical rules of the system, the states of the nodes can also be continuous values instead of binary or discrete classes. As we apply our method to any kind of system, we also consider the application of the input permutation method to continuous values, or, in other words, to the MLP for regression. In that case, we formulate the process the same as in a classification model, but the only difference is that the output of the model is not probabilistic. So we cannot



(a) Neighbor **(b)** Non-Neighbor

Fig. 3.3.: Figure shows the example of differences between class probabilities in original and permuted versions of a feature which is neighbor in (a) and non-neighbor in (b). We can see that input permutation helps us to see the effect of a feature on the predictions. Plots are generated from the model of the 0th node of the grid graph(7x7) with Voter dynamics

measure the L1 distance between probabilities while permuting the feature to see the difference. Instead of measuring the absolute differences between probabilities, we use the MAE, which is the mean absolute error of the output values, which the model predicted with original feature j and uniformly randomly permuted feature j. And for the regression task, equation 3.5 can be written as follows:

$$\hat{S}^P(j) = \frac{1}{n}\sum_{i=1}^{n} |\hat{y} - \hat{y}_{(j)}| \tag{3.10}$$

Where $\hat{y}_{(j)}$ is the predicted value with the permuted feature j. We can access the importance or approximated effects on prediction in regression problem cases by applying this equation to all features (nodes) after randomly permuting them. In both cases (classification and regression) we apply the input permutation method to the test set of each model after training the model on the training data and measure the overall effect of the respective node in the prediction which shows the approximate strength of their relationship and will be used to identify if they can be neighbors or not. This process is done for all models separately. Technically, it is possible to apply the SA on training data too, but as the test data is a better approximation of the model's performance and real predictive power, we assume that sensitivity analysis on test data will also give a better approximation of the relationships between nodes. However we apply permutation to test values only, permutation can also be applied to the whole feature, including train and test set together, not only to test values. But to get a more random and changed feature, especially for the binary or discrete features when there is not an equal number of

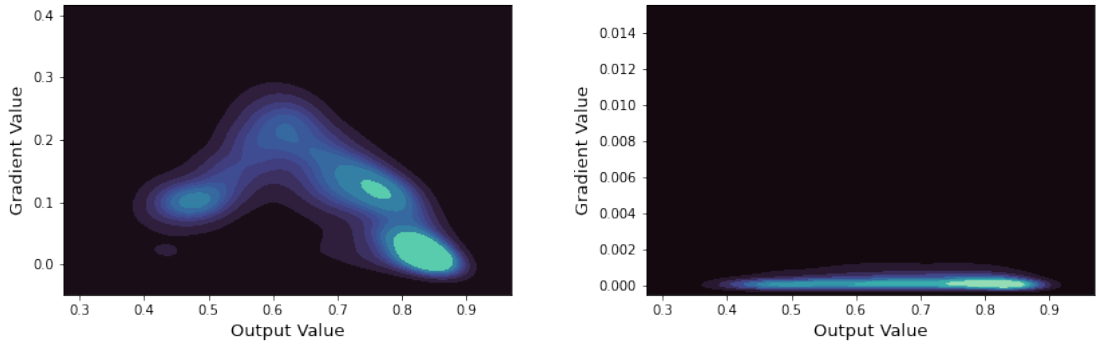
states (classes) in the test part of the feature. Surely this process has to be done after training, which means the test data is never seen by the model and there is no information leakage from the train set to the test set. You can observe the differences between the probability distribution of classes before and after the RP process on one of the neighbors of the target node in Figure 3.3.

Partial derivatives method

Our second experimental sensitivity analysis method is the well-known partial derivatives method, which is widely applied to artificial neural networks to interpret them and find underlying relationships between features. The partial derivatives allow you to determine how small changes in each input affect the output by measuring the gradient locally at this point. This is done by calculating the Jacobian matrix, which contains the first-order partial derivatives of outputs with respect to inputs. It tells you how much the function value will change as a result of a small change in the input. As our aim in this work is to find the impacts of nodes on other nodes and identify their underlying relationships, derivatives can help us a lot in this manner. That is why we have chosen this traditional method for testing in our work together with the other two SA methods and for their comparison. We refer to the related work [41] in our method for calculating the partial derivatives of outputs concerning the inputs of the MLP model, and the resulting values are accepted as sensitivities of given inputs to predicted node. This method is the local sensitivity method, and we measure the sensitivity of each variable at a certain point but calculate the average effect of each variable based on the resulting values from each observation of test data. We define the partial derivatives of the output of MLP concerning its input variable j as:

$$s_{jk}|_x = \frac{\partial y_k}{\partial x_j}(x) \tag{3.11}$$

where y_k is the output of k^{th} neuron in output layer, x_j is the input of the j^{th} neuron in input layer, and x is the t-th sample of data which derivatives are evaluated on. These sensitivities are collected by applying the chain rule to the partial derivatives of the inner layers of the MLP model. A more detailed explanation of the calculations of the derivatives can be found in [41]. As the process we followed in the previous analysis method "input permutation", we apply the same here for the partial derivatives method. After training the model for each node independently, we apply the partial derivative method to each model based on test data. There are several methods to calculate the overall sensitivity of an input variable based on



(a) Neighbor **(b)** Non-Neighbor

Fig. 3.4.: Figure shows the density of the gradient values of outputs with respect to inputs over test data for the neighbor (a) and non-neighbor(b) of a predicted node. We can see that gradient values are in a broader range for the neighbor node than for the non-neighbor. The brighter regions in the plot indicate the higher density of the gradient values. Plots are generated from the model of the 5th node of the grid graph (7x7) with CML dynamics

calculated gradients, such as mean sensitivity, sensitivity standard deviation, and mean squared sensitivity. We apply the mean squared sensitivity in our work, which can be formulated as below:

$$S_{jk}^{sq} = \sqrt{\frac{\sum_{t=1}^{N}(s_{jk}|_{x_t})^2}{N}} \tag{3.12}$$

Where N is the number of samples in the test data of the respective model. With this equation, we calculate the average sensitivity of the jth variable to the kth output neuron. We apply this equation in this form in regression problems where the node states are continuous values, but in classification problems, we take an additional mean value, which is the average sensitivities of each node for each class. As we calculate the importance of each node on the target node based on partial derivatives, we assign importance scores to each node and start the neighbor-choosing process.

DeepLift

Till now, we have introduced two SA methods that we have applied to the trained models of nodes to measure the effects of other nodes on this model's predictions. These methods were mainly intended to measure the sensitivity of the output values of the model to small changes in the input values. However, they are sufficient to approximate the effects of nodes on each other and identify their underlying

relationships, but they have some limitations in some cases in terms of measuring the importance. One of the possible problems in the other two methods, that Deeplift tries to tackle is the saturation problem, which is a common issue in deep learning algorithms. For instance, if we think a simple model $y = 1 - max(0, 1 - i_1 - i_2)$ and inputs i_1 and i_2 are taking the values 1 and 1 respectively, then the output of the model will be 1 (an example is taken from [48]). If we apply the input permutation method on feature i_1 and its value changes to 0 randomly, in this case, the output value will be 1 again, and we will not measure the contribution of the feature accurately. Or if we apply the partial derivatives method, hence the gradient of the activation function with respect to input will be zero, we will not be able to measure the sensitivity of output to the input. Another problem is the threshold problem, where the permutation and gradient method fails to identify the contribution added by bias in the activations. To provide sufficient results in such cases, we have added a method called DeepLift, which has been proposed in 2017 [48], to address the mentioned issues of permutation and gradient methods in the sense of model interpretation. Calling this method the SA method is not the best identification, because DeepLift is mainly focused on measuring the amount of contribution of each input to the output value that is predicted by the model, which is trying to find the amount of the contribution by any feature in a positive or negative direction on the output value based on a single input sample x, and it is more appropriate to call DeepLift the model interpretation method. It is again a local interpretation or sensitivity method, but we will take the average importance by passing all samples in test data to approximate the overall effect of the feature (node) as in the other methods.

DeepLift tries to measure the feature importance by focusing on the slope which is the change of the output upon the difference between the input value and some reference value, instead of looking at the small changes in the actual value of input as in the gradient method. We can think of a DeepLift method as a combination of input permutation and partial derivatives methods in terms of methodology, where we try to find the sensitivities based on the difference of output values coming from original and reference input, and we do this with backpropagation in the single backward pass, which is explained below in more details. The reference values are chosen by the user based on the domain knowledge about the use case. We change input values with some reference value (zero for example) and calculate the output. If we define the input neurons in some layer as $[x_0, x_1, ...x_n]$ and the output neuron as y, DeepLift is measuring the difference of Δy upon to Δx which is the difference of inputs from their references $[\hat{x}_1, \hat{x}_2...\hat{x}_n]$. So the method aims to calculate the amount of Δy that occurred because of Δx_i. To measure the effect of Δx_i on Δy they define a 'multiplier' which is a kind of replication of the gradient. If we

define the contrubution as $C_{\Delta x_i \Delta y}$, multiplier is $m_{\Delta x_i \Delta y} = \frac{C_{\Delta x_i \Delta y}}{\Delta x_i}$, and consequently $C_{\Delta x_i \Delta y} = m_{\Delta x_i \Delta y} \Delta x_i$. Based on the summation-to-delta property, the sum of the contributions from all inputs values in an observation must be equal to the Δy, such as:

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta y} = \Delta y \tag{3.13}$$

If we say that $x_1, x_2, \ldots, x_n$ are inputs, $h_1, h_2, \ldots, h_n$ are hidden neurons, and y is the output neuron, to calculate the multiplier $m_{\Delta x_i \Delta y}$, the chain rule is used as calculation of derivatives, which holds for multipliers too. The chain rule for multipliers hold as [1]:

$$m_{\Delta x_i \Delta y} = \sum_{j} m_{\Delta x_i \Delta h_j} m_{\Delta h_j \Delta y} \tag{3.14}$$

By this chain rule, we can calculate the impact of each feature on output similar to the partial derivatives method as $C_{\Delta x_i \Delta y} = m_{\Delta x_i \Delta y} \Delta x_i$. As we said before, the DeepLift method is mainly a model interpretation technique and defines the direction of the contribution separately as negative and positive contributions to explain the impact of the input on the decision, but we consider them together as total impact to define the importance of input on respective output. In the linear parts of the model such as the Dense layer, the multiplier of the respective neuron w.r.t its input is the weight between them as:

$$\begin{aligned} y &= b + \sum w_i x_i \\ \Delta y &= \sum w_i \Delta x_i \\ \Delta y &= \sum_{i=1}^{n} C_{\Delta x_i \Delta y} = \sum \Delta x_i m_{\Delta x_i \Delta y} \\ m_{\Delta x_i \Delta y} &= w_i \end{aligned} \tag{3.15}$$

And on the non-linear part of the network, such as activation functions, there is only one input and the summation-to-delta property holds as $\Delta y = C_{\Delta x_i \Delta y}$, so $\Delta y = m_{\Delta x_i \Delta y} \Delta x$, and $m_{\Delta x_i \Delta y_i} = \frac{\Delta y_i}{\Delta x_i}$. In the case of $\Delta x_i = 0$ which means $x_i = \hat{x}_i$, it is possible to set the multiplier to $0.5 w_i$ in linear parts or set it to the actual gradient in non-linear parts to avoid any numerical instability [48]. These are the applied 'Linear rule' and "Re-scale rule' of the DeepLift method in our work, but there are other applications such as the 'Relevance rule' or integration with shapely values method [36] too which are not considered in our work, but for further explanation and details you can review the DeepLift's original paper [48], which we refer to for explanations of the method and equations (notations) for the explanation. In our

[1] Proof can be found in `http://proceedings.mlr.press/v70/shrikumar17a/shrikumar17a-supp.pdf` [2]

work, we apply the method on test data after training, and have sum the contribution of each node to each target class, and take the average contribution value by dividing the number of classes in the classification case. As there is only one output value in the regression case, there is no need for such an operation. As a reference value, we have chosen 0, which found as a more optimal choice for both cases (classification and regression) empirically.

Summary

In summary, we have modified and applied 3 different sensitivity analysis methods: input permutation measures the importance of nodes on each other by approximating the difference of the model's output when a node is not included in the model; partial derivatives measures how output values are sensitive to small changes in input values; and DeepLift measures the amount of contribution coming from the difference between a node's actual state and reference state to the difference between output and reference output. Using three different methods is for comparison of the results and to see their effects on the different scenarios, but from a broad point of view, all of them are sufficient to use in our method for network reconstruction. We start by training the independent model of node i and apply one of these SA methods on the test data to collect an importance score for each node, which is a feature of the respective model. We apply this process to every single node, and at the end, we get the importance score of nodes on each other. As a result, we get the weighted adjacency matrix W of the graph, which contains the node's relative importance scores. If we have N nodes in the network, there will be N trained models, and each model will produce a vector of importance with $(N-1)$ elements. Then the predicted weight matrix W matrix will have a size of $(N \times N-1)$. In the next section, we will explain how we convert W into a binary and symmetric matrix with a size of $(N \times N)$, as we limit ourselves to undirected graphs in this work.

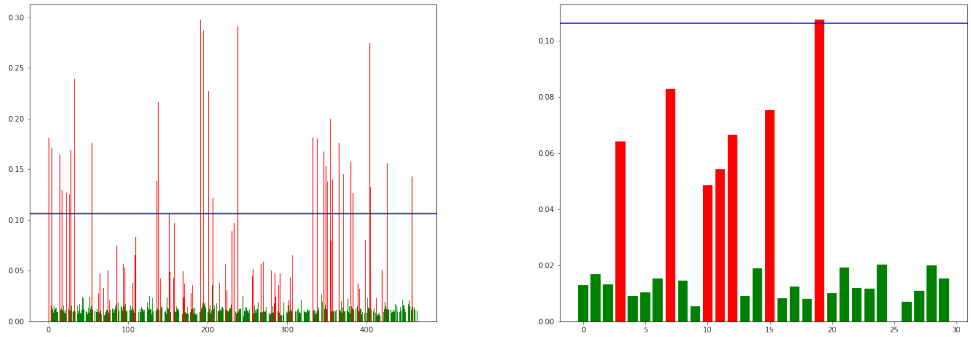
3.3 Neighbor choosing and reconstruction

In this section, we will explain the process of choosing a neighbor for each node based on the importance scores gained from the sensitivity analysis of trained MLP models. We mentioned that after the SA "layer" of the process, we get a weight matrix W where each row vector contains the importance scores of nodes on a specific node except itself. If we notate the vector of importance scores as W_i for the node i, it is the weighted vector of size $N-1$, and our aim in this section is to

apply a reasonable method to find an optimal threshold to classify these weights as neighbors or non-neighbors of the respective node without using the ground truth information. As we mentioned before, we assume that the ground truth graph is undirected, meaning that we do not consider the direction of the possible interaction between nodes and we only focus on the existence of the edge between two nodes. That is why finding a sufficient threshold that classifies the weights where node i and node j are classified as neighbors in each other's weight vectors, W_j and W_i, will be enough to reconstruct an undirected graph. However, there could be an alternative analysis based on the relative importance scores as if there were a strong effect of the node i on the node j, but there is not such an impact of the node j on the node i, and based on the approximated weights, the direction of the relationship can be approximated too, but this is not part of our work in this thesis.

In many related approaches, network reconstruction methods return a weighted adjacency matrix of the predicted network as an output, which is also the case in our work, and to classify these weights, there are several approaches such as Fisher's Z-transformation which try to find z-scores for the weights and convert them to p-values, then choose the threshold based on the p-values, or choosing based on clustering coefficient [14], or choosing based on random matrix theory [37], or choosing threshold based on a given quantile of the weights. There are approaches like deciding the threshold based on the characteristics of the ground truth graphs, such as the average degree of the true graph, which requires the ground truth information. Most of these approaches depend mainly on the specific setting or domain knowledge to decide the cut-off quantile or the cut-off for p-values. As we do not use any GT information, we will not use supervised methods based on the average degree of the true graph too, but instead, we will use unsupervised learning, which is clustering in our method.

Based on the assumption that the nodes that are connected are more likely to affect each other's future behaviors the weighted importance scores should tend to create underlying groups with higher and lower scores, which can be treated as neighbors of the node or not in our case. To identify the underlying group similarities in the weight vectors, we use the type of hierarchical clustering known as agglomerative clustering [46]. Agglomerative clustering is working as a bottom-up approach, which means that it starts from the smallest cluster and continues to the bigger one. More precisely, it takes each weight in a vector as a single cluster at the beginning. Then, based on the distance between every single cluster, it takes the two closest clusters and combines them into a common cluster. Ecludian distance is used as a distance measure between the clusters in our work. And this process continues until there is one whole cluster. Then, based on the use case, the user can choose the optimal



(a) Global Clustering of the whole network **(b)** Local Clustering of Node 25

Fig. 3.5.: Figure shows the example of differences between local and global clustering in the neighbor-choosing process. The blue line in both plots is the threshold of global clustering. Red bars are actual neighbor pairs and green bars are non-neighbors. We can see that there are many red bars under the global threshold, but they are identical in local clustering.

number of clusters by using dendrograms. In our case, the number of clusters is two, whether they are neighbors or not. By using this clustering algorithm, we automatically define the threshold based on importance scores without making any statistical assumptions or using any information about the ground truth graph. One way of applying clustering to the weighted adjacency matrix of the predicted network is by applying global clustering to the weights, which finds a global threshold for the whole network. After the sensitivity analysis, if we combine all the vectors of importance in the matrix and fill the diagonal of the matrix with zero, we will get a weighted asymmetric matrix, $\hat{W}$, and we can apply clustering to this matrix after making it symmetric by $\frac{(\hat{W}+\hat{W}^T)}{2}$, and the resulted matrix will be the symmetric and weighted adjacency matrix, which contains the average effect of nodes on each other. Then, by applying clustering to the flatten upper-triangle part of the matrix, we can get a global threshold for the network, and based on the threshold, we can classify the nodes as neighbors or not of each other which will yield a binary and symmetric adjacency matrix. However, it gives a good approximation of the optimal threshold for the whole network, but in some cases, it can be sensitive to the 'outliers' in the weights. For example, when the sum of the importance of node i and node j is very far from the distribution of the other weights, this can cause the threshold to be higher, whereas other important pairs can be under these thresholds, which will lead to false negatives in the reconstruction of the network. You can review this case from the 3.5b. For this reason, we use local clustering instead of global clustering in our method. This means that instead of using single clustering based on the average importance of node pairs, we apply clustering to each vector of the

W individually. As opposed to the global method, we first classify the neighbors

Algorithm 1 Neural Sensitivity-Based Network Reconstruction(**NSNR**)

Require: X: Time-series data, SA: Sensitivity analysis method, N: Number of nodes
Ensure: A: Predicted adjacency matrix, $Models$: Trained models
Initialize $Models \leftarrow \{\}$
Initialize $Weights \leftarrow \{\}$
for $i \leftarrow 0$ to $N-1$ **do**
    Data preparation: $X^t_{-i}, Y^{t+k}_i \leftarrow prepare\_data(X, i)$
    Train model of node i: $f_i.train(X^t_{-i}\_train, Y^{t+k}_i\_train)$
    Calculate scores: $scores \leftarrow SA(X^t_{-i}\_test, Y^{t+k}_{-i}\_test)$
    Add scores to weight matrix: $Weights.append(scores)$
    Add trained model to Models: $Models.append(f_i)$
end for
Neighbor choosing: $labels \leftarrow clustering(Weights)$
Reconstruct adjacency matrix: $A \leftarrow reconstruct(Weights, labels)$
return A, $Models$

of each node independently and then combine the results of each clustering in a matrix $\hat{A}$ which is a binary and asymmetric adjacency matrix in this case. We fill the diagonal of $\hat{A}$ with zero to indicate each node's connection with itself after the clustering process to prevent the effect of zero weight on the clustering threshold. Then, because it is assumed to be an undirected network, we look at the unequal pairs of the $\hat{A}$ to convert $\hat{A}$ to a symmetric matrix A. It is important to consider the importance of both pairs on each other not to lose information about their relative functional connectivity and we look at the average importance of unequal pairs on each other as in the equation below:

$$A_{ij} \text{ and } A_{ji} = \begin{cases} 1, & \text{if } \frac{1}{2}(\hat{W}_{ij} + \hat{W}_{ji}) \geq \frac{c_i + c_j}{2} \\ 0, & \text{otherwise} \end{cases} \tag{3.16}$$

Where c_i and c_j are the local clustering thresholds for the weight vectors of node i and j respectively, and $\hat{W}$ is the weighted adjacency matrix. This selection ensures that an edge is assigned between unmatched pairs of $\hat{A}$ if their average scores are sufficiently large. Overall, by applying local clustering to each vector of the predicted weight matrix W, we ensure that the threshold is not sensitive to the "outlier" weights in the matrix and also do not lose the node's mutual importance by considering unmatched pairs' average scores. Applying clustering globally may not always be the problem for statistical measures such as nodes' correlation, where the weights are pair-wise and symmetric, but in experiments, we realized that in some dynamical models, this is a problem for them too, and applying local clustering like

in our method can improve their results. Neighbor choosing and reconstruction was the end of the whole process, and after this section, we get the output of our method as a predicted and symmetric $N \times N$ adjacency matrix A, which is assumed to be the true network structure predicted based on the time-series data generated by the ground truth graph during some period or periods with some unknown underlying dynamical rules. The main objective of the method was to infer the network structure, but at the end of the process, we get N trained predictors as output too, which we can use to predict node i's future state with its model f_i. You can review the entire method from Algorithm 1.

4 Experiments

In this chapter, we will introduce the experiments we conducted with our method in different settings to see the effectiveness and efficiency of the proposed work, as well as its limitations. We will focus on the performance of the method on different dynamical models and different types of graphs, the effect of data size on its performance, and its performance on different numbers of edges and nodes. We will consider the use of different time intervals, the effect of using more than a one-time step in prediction with LSTM model architecture, and the comparison of using the node itself in the prediction versus not using it. We will also see how the proposed method can reconstruct networks when the nodes are not homogeneous. However, we have limited our work to undirected graphs, we will experiment with the performance of the method on directed networks too. As mentioned before, we are using local clustering to find the neighbors, and in this section, we will see some comparisons to see the difference between local and global clustering. And comparison with different statistical baselines such as Granger causality and others mentioned in related works will be discussed.

4.1 Data generation

In our experiments, we have used synthetic data that is generated through simulation on different graphs with the dynamical models mentioned in Chapter 2. We have conducted simulations on the Barabási–Albert graph, the Erdős–Rényi graph, the random geometric graph, and the 2D-grid graph with different numbers of nodes.

- **Barabási–Albert graph**

  The Barabási–Albert (BA) [7] model is one of the scale-free network models that generate random graphs with growth and preferential attachment mechanisms. The growth concept means that it is adding nodes over time and that the size of the network in terms of the number of nodes increases over time. Preferential attachment makes the nodes that have a higher degree more likely to connect with new nodes. As we mentioned, the Barabási–Albert model is a scale-free network generator, which means that it follows the power law

and is suitable to cover many natural and human-made processes such as the internet, citation networks, or some social networks. By simulating dynamical models on this graph, we can approximate our method's performance on such systems.

- **Erdős–Rényi graph**

  The Erdős–Rényi model [18] generates random graphs. This model does not follow the power law and is not scale-free. This means that every node has the same probability of being connected with an edge to another node. If we notate the model $G(N, p)$, where N is the desired number of nodes, each edge that will connect these nodes has a probability of p to be present in the network independent of the presence of other edges. As the parameter p increases, the number of edges in the graph increases.

- **Random geometric graph**

  A random geometric graph is a network type where nodes are connected based on a given metric, such as the distance between nodes, and if the distance is greater than a given threshold, they are not connected with an edge. One of the specialties of random geometric graphs is that they are very similar to the human social network in that the nodes that share similarities tend to be easily grouped into one cluster, which is not the case in the previously mentioned mathematical graphs. More popular nodes are likely to be connected with other popular nodes in random geometric graphs [8].

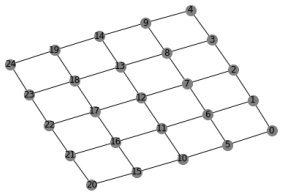
- **2D-grid graph**

  A 2D-grid graph also called a lattice graph or rectangular graph is kind of a finite part of an infinite graph where there are $m \times n$ nodes on a 2D surface (plane) and nodes are connected with their 4 closest neighbors maximum.

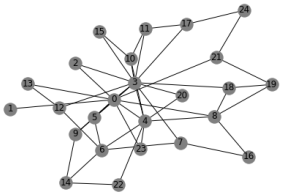
You can see the example graphs from Figure 4.1 for each mentioned graph type.

Simulation of dynamical models

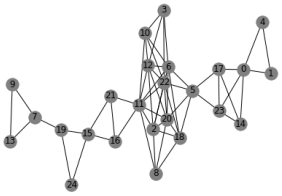
We have run several simulations over the mentioned graphs above with the dynamical rules of the Voter, SIS, Game of life, RPS, Forest fire, and CML models to get the desired number of samples for experimenting with our method. As mentioned before, for the dynamical models except for CML, we have used CMTC semantics and generated data over continuous time instead of discrete time points. In this case, to get enough information about the interactions, it is important to run long



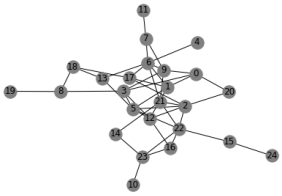
(a) 2D grid graph



(b) Barbasi-Albert graph



(c) Random geometric graph



(d) Erdos-Renyi graph

Fig. 4.1.: Example networks used in experiments

simulations because in each time unit only one rule of the dynamical model fires on a single node. If we take each step of the one simulation, there will be only one change in each next observation. Instead, we have used discrete sampling, which is defined by parameter k, and after each k step, we have collected the states of nodes. In that scenario, the number of changes between two-time observations becomes around k. The time passing between state transitions is not equal in continuous time models, but we try to approximate it with the parameter k, and in this way, we can collect enough state transitions in each simulation run to be able to learn dynamical rules more accurately. We have tried this process with three different values of k ($k = 5$, $k = 15$, and $k = 30$) to see the effect of sampling rate on learning the dynamical behaviors of nodes. It is the common property of simulations for all dynamical models except CML. Model-specific simulation characteristics can be reviewed below:

- **Voter**: We simulate the voter dynamics on networks until the desired number of observations are collected or the network reaches its consensus state, where all nodes agree and no more state transitions can happen. If that is not the case, we run each simulation until 1000 time steps are collected and then run another simulation with different initial states to collect more information about the evolution of the system.

- **SIS**: We apply the same structure of the data generation process in the SIS dynamical model too, as each simulation runs until 1000 steps have been collected or the infection has died in the network. The recovery rate and infection rate for all networks are $\beta = 1.0$ and $\mu = 2.0$, respectively.

- **Game of life**: There is no model-specific condition for a game of life dynamics, and simulations run until all network is dead or 1000 steps are collected.

- **RPS**: Same procedure is applied for the RPS model, and the change rate is 1.0 for all networks

- **Forest fire**: In the Forest fire model, growth rate $g = 1.0$, lightning rate $l = 0.1$, fire-spread rate $s = 2.0$, fire-extinct rate $e = 2.0$ for all networks.

- **CML**: Unlike the dynamical models mentioned above, we have used a different procedure for the CML dynamics. The first difference is that we have generated data in discrete time-space, and that is why we have not used sampling with a sampling rate α, instead, we take each discrete time observation. λ parameter is 3.5 and 3.8 for non-chaotic and chaotic cases, respectively. We observed that in non-chaotic dynamical behaviors of nodes, they tend to converge to a constant value or fluctuate within a small range after around 10–20 time steps, generally. To get more information, we reinitialize the states and run a new simulation after every 20 steps for the non-chaotic case, and apply the same procedure to the chaotic transitions too. You can see the difference between chaotic and non-chaotic CML in Figure 2.3.

Metrics

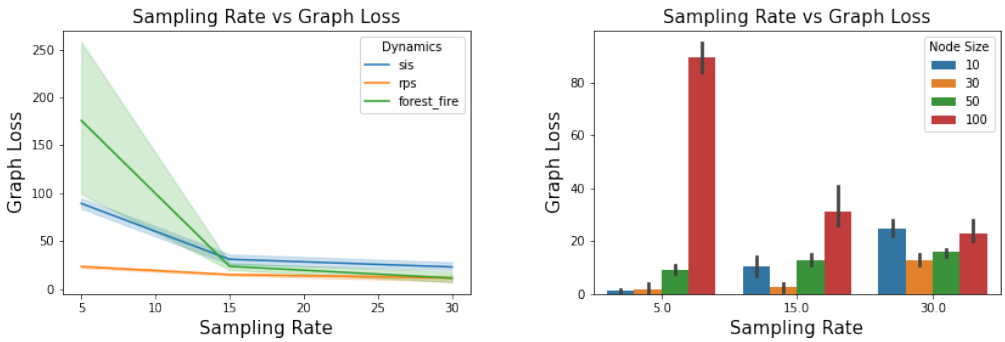
We have used the performance metrics mentioned below in the experiments to measure and compare the performance of our method:

- Accuracy: Accuracy is calculated based on the correctly classified entries of the adjacency matrix, which is a ratio of correct predictions to the total number of entries ($N \times N$).

- True positive rate is the ratio of the number of correctly predicted edges to the sum of correct and false predicted edges: $\frac{TP}{TP+FP}$

- False positive rate is the ratio of the number of false predicted edges to the sum of correctly predicted non-edges and false predicted edges: $\frac{FP}{TN+FP}$

- Graph loss or Graph distance is an L1 distance between the GT graph and the predicted graph: $\frac{\sum_{i,j}|A^*_{ij}-A_{ij}|}{2}$ Note that, we do not divide the sum by 2, in directed networks, in 4.6



(a) k vs Graph loss on 100 nodes Barabási–Albert graph with different dynamics

(b) k vs Graph loss on Barabási–Albert graph with SIS dynamics

Fig. 4.2.: Effect of sampling rate on network reconstruction.

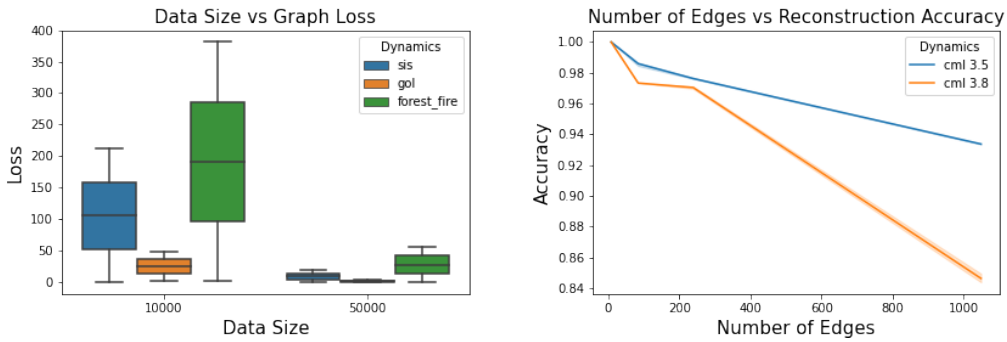
4.1.1 Experiment 1: Effect of sampling rate

In this part, we will evaluate the effect of the sampling rate k on the quality of reconstructed networks with different numbers of nodes and dynamical models. In this experiment, we have used the Barabási–Albert graph with the SIS, RPS, and Forest fire dynamical models and used data sizes of 50K in all combinations, which contain 50 samples with 1000 time steps. We observed based on the average results of all experimented [number of nodes-dynamics-k] combinations that, as the number of nodes increased, the sampling rate should be increased to get accurate reconstruction results. It was not unexpected; as we mentioned before, the rule fires on a single node at each time in continuous time models, and if the sampling rate is too big or too small proportional to the number of nodes in the network, we lose the effect of previous steps on next steps by waiting too long or too short to take the observation. For example, if there are 10 nodes in the graph and the sampling rate is 30 in simulation, it means that we collect data after every 30 changes, where nodes can change their states around 3 times until the observed time point, and the relationship between previously taken observations would be very low or broken. And this led to poor predictions of the next states, and accordingly to miss information about their effect on each other. We found empirically that the optimal sampling rate should be $k = \frac{N}{3}$ where the N is the number of nodes. You can see from Figure 4.2a that, in the Barabási–Albert graph which has 100 nodes, its reconstruction quality is measured by a graph loss which is the distance between the true adjacency matrix and the predicted matrix, decreases while the sampling rate increases for SIS, RPS, and Forest fire dynamics. Similarly, in the plot 4.2b, we can see that the graph loss has a negative linear relationship with the sampling rate on

100 nodes graph with SIS dynamics. These results are the average results of three SA methods, which are explained in Chapter 3.

4.1.2 Experiment 2: Effect of data size and number of edges

In this experiment, we tested our method's performance on 10,000 and 50,000-time observations to see how the number of observations affected the reconstruction quality. We have used Erdos–Rényi and Grid graphs with 50 and 49 nodes, respectively, and a sampling rate of 15. The dynamical models used in these experiments are SIS, Game of Life, and Forest Fire. Unsurprisingly, we observed that data size directly affects the quality of reconstruction for all types of graphs and dynamical models, where a large number of observations leads to better reconstruction quality. It is firstly related to the nature of deep learning architectures or any machine learning models in that the amount of data provides more information about the internal relationships, which makes the models better able to approximate true interactions. However, this being the case, the reconstruction quality of our method on 10,000 observations is still successful for most of the dynamical models and networks. You can see the experiment results in Figure 4.3, which describes the data size and reconstruction accuracy dependencies over discrete and binary dynamical models on networks. In the second part of the experiment, we checked the reconstruction performance of our method on different numbers of edges. We have tested our three SA methods on CML dynamics for both chaotic and non-chaotic cases over random geometric graphs up to 1049 edges. We observed that the accuracy of the predicted adjacency matrix gets below 90 percent after the number of edges is around 700 for both cases (chaotic and non-chaotic), and as expected, its results are generally better for the non-chaotic case than the chaotic one. It is generally the case that as the complexity of networks increases, the performance of many methods decreases, and one of the reasons for this in our method is related to the fixed structure of the MLP models. However, the parameters of the models (number of hidden units and number of hidden layers) are tried to be optimal, but as the number of nodes and edges increases, these pre-defined parameters may not be optimal choices for the specific case; they may underfit or overfit the underlying interactions based on the complexity of the graph. Secondly, as we apply sensitivity analysis to reveal the interactions of nodes, these interactions can be very complex for a specific node that has a lot of neighbors, such as in random geometric graphs, and few effects may not be identified by SA methods. But we can see that usually, our method can provide a reasonable approximation of the ground truth graph even when there are a large number of edges.



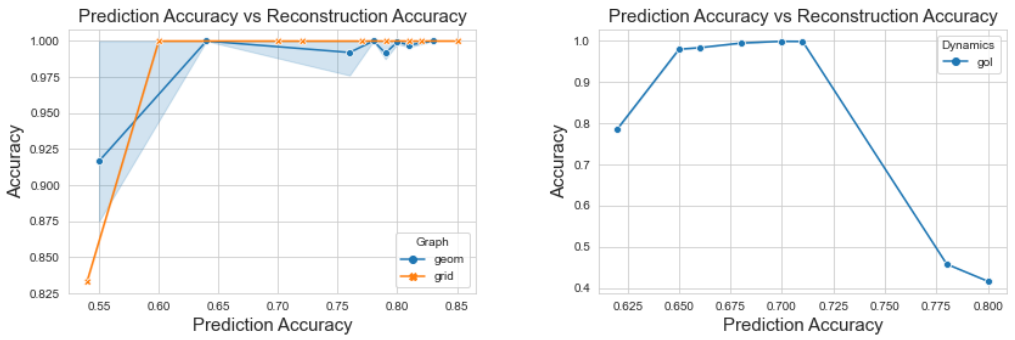
(a) Data size vs Graph Loss **(b)** Number of edges vs Accuracy on CML

Fig. 4.3.: Figures show the effect of data size on network reconstruction quality and method's performance related to network complexity

4.1.3 Experiment 3: Next state prediction and reconstruction quality

We tested our method in terms of prediction accuracy of the nodes' next state in this experiment. As we note that a node's future behavior depends on its neighborhood, it is assumed that the high prediction accuracy of a node's future states will yield better reconstruction accuracy where a node's neighbors are enough to predict its next states. We have experimented with our method on the grid and random geometric graphs with different numbers of nodes from 10 to 100 with the Voter dynamical model. When we look at the average scores of the experiments, prediction accuracy and reconstruction accuracy have a strong positive linear relationship. As our method trains an independent model for each node, and as a consequence, each node's model has different prediction accuracy, we take the average accuracy of all models as a unique dynamics prediction accuracy indicator of our method in this experiment. You can see from Figure 4.4a that the reconstruction accuracy of both graphs with voter dynamics increases as the average prediction accuracy increases. The datasets used in this experiment have different sampling rates (5, 15, and 30), where an inconvenient sampling rate yields low prediction accuracy and consequently low reconstruction accuracy. As you see from Figure 4.4b, It is not the case for a game of life dynamics on the Erdos–Rényi graph. We explored that the imbalance between the classes for each node is a reason for this scenario after analyzing the time series of the respective case. This means that nodes die too early during the simulation and most of the time steps are in the same state. In this case, our model learned only this particular state transition, so it can predict it with high accuracy, but it will not be a reliable prediction as it cannot learn the opposite case. This results in the overfitting of one class (state) and underfitting of the opposite class, resulting in training models that lack accurate information about

node interactions due to poor data quality. This is also the case for SIS dynamics, where infections die or spread too slowly because of their chosen infection and recovery rates. It means that it is very crucial to avoid models overfitting and to use balanced datasets where states are distributed equally over time to get more accurate network inference and learn dynamics better. It has to be noted that the prediction accuracy of the next states is not necessarily an indicator of the reconstruction quality in our work. As we do not use the node's own information in prediction, the accuracy of the next states may not be high for the dynamical models where the node's own information is important for the prediction, such as forest fire or CML. In our work, models are trying to predict without a target node as accurately as possible, but the main objective is to learn candidate nodes' interactions with and effects on the target node rather than getting the most accurate prediction. That is why the prediction accuracy may be around 55 to 60 percent, but the resulting reconstruction accuracy can be very sufficient. The crucial point is to monitor the validation loss during training and avoid the models from overfitting, which means that the generalization ability of the model is much more important than its prediction ability, where it can learn the pure interactions and effects coming from other nodes as accurately as possible. Briefly, prediction accuracy is not the crucial metric for reconstruction, but an increase in prediction accuracy without overfitting also yields an increase in reconstruction accuracy, especially in dynamical models such as Voter, where a node's next state is not dependent on its previous state. Using or not using a node's state is explained in more detail in the next experiment.



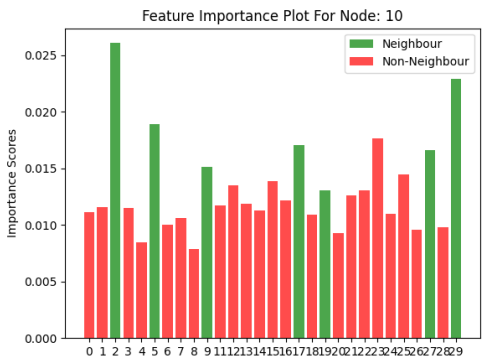
(a) Prediction accuracy vs reconstruction accuracy of Grid and Geometric graphs with Voter

(b) Prediction accuracy vs reconstruction accuracy of Erdos-Renyi graph with a Game of Life

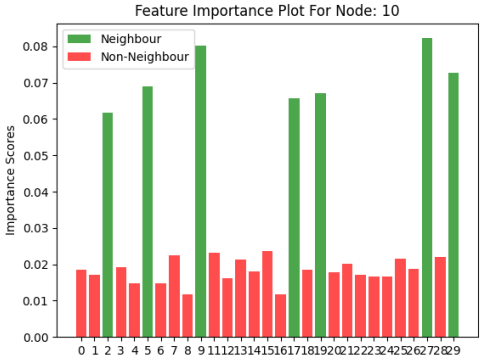
Fig. 4.4.: Effect of dynamics prediction on network reconstruction quality in terms of accuracy. Points on the lines in both plots indicate the independent experiment with different numbers of nodes and sampling rates

4.1.4 Experiment 4: Including node itself to its next state's prediction

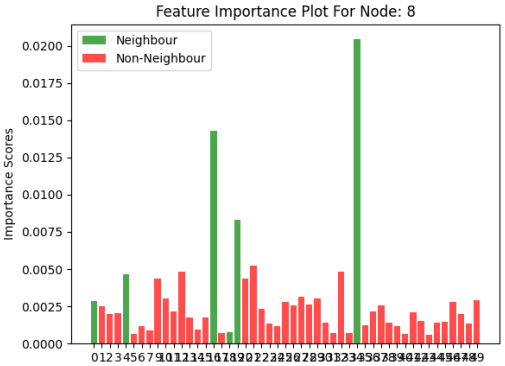
We try to identify which nodes affect the target node's future behavior more, and building a predictor that only depends on other nodes' individual and group effects would be a more suitable way to determine which of them have a crucial effect on the target node's future states. Using the node itself in the prediction may increase the prediction accuracy and help learn dynamics better, but it may decrease the reconstruction quality if the target node's information is enough for the prediction and the model mainly focuses on the node itself. In some dynamical models, such as Voter, the node's state at the next step is independent of its previous state, but in some dynamical models, such as SIS or Forest fire, the node's previous state has a crucial effect on its next state, which means that to be able to predict the next behavior of nodes, it is more suitable to use that information too to get high prediction accuracy. However, while it is critical for the accuracy of dynamics prediction for the mentioned dynamical models and learning dynamical rules, we discovered that it is not always the best option for inferring underlying network structures with SA. Because this high prediction accuracy will mainly depend on the contribution of the node itself and not give enough information about the effects of other nodes. Especially in models that have binary or discrete state spaces, state transitions happen based on specific rules, where these rules may include only the node's own information (e.g., recovery in the SIS model). This will lead to a model depending on the node itself and may lead to model overfit on training data, especially if the data mainly contains information about these specific scenarios, and the revealed prediction accuracy will not be interpretable in terms of other nodes' effects. For example, removing a neighbor or making small changes to the candidate neighbor will not make an identical difference, as the node itself will be enough for the model's prediction. We observed that, in most cases, removing the node itself from its prediction model generates more qualitative predictions about the structural connectivity, especially in terms of false positive edges. Including a node's own information in the model will decrease the effects of other nodes on prediction, and in the neighbor-choosing process, the difference between neighbors and non-neighbors may not be as distinguishable. For these reasons, to learn the pure effects of other nodes on a target node, we try to predict its next states without its own information, which leads us to have more interpretable models about other nodes' effects at the cost of losing more accurate dynamics learning and prediction accuracy. However, while this is the better option for the specified scenarios, we also lose the information that comes from the interaction effects of a node and its neighbor, which may be important. Especially in continuous state spaces, such as in the CML model, there are no specific rules; a node and its neighbors always decide



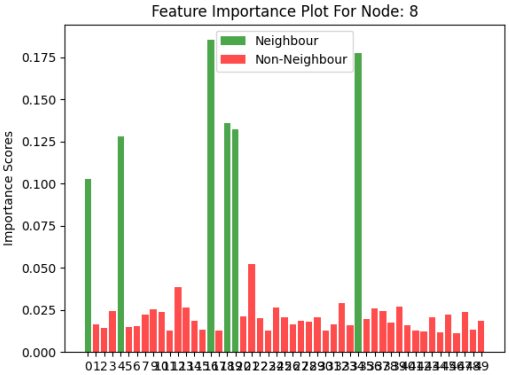
(a) Importance scores with the node itself for Forest Fire model



(b) Importance scores without node itself for Forest Fire model



(c) Importance scores without node for CML with continuous states($\lambda = 4.0$)



(d) Importance scores without node for CML with categorical states($\lambda = 4.0$)

Fig. 4.5.: Input permutation method on with Forest fire and CML dynamics 'with/without node itself' examples.

the node's next state together. In such cases, we can identify the neighbors more clearly if every state of a node only depends on its interaction with neighbors and not individual effects, because other nodes will not have an identical influence on the target node without the node itself. We have focused on mainly candidate neighbors' interactions with and effect on a target node in this work, but the opposite case may also be an option depending on the problem, where it is not possible to learn other nodes' effects without the node's previous information. In this experiment, we have checked these two cases to compare both of them in different scenarios. We have run our method with input permutation SA on a random geometric graph, an Erdos–Rényi graph, and a Barabási–Albert graph with 30 nodes with forest fire dynamics (where the node itself is important for its future behavior based on specific rules), and CML dynamics (where the next state of a node always depends on itself and its neighbors), with and without the node's previous states. You can see from Table 4.1 that the false positive rates are higher and consequently the loss is higher in the forest fire model for all networks when we include the node itself in

Effect of node itself(Forest fire)				
Graphs	Acc	TP	FP	Loss
Geom(with)	0.98	0.96	0.014	8
Geom(without)	**0.99**	**1.0**	**0.001**	**1**
ER(with)	0.99	**1.0**	0.005	2
ER(without)	**1.0**	**1.0**	**0.0**	**0**
BA(with)	0.98	**1.0**	0.015	6
BA(without)	**0.99**	0.95	**0.0**	**3**

Tab. 4.1.: Results of Input Permutation with/without node itself on Forest fire dynamics

Effect of node itself(CML)				
Graphs	Acc	TP	FP	Loss
Geom(with)	**1.0**	**1.0**	**0.0**	**0**
Geom(without)	0.98	0.96	0.01	7
ER(with)	**1.0**	**1.0**	**0.0**	**0**
ER(without)	0.99	0.99	0.007	4
BA(with)	**0.99**	**1.0**	**0.002**	**1**
BA(without)	0.98	0.87	0.005	9

Tab. 4.2.: Results of input permutation with/without node itself on CML dynamics

the model. Depending on the type of network and its complexity in terms of the number of nodes and edges, this difference increases. You can also see in Figure 4.5 the example of the effect of using a node's own history on the results of the Input Permutation method. It is clearly shown that there is quite a bit of difference between the two cases, as non-neighbors are very close to the actual neighbors in terms of importance scores, which leads to false positives in the clustering phase, but they are very identical in our application when we do not include the node itself. On the other hand, we can see from Table 4.2 that results are better when using the node itself in prediction for CML dynamics in all networks because of the reasons we mentioned above. We observed that when the chaotic behavior of the nodes is very high ($\lambda > 3.9$), prediction of the next state without the node itself is nearly impossible, and consequently network reconstruction quality is very low, but using the node's own information leads us to reconstruct the network with almost zero loss even when the parameter $\lambda = 4.0$. We tried to simplify the state space to check this case, such as by dividing continuous values into 5 equal bins based on their ranges, and tried to infer network structure as a classification problem instead of a regression problem. We got very good results even when the λ was set to 4.0 and network complexity was high, which tackled the problem of not using the node itself for CML. Also, the complexity of the regression model is higher than the classification model in our work, which has more hidden layers and hidden units to

be able to capture continuous state space, and by simplifying state space and using the classification model, we decrease the computational cost of the training too and get better reconstruction accuracy. You can see the difference in Figure 4.5d as an example of importance scores for the $8th$ node of the 50-node Barabási–Albert graph with chaotic ($\lambda = 4.0$) CML dynamics. In conclusion, depending on the dynamical model, both options may outperform each other, but overall, each of them allows us to reconstruct networks with sufficient accuracy. But we observed that except for the CML dynamics (if we do not manipulate the values), removing a node from its model generally gives similar or better results for other dynamical models, and by simplifying the state space in the continuous case (CML in our work), we can get accurate results too.

4.1.5 Experiment 5: Effect of local and global clustering on network reconstruction

We mentioned before that we are using local clustering after identifying the importance scores for the nodes, which means that we choose the neighbors of each node with clustering independently and then reconstruct the symmetric adjacency matrix based on unmatched pairs. In this experiment, we have checked the effect of using local or global clustering on the reconstruction quality, especially in terms of true positive and false positive rates, as they are more identical measures of the quality of assigned edges. We have experimented with two graphs, Barabási–Albert, and random geometric graphs, each with 30 nodes, and with two dynamical models, SIS and Game of Life. You can review the results in Table 4.3. As you can see, there

Effect of local and global clustering					
Dynamics	**Graphs**	Acc	TP	FP	Graph loss
SIS	Geom (global)	0.88	0.40	**0.0**	50
SIS	**Geom (local)**	**0.99**	**0.96**	**0.002**	**4**
GOL	Geom (global)	0.98	**1.0**	0.014	5
GOL	**Geom (local)**	**1.0**	**1.0**	**0.0**	**0**
SIS	BA (global)	0.98	0.88	**0.0**	7
SIS	**BA (local)**	**1.0**	**1.0**	**0.0**	**0**
GOL	BA (global)	0.94	0.5	**0.0**	28
GOL	**BA (local)**	**0.99**	**0.96**	0.01	**6**

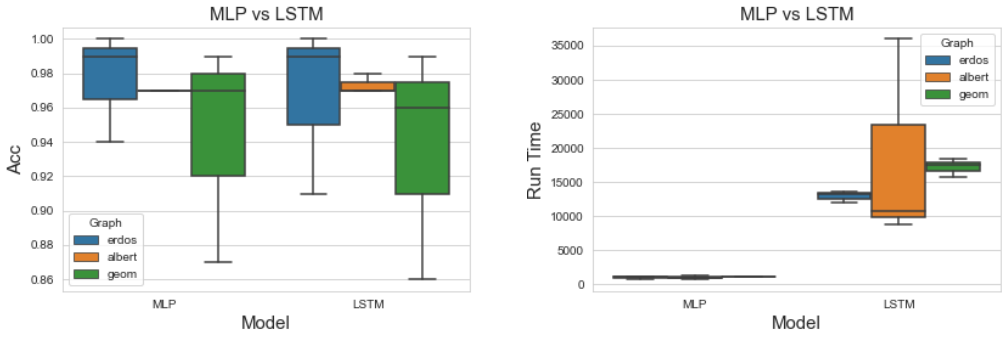
Tab. 4.3.: Results of applying local and global clustering

are significant differences between local and global clustering applications on true positive edges of predicted graph structure in most cases. This quality difference

may be larger when the network complexity in terms of the number of nodes and edges increases.

4.1.6 Experiment 6: Using more than one previous state for prediction

In our work, we focus on the prediction of the next states by using one previous state of other nodes, and we assume that one previous state contains all the necessary information for the prediction. But we tried to check if there would be any differences if we considered more previous time steps for the prediction. To check this, we have replaced the MLP models with LSTM (long-short-term memory) neural networks, as they are very powerful architectures for the prediction of sequential data, such as text or time series forecasting. They can remember the necessary sequential information in the memory cell of the network or forget the unnecessary inputs with the forget gate, and by this way they avoid the gradients being vanished during the optimization, and the algorithm is able to converge to the optimal point of the loss function, which is the main limitation of RNN networks, especially for long sequences. In our work, we have used one LSTM layer with 100 units, followed by two hidden layers and an output layer. We have used 10 previous steps, where the model tries to predict the next state of the node based on 10 previous states of other nodes. We have experimented with LSTMs on Erdos–Rényi, Barabási–Albert, and random geometric graphs with 50 nodes with Voter, SIS, and Game of life dynamical models. We observed that using sequential information in the prediction did not affect the average prediction accuracy and generated similar predictions as MLP models which were not unexpected as we use Markovian time models. For the SA part, we have used only the input permutation method, similar to the MLP case. Similarly, in terms of the reconstruction quality, we do not see that much difference between MLP and LSTM models for the used graphs and dynamical models; however, MLP seemed slightly better than LSTM in these experiments, which was expected because of their better interpretability. Finally, the run time of our method with LSTM models is very high compared to MLPs. And based on the results, we can get the same or even better reconstruction quality with MLPs by using only one previous piece of information in a very short amount of time compared to LSTMs. You can see the results of reconstruction accuracy for both models from the box plot 4.6. However, it would be better to use LSTMs when the dynamical models do not follow the Markov property.



(a) Reconstruction Accuracy for MLP and LSTM **(b)** Run time (in sec) for the MLP and LSTM

Fig. 4.6.: Comparison of the MLP and LSTM networks in terms of reconstruction accuracy and run time

4.1.7 Experiment 7: Heterogeneous nodes

We have experimented till now with dynamical models, where the nodes affect each other with similar behaviors which means that the rules (or the parameters) of the dynamical system are the same for each node. It is also the case in most of the related works mentioned before. In this experiment, we have tested our method with heterogenous nodes, where each node has its own specific behavior or it affects the other node in a different amount than others which is the case in most real-world scenarios. We have formulated the SIS and CML dynamical models on Erdos–Rényi and Barabási–Albert graphs with 30 nodes. For the SIS dynamics infection rate and recovery rate for the network are not the same for all nodes, instead, each node receives its infection and recovery rates uniformly from the range $[1.0, 2.0]$ and $[2.0, 5.0]$ respectively. The parameter ranges are chosen empirically to make sure that node's states are approximately balanced, which means that there are enough infected and susceptible cases to not face the imbalanced states and to make meaningful predictions. Similarly, λ is drawn from a uniform distribution in a range $[3.0, 4.0]$ for each node in CML dynamics, and as a consequence, some nodes' behavior is chaotic and other nodes are not. Experiments have been done with all three SA methods, and you can review the results from Table 4.4 and 4.5, for SIS and CML respectively.

We observed that in an SIS model with binary state space, all of the SA methods can achieve sufficient results in finding correct edges even when the nodes behave heterogeneously. Especially the input permutation method can identify the edges with almost 100 percent accuracy in both graphs, where we observed that there is no difference in whether the dynamics are homogeneous or heterogeneous. This is very similar to statistical methods those are identifying the relationships for each node

Graphs	Methods	Acc	TP	FP	Graphh Loss
B-Albert	Input.Perm	0.99	0.94	0.005	3
	Part.Derv	0.93	0.88	0.065	33
	DeepLift	0.92	0.84	0.065	35
	Corr	**1.0**	**1.0**	0.002	1
	MI	0.99	0.91	**0.0**	5
	Par.Corr	**1.0**	**1.0**	**0.0**	**0**
E-Renyi	Input.Perm	**1.0**	**1.0**	**0.0**	**0**
	Part.Derv	0.98	**1.0**	0.018	7
	DeepLift	0.98	**1.0**	0.015	6
	Corr	**1.0**	**1.0**	**0.0**	**0**
	MI	**1.0**	**1.0**	**0.0**	**0**
	Par.Corr	**1.0**	**1.0**	**0.0**	**0**

Tab. 4.4.: Results of methods on heterogenous SIS

Graphs	Methods	Acc	TP	FP	Graphh Loss
B-Albert	Input.Perm	0.94	0.66	0.01	25
	Part.Derv	**0.96**	0.73	**0.005**	**17**
	DeepLift	0.95	0.73	0.01	19
	Corr	0.84	**0.89**	0.17	72
	MI	0.90	0.75	0.07	43
	Par.Corr	0.95	0.75	0.018	21
E-Renyi	Input.Perm	**0.97**	0.82	**0.002**	**13**
	Part.Derv	0.94	0.71	0.01	23
	DeepLift	0.94	0.71	0.01	23
	Corr	0.76	**0.92**	0.27	108
	MI	0.90	0.83	0.09	47
	Par.Corr	0.96	0.88	0.03	20

Tab. 4.5.: Results of methods on heterogenous CML

separately which leads to successful results in heterogeneous nodes. Additionally, the linearity of the SIS model is another reason for the successful results of statistical methods. In continuous state space, which is CML dynamics, results were still sufficient for both graphs, even though the state space is more complex. However, the average results for CML dynamics are poorer than SIS dynamics. Differences in parameter λ make the nodes that have indirect relationships with the target node (such as a neighbor of the target node's neighbor) affect the prediction more if they have a sufficiently higher value of λ than its direct neighbor. As we rely on functional connectivity, it is not always possible to identify the direct and indirect effects in such cases. But we observed that our method outperforms statistical methods in heterogenous CML dynamics by a large amount. Overall, we can say that our method can identify the interactions between heterogeneous nodes with a

reasonable amount of accuracy which makes our approach a reasonable alternative for mentioned scenarios.

4.1.8 Experiment 8: Directionality

Our method is designed and limited for undirected networks in this work, where nodes that have edges between them affect each other in both directions interactively. However, it is limited with undirected networks, we tried to check if we can approximate the direction of the effects based on the importance scores by thinking of them as representing the weight of direction. For example, if we apply the SA method to the model of nodes i and j, and if node i is important for node j but node j is not for node i, we can assume that there is an interaction between them but only in one direction. Based on this assumption, we tried our method on a directed Erdos–Rényi graph with 25 nodes and with Voter and CML dynamics ($\lambda = 3.5$). For this, we ignore the symmetrization part of our method and just take the clustering results over the predicted importance score. Each row of the adjacency matrix of directed graphs contains information about affected nodes by the respective node, and for this reason, the predicted matrix after the clustering process should be transposed because in our method each row contains information about a node's neighbors. We have changed the formulation of dynamical models so that instead of neighbors of nodes, only their predecessors may affect their state transitions with the respective dynamical rules. You can see the results from Table 4.6. As you can see,

Dynamical Model	Methods	Acc	TP	FP	Graphh Loss
CML (non-chaotic)	Input.Perm	0.89	0.65	0.073	70
	Part.Derv	**0.89**	**0.70**	**0.069**	**67**
	DeepLift	0.89	0.70	0.069	67
Voter	Input.Perm	0.96	0.99	0.05	27
	Part.Derv	**0.96**	**0.99**	**0.04**	**24**
	DeepLift	0.96	0.99	0.04	24

Tab. 4.6.: Results of methods on directed 25 nodes Erdos-Renyi graph

accuracy and the true positive rates are quite sufficient for the voter dynamics, but the false positive rate is higher when compared with undirected networks in previous experiments. For the CML dynamics, results are worse than voter for all SA methods. Based on the experiments, we observed that the effects of the predecessors of a node are generally higher than those of its child nodes in most cases and can be selected by a clustering algorithm. But, similarly, their child nodes may be important in their predictions, as there is a relationship between their time series, especially when a

node is the predecessor of some nodes but has no predecessors, which is the reason for higher false-positive rates. As a result, we cannot say that our method cannot identify the direction of the interactions clearly, but it may be a good approximation for it depending on the dynamical system.

4.1.9 Experiment 9: Comparison with statistical baselines

In this section of the experiments, we have compared our method with the statistical model-free approaches mentioned in the related work section. We have tested correlation, partial correlation, and mutual information methods on the discrete state spaces based on the dynamical models mentioned above over various graph types. And additionally, we have experienced Granger causality for the continuous dynamical model CML. We have experimented with all statistical methods and our method with 50K data points, which contain 50 samples, and in each sample, there are 1000-time steps for discrete and binary dynamical models, and 2500 samples and 20-time steps in each sample for the CML dynamical model in both chaotic and non-chaotic dynamics. Note that 20 percent of the data is used for sensitivity analysis as test data, and 80 percent is used for training the models for dynamics learning. You can review the mathematical overview of baseline methods below:

- **Correlation**: The correlation method returns the correlation matrix of the nodes' time series, where Pearson's correlation coefficients of the node pairs are contained in the matrix. This method is very simple and fast and also can reveal very accurate approximations about the ground truth graph if the relationships between nodes are linear. However, in nonlinear cases, it is not an effective method, and it is not able to differentiate direct and indirect relationships. The correlation method calculates the correlation coefficients of the nodes based on their states at the same time step based on 4.1, which is a loss of significant information coming from previous states.

$$Corr(X_i, X_j) = \frac{Cov(X_i, X_j)}{\sqrt{Var(X_i)Var(X_j)}} \tag{4.1}$$

  Where X_i and X_j are the time series of node i and j.

- **Partial-correlation**: The Partial-correlation method calculates the node's dependencies not only pairwise but also considers the other nodes' effects, consequently, it can distinguish direct and indirect effects. This means that the

PC coefficient between node i and j is zero if and only if they are independent given all other nodes [6]. PC_{ij} is calculated based on 4.2:

$$PC_{ij} = -\frac{R_{ij}}{\sqrt{R_{ii}R_{jj}}} \tag{4.2}$$

Where Rij is the inverse of the covariance matrix of the node i and j

- **Mutual information**: In probability theory and information theory, MI is a measure of the relationship between two random variables based on the amount of information that is contained in one variable about another variable and can measure the non-linear relationships between two variables too. MI can be calculated for discrete variables by the formula given below:

$$I(X_i; X_j) = \sum_{i,j} P(x_i, x_j) log \frac{P(x_i, x_j)}{P(x_i)P(x_j)} \tag{4.3}$$

For continuous time series, we divide the time series of the nodes into 10 bins for the calculation of MI.

Similar to our method, these methods also return a weighted matrix that needs to be thresholded to select the possible edges. For this threshold and neighbor choosing, we apply the same procedure as in our method, to get more comparable results. However, we get a similar methodical approach for binary and continuous states, in discrete states like in Forest fire and RPS models, we apply integer labeling to the states such as $(0, 1, 2)$ which is not the suitable way which breaks the statistical dependencies as they are not ordinal variables, and this can be also counted as one of the limitations of statistical approaches in terms of dynamical models which have discrete state space. Used graphs in this experiment, are 10×10 grid graph $(N = 100, |E| = 180)$, Barabási–Albert graph $(N = 100, |E| = 196)$, Erdos–Rényi graph $(N = 50, |E| = 186)$ and random geometric graph $(N = 30, |E| = 84)$ where N is the number of nodes, and $|E|$ is the number of edges. The sampling rate α is approximately equal to $\frac{N}{3}$ for the datasets, where it is 30 for 100 nodes, 15 for 50 nodes, and 5 for 30 nodes graphs.

Voter

Most of the statistical methods failed in the reconstruction of the networks with the voter dynamical model, as is seen from Table 4.7. We observed that the reason for the case is the lack of ability to distinguish the direct and indirect effects coming from the nodes to a specific node, which is one of the main limitations of many

Graph	Method	TP	FP	Loss	Run Time
Grid	Corr	**1.0**	0.25	1196	**< 1**
	MI	**1.0**	0.13	628	17
	Par-Corr	**1.0**	**0.0**	**0**	2023
	NSNR (Inp.Permut)	**1.0**	**0.0**	**0**	2021
	NSNR (Par-Derv)	**1.0**	**0.0**	**0**	2447
	NSNR (DeepLift)	**1.0**	**0.0**	**0**	2021
E-Renyi	Corr	**1.0**	0.005	6	**<1**
	MI	0.99	0.003	5	5
	Par-Corr	**1.0**	**0.0**	**0**	194
	NSNR (Inp.Permut)	**1.0**	**0.0**	**0**	745
	NSNR (Par-Derv)	**1.0**	0.0009	1	998
	NSNR (DeepLift)	**1.0**	0.001	2	793
B-Albert	Corr	0.85	0.05	311	**<1**
	MI	0.77	0.03	165	16
	Par-Corr	**0.95**	0.006	41	2171
	NSNR (Inp.Permut)	0.82	0.001	41	1284
	NSNR (Par-Derv)	0.86	0.001	**36**	1702
	NSNR (DeepLift)	0.83	**0.0008**	37	1272
Geom	Corr	**1.0**	0.26	93	**<1**
	MI	0.96	0.08	32	1
	Par-Corr	**1.0**	**0.0**	**0**	30
	NSNR (Inp.Permut)	**1.0**	**0.0**	**0**	704
	NSNR (Par-Derv)	**1.0**	**0.0**	**0**	861
	NSNR (DeepLift)	**1.0**	**0.0**	**0**	738

Tab. 4.7.: Comparison of NSNR with statistical baselines on Voter dynamics

statistical approaches. As the correlation and mutual information, methods calculate the dependencies pair-wise without considering the other nodes' interactions, this leads them to fail in such cases where there is strong collinearity in the network. Especially in a grid graph, every node is connected indirectly, and this resulted in a large graph loss for correlation and mutual information methods. Also in the Voter model, a subset of the neighbors affect nodes together rather than individual effect, and it is one of the reasons why pair-wise methods can fail or generate less accurate networks. In all cases, our method, especially with input permutation SA, is getting very low graph loss and high accuracy metrics in terms of true positives and false positives. As well as partial correlation method is showing very similar results, because of its ability to detect the indirect and direct effects by calculating casualties given other nodes' interactions. We can say that our method with all three SA techniques is showing quite sufficient performance than statistical baselines for Voter dynamics.

SIS

Graph	Method	TP	FP	Loss	Run Time
Grid	Corr	**1.0**	0.003	17	**< 1**
	MI	**1.0**	**0.0**	**0**	19
	Par-Corr	**1.0**	**0.0**	**0**	2153
	NSNR (Inp.Permut)	**1.0**	**0.0**	**0**	1739
	NSNR (Par-Derv)	**1.0**	**0.0**	**0**	2179
	NSNR (DeepLift)	**1.0**	**0.0**	**0**	1730
E-Renyi	Corr	**1.0**	0.005	5	**<1**
	MI	0.95	**0.001**	11	4
	Par-Corr	1.0	0.003	**4**	245
	NSNR (Inp.Permut)	0.94	0.006	19	968
	NSNR (Par-Derv)	0.87	0.01	35	1219
	NSNR (DeepLift)	0.88	0.014	35	1038
B-Albert	Corr	0.91	0.0002	18	**<1**
	MI	0.87	**0.0**	25	18
	Par-Corr	**0.99**	0.0004	**3**	2097
	NSNR (Inp.Permut)	0.86	0.0002	28	1835
	NSNR (Par-Derv)	0.89	0.0002	21	2256
	NSNR (DeepLift)	0.90	0.0002	20	2008
Geom	Corr	1.0	0.006	5	**<1**
	MI	0.90	**0.0**	8	1
	Par-Corr	**1.0**	**0.0**	**0**	43
	NSNR (Inp.Permut)	0.96	0.002	4	517
	NSNR (Par-Derv)	0.96	0.002	4	671
	NSNR (DeepLift)	**1.0**	0.011	4	548

Tab. 4.8.: Comparison of NSNR with statistical baselines on SIS dynamics

For the SIS dynamical model on the different networks, our method shows reasonable results on average, where accuracy and true positive rates are generally over 90 percent, as well as the distance between the ground truth graph and the predicted graph, is very low. Surely, if there are more edges or nodes in the network, the results will not be the same as the experiment, but it would be a good approximation. We can see from Table 4.8 that the statistical methods are successful (even better sometimes) in the reconstruction of networks with SIS dynamics, and shows very similar results to our method. It is because of the linearity of the SIS model, where nodes interact with each other linearly as infected nodes infect their neighbors, and the distance between nodes makes the likelihood of infection less possible which does not increase the number of indirect effects.

Game of life

Graph	Method	TP	FP	Loss	Run Time
Grid	Corr	0.966	0.001	14	< 1
	MI	0.961	0.001	15	15
	Par-Corr	0.966	0.01	58	2044
	NSNR (Inp.Permut)	**1.0**	**0.0008**	**4**	2418
	NSNR (Par-Derv)	**1.0**	**0.0008**	**4**	2848
	NSNR (DeepLift)	**1.0**	**0.0008**	**4**	2407
E-Renyi	Corr	0.96	0.02	35	**<1**
	MI	0.91	0.008	24	15
	Par-Corr	0.77	0.05	97	2044
	NSNR (Inp.Permut)	**1.0**	**0.0**	**0**	947
	NSNR (Par-Derv)	1.0	0.001	2	1201
	NSNR (DeepLift)	0.99	0.004	6	988
B-Albert	Corr	0.65	0.008	110	**<1**
	MI	0.47	**0.001**	108	17
	Par-Corr	0.53	0.01	141	2233
	NSNR (Inp.Permut)	0.75	**0.001**	54	2285
	NSNR (Par-Derv)	0.83	0.0014	**39**	2721
	NSNR (DeepLift)	**0.84**	0.0020	40	2269
Geom	Corr	0.94	0.07	29	**<1**
	MI	0.83	0.03	24	1
	Par-Corr	0.65	0.22	107	44
	NSNR (Inp.Permut)	**1.0**	**0.0**	**0**	602
	NSNR (Par-Derv)	**1.0**	**0.0**	**0**	760
	NSNR (DeepLift)	**1.0**	**0.0**	**0**	634

Tab. 4.9.: Comparison of NSNR with statistical baselines on Game of Life dynamics

Our method with all sensitivity techniques outperforms the statistical approaches in Game of Life dynamics on all experimented networks. It is not surprising based on the formulation of the Game of Life dynamical model, as the relationship between nodes is not linear as in SIS dynamics. Each node can change its state based on the difference between the number of alive and dead neighbors, which means that the same neighbor can kill or make the node alive in different situations. The second reason is the crucial effect of the interaction of neighbors rather than individual effects on a specific node. However MI method can detect the non-linear relationships between nodes, it is a pair-wise measurement and does not consider the effect of the subset of alive or dead neighbors together. We can see from Table **??** that the MI method is slightly better than the other two statistical methods, but cannot reach our method in terms of reconstruction quality as we can detect non-linearities, interaction effects, and distinguish indirect effects.

RPS and Forest fire

Graph	Method	RPS			Forest Fire		
		TP	FP	Loss	TP	FP	Loss
Grid	Corr	**1.0**	0.055	267	**1.0**	0.005	25
	MI	**1.0**	0.0037	18	**1.0**	**0.0**	**0**
	Par-Corr	**1.0**	0.0018	9	**1.0**	0.0035	17
	NSNR (Inp.Permut)	**1.0**	**0.0002**	**1**	**1.0**	**0.0**	**0**
	NSNR (Par.Derv)	**1.0**	**0.0002**	**1**	**1.0**	**0.0**	**0**
	NSNR (Deeplift)	**1.0**	**0.0002**	**1**	**1.0**	**0.0**	**0**
E-Renyi	Corr	0.96	0.02	27	0.677	0.262	333
	MI	0.89	0.002	23	0.63	0.0009	69
	Par-Corr	0.83	0.028	61	0.82	0.335	382
	NSNR (Inp.Permut.)	1.0	0.0019	2	0.84	**0.025**	56
	NSNR (Par.Derv)	**1.0**	**0.0**	**0**	**0.86**	0.026	**54**
	NSNR (Deeplift)	**1.0**	**0.0**	**0**	0.81	0.065	102
B-Albert	Corr	0.95	0.096	467	0.84	0.016	107
	MI	0.79	0.0025	52	0.81	0.001	41
	Par-Corr	0.81	0.037	215	0.82	0.075	395
	NSNR (Inp.Permut)	**0.98**	**0.001**	**8**	**0.96**	**0.0**	**7**
	NSNR (Par.Derv)	0.97	0.0016	12	0.95	0.0	9
	NSNR (Deeplift)	0.97	0.0012	11	0.92	0.0	15
Geom	Corr	0.95	0.04	19	0.82	0.096	49
	MI	0.90	0.014	13	0.89	**0.0**	9
	Par-Corr	0.70	0.165	83	0.94	0.199	75
	NSNR (Inp.Permut)	**1.0**	**0.0**	**0**	**1.0**	0.002	**1**
	NSNR (Par.Derv)	**1.0**	**0.0**	**0**	0.95	0.002	5
	NSNR (Deeplift)	**1.0**	**0.0**	**0**	0.88	0.008	13

Tab. 4.10.: Comparison of NSNR with statistical baselines on RPS and Forest fire dynamics

We mentioned that in these two models, discrete states are treated as ordinal variables by statistical methods, which is one of the limitations related to the state space. Together with the complexity of the state space, in RPS and Forest fire models, dynamical rules are similar to the Game of Life, and a subset of neighbors makes the node change its state depending on its state, which makes the pair-wise methods inconvenient and creates non-linear dependencies between nodes' time-series. We observed similar results for both models, where our method can able to infer the underlying network structure near to zero graph loss which is much better than statistical approaches in those scenarios. Only in the Erdos–Rényi graph with the Forest fire model, our method's results are weaker compared to other networks, which may be caused by the sampling rate where most of the transitions are occurred by the effect of the node itself, not by its neighbor effects, and consequently, we have

not learned the node's effect more accurately, especially for the nodes which has quite several neighbors. Overall, we can see from Table 4.10 and 4.10 that results are quite sufficient regardless of the state space complexity of the networks.

CML

Graph	Method	$\lambda = 3.5$			$\lambda = 3.8$		
		TP	**FP**	**Loss**	**TP**	**FP**	**Loss**
Grid	Corr	**1.0**	0.068	329	**1.0**	**0.0**	**0**
	MI	**1.0**	0.0008	4	**1.0**	**0.0**	**0**
	Par-Corr	**1.0**	**0.0**	**0**	**1.0**	0.0002	**1**
	Granger.Cas	0.95	0.78	3773	0.55	0.38	1914
	NSNR (Inp.Permut)	**1.0**	**0.0**	**0**	**1.0**	**0.0**	**0**
	NSNR (Par.Derv)	**1.0**	**0.0**	**0**	**1.0**	0.0004	2
	NSNR (Deeplift)	**1.0**	**0.0**	**0**	**1.0**	0.0004	2
E-Renyi	Corr	**0.99**	0.0048	**6**	**1.0**	**0.0**	**0**
	MI	0.97	0.015	21	0.99	0.0038	5
	Par-Corr	0.97	**0.0009**	**6**	0.97	0.0048	10
	Granger.Cas	0.29	0.46	612	0.043	0.55	752
	NSNR (Inp.Permut.)	0.97	0.01	17	0.95	0.004	14
	NSNR (Par.Derv)	0.97	0.007	14	0.97	0.01	17
	NSNR (Deeplift)	0.97	0.007	14	0.98	0.01	16
B-Albert	Corr	0.87	0.087	439	**0.88**	0.029	165
	MI	0.79	0.0037	58	0.82	**0.001**	**39**
	Par-Corr	0.77	**0.001**	51	0.77	0.0018	53
	Granger.Cas	0.59	0.44	2213	0.55	0.38	560
	NSNR (Inp.Permut)	0.82	0.002	**46**	0.80	**0.001**	48
	NSNR (Par.Derv)	**0.89**	0.005	47	0.84	0.003	46
	NSNR (Deeplift)	0.86	0.004	48	0.85	0.003	45
Geom	Corr	**0.98**	0.12	44	**1.0**	0.002	**1**
	MI	0.95	0.019	11	0.98	**0.0**	**1**
	Par-Corr	0.97	**0.002**	**3**	0.96	0.011	7
	Granger.Cas	0.70	0.054	44	0.30	0.15	112
	NSNR (Inp.Permut)	**0.98**	0.011	6	0.92	0.017	12
	NSNR (Par.Derv)	0.96	0.01	7	0.94	0.019	12
	NSNR (Deeplift)	0.96	0.008	6	0.94	0.019	12

Tab. 4.11.: Comparison of NSNR with statistical baselines on CML dynamics

In our experiments with the CML dynamical model on networks, we have observed that our method is capable of accurately capturing the relationships among the nodes and determining the edges in most cases, even in highly chaotic environments. Depending on the nature of the logistics map function, however, the nodes' previous

states and next states are in a non-linear relationship, there is also a linear relationship between a node and its neighboring states at the current time step. As a result, most statistical methods show similar results to our method, with partial correlation outperforming other methods in most cases. This is because partial correlation takes into account the interactions between nodes as mentioned before, which is a crucial factor in CML dynamics. In contrast, the correlation method struggles to determine the edges between nodes accurately when the number of nodes and consequently the number of indirect effects is high, such as in Grid or Barabási-Albert graphs. We also tested the Granger causality method using the *netrd* library, which determines causality between nodes' previous and next states by fitting a linear model to the data. This method fits a linear model with only the node's previous states to predict its next state and another model with both the node's and another node's previous states. By comparing the errors of the two models, it determines causality between the nodes, which is similar in terms of application to our method. However, it is limited by its reliance on a simple linear function and its inability to capture the nonlinear dependencies between previous and next states, as well as it does not consider the interactions, calculate the differences pair-wise, which is a significant loss of information, and very difficult to include all possible combinations of interactions to the linear model, where MLPs do it automatically. We can see the results from Table 4.11.

However, our method is showing sufficient results in CML dynamics too, we lose information that comes from the node itself in this model as we mentioned before in previous experiments, where the node's states give significant information about its interactions with other nodes. We tested that, when we include the node itself in the prediction, our reconstruction loss is almost zero in all networks in both chaotic and non-chaotic scenarios.

Summary

To conclude, we observed that through all experiments and comparisons with the statistical baselines, our method can infer the structural connectivities of the nodes based on their functional connectivities with high accuracy in most of the scenarios. We could reconstruct the networks with different numbers of nodes and dynamical rules, regardless of the linear or non-linear relationships, chaotic or non-chaotic environments, as well as when nodes have homogeneous or heterogenous behaviors during the observed time. We saw that while the interaction effects, non-linearities, or undirect effects were the problems for statistical methods, we could tackle those limitations with NSNR. We should consider that all of the experiments (except CML)

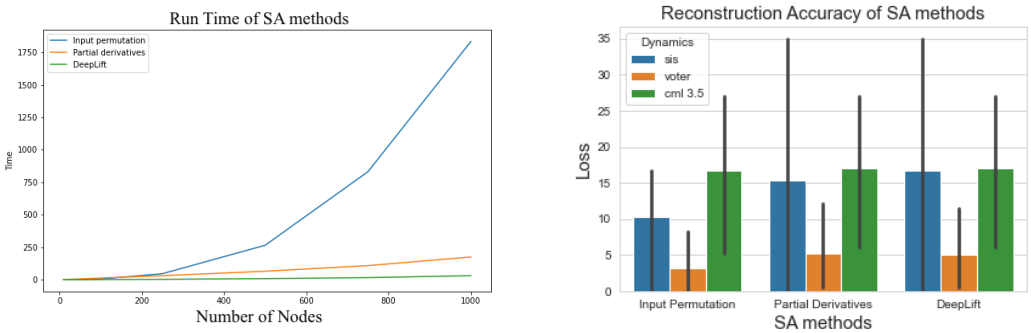
were based on continuous time models which can be more challenging than discrete time models in terms of predictability as the dependencies can be broken between time steps based on the chosen sampling rate, but our method could reveal sufficient results in most of the scenarios for reconstructing various type of networks.

5 Discussion

In this chapter, we will discuss our method's overall performance and its advantages and disadvantages over related approaches given in the related works. We will compare the performances of three commonly used sensitivity analysis methods in terms of their capability of finding edges between nodes based on trained models and their computational efficiencies. One more mentioned topic will be the scalability of our method. We will talk about possible improvements and future work that can be done related to our work.

5.0.1 Performance of SA methods

Through the thesis, we have given detailed information about the three SA methods used in our work and about their implementations. As well as their performances in different scenarios. Based on those facts, we can give overall information about them individually in the context of network reconstruction. We have given detailed information about the input permutation method SA, which we use to approximate the effect of a feature (node) when it is included or not included in the model. Through the experiments, we observed that the input permutation method generates more accurate results in most cases in terms of network reconstruction accuracy than the other two methods. As we permute the entire input and calculate how output probabilities vary, it can approximate the global sensitivity of the respective



(a) Run time of SA methods based on network size

(b) Graph loss for SA methods

Fig. 5.1.: Comparison of the SA methods in terms of reconstruction accuracy and run time

input feature better than DeepLift and partial derivatives, which are more suitable for measuring sensitivities or importance locally. Secondly, while gradient-based approaches approximate a model as a linear function, input permutation is model-agnostic, generating a more accurate approximation if the relationship between input and output is non-linear. We permute each feature only once, and this can underestimate the importance of a node in specific scenarios, such as when the states are not equally distributed for binary or discrete state spaces. However, it would give better results if we tried the method on different subsets to be able to capture internal interactions; it is not the optimal choice in terms of computational efficiency. We observed that the input permutation method is generally faster in small and medium-sized networks; the time complexity of the method is increasing exponentially with the network size as we apply it $N(N-1)$ times in the whole reconstruction process, where N is the number of nodes. Consequently, we train N models, and each model has $N-1$ input variables. For large networks, it is not efficient to use because of this drawback if we do not have enough computational power. However, while the local interpretation ability of the partial derivatives and DeepLift is successful, they are not always good estimators of the global importance of the feature for output as input permutation method. As we mentioned in Chapter 3, partial derivatives also suffer from saturation and sometimes fail to observe that the relationship between input and output is non-linear, while DeepLift can handle this issue by considering how output changes while input changes from its reference instead of considering an infinitesimal change in input. We can observe from the experiment results that, in most cases, input permutation outperforms these two SA methods in global approximation. But the main advantage of DeepLift and partial derivatives before input permutation is their time complexity. The calculation of the importance scores for each node is done by backpropagation, and if the model structure is fixed in terms of hidden layers and several neurons, the time complexity is increasing linearly while the number of nodes increases for both methods. To conclude, in small and medium-sized networks, input permutation is giving more accurate results and is efficient in terms of computation. On the other hand, DeepLift and partial derivatives can be more convenient methods to use if the network is large in terms of run time, but the network reconstruction accuracy is not guaranteed to be successful if the prediction model of the node is highly non-linear and contains a high degree of complex interactions. You can review the performance of the SA methods in Figure 5.1 based on run time and reconstruction accuracy. Note that the training time of the models is not included, and the results are only for the run time of the SA part of our method. For exploring the performance of the method in terms of run time, we have experimented on the Erdos-Renyi network with SIS dynamics, and have used only 20-time observations because of computational expenses, on

5.1b we have tested SA methods on 50 nodes in all mentioned networks with Voter, SIS, and CML($\lambda = 3.5$) dynamics, where all samples have 50K time observations.

5.0.2 Scalability, advantages, and limitations

We observed from the experiments that our method has better performances in all types of graphs, especially in identifying non-linear relationships and group interactions compared to statistical baselines. Especially in small and medium-sized networks, we can have good reconstruction accuracy. However, this is not the case in most real-world scenarios, where there are more complex and large networks and more complicated dynamical rules. This requires a more complex architecture of the predictor models, which will be able to capture the interactions, and consequently, more efficient SA methods in terms of time and accuracy to have more robust results. For linear dynamical models such as SIS, statistical methods are showing better performance. As well as for the run time, statistical methods are much more efficient than ours and are much more optimal to use if the network is very large and the dynamical rules are generally linear (which is not the case in real-world situations). More specifically, we have addressed some of the limitations of traditional methods, such as the linearity assumption of dynamical rules, considering interactions rather than pair-wise effects, considering time information, dealing with indirect effects to some extent, the possibility of reconstruction in heterogeneous behaviors, and also the possible alternative of identifying the direction of the relationships in directed networks without any ground truth information. We can say that our method can be a good estimator and an alternative approach to statistical methods for capturing the more complex interactions and identifying structural connectivity. As well, our predictors (dynamics learners) are very simple models compared to other deep learning frameworks for prediction-based network reconstruction, which can be an alternative option to reduce the computational requirements of those frameworks. Based on the results, we see that we can infer the underlying structure of the social networks, natural processes (such as Barbasi-Albert, random geometric graphs that are good representations of them), as well as chaotic physical systems and evolving systems with sufficient accuracy in small and medium-sized networks. On the other hand, we cannot ensure the robustness of our method in bigger networks, as the number of nodes and consequently the interactions among them will be more complex, which will not be clearly identifiable by the SA methods we are using, especially in more complex state spaces and chaotic behaviors. For example, if some neighbors are much more effective at prediction, the prediction model is more likely to be optimized based on those nodes, while other less effective neighbors

may appear as non-neighbors during the clustering operation. We observed that this is the main problem for partial derivatives and the DeepLift method when we use the node itself in the prediction if it is the most important node for prediction. Secondly, we mentioned that not using the node itself can reveal less accurate results in specific cases, which makes it difficult to trust the reconstruction if it depends on the prediction models, which have lower accuracy. It would be a better choice to check both options and select the most identical result based on feature importance plots, where we are more confident in classifying differences based on scores. It is recommended to use input permutation SA when we include the node itself in the prediction to get more robust results. One other problem is the complexity of the state space. We saw that our approach is much more successful in dynamical models with simple state spaces, such as Voter, Game of Life, or RPS. It is getting worse in chaotic and continuous states like in CML, and we have observed that we can get better results if we simplify those kinds of spaces to lower dimensions. In this work, we have tested our work with Markovian time models, where one previous step is enough for the prediction, and MLP is enough for those kinds of time series. But it is not limited, as if the future behaviors of the nodes depend on the sequential information rather than single previous information, one can replace the MLP with RNN frameworks, and with the input permutation (as it is model-agnostic), one can reveal the structural connectivities of the networks, which surely will require more time and computational power. However, this is the case, and we have challenged ourselves with the continuous time models in our work where we approximate the previous states with a sampling rate and still get good results, which can be more successful if dynamical models are discrete-time models where dependencies between time lags are more identical. Furthermore, SA methods are used in our work to help us find important nodes and reconstruct networks; they also give us a chance to interpret the interactions based on feature importance plots, which means that our approach is not only limited to finding an edge between nodes but also provides us with meaningful insights about the relationships. But we mainly limited ourselves to simple sensitivity approaches to need less computational resources; more advanced global SA methods can reveal more accurate results about the interactions and better capture non-linear and group interactions. Finally, this method is not completely model-free, such as correlation, as the reconstruction accuracy may depend on the hyperparameters of the models for a specific problem type. Whereas the MLP's architecture in terms of several hidden layers and units, learning rate, and other hyperparameters should be optimized to avoid overfitting and obtain models with greater generalization power. However, we have provided a default model structure that one should optimize to get more accurate results if it is not a good fit for the data in hand. Briefly, we explained some advantages and

disadvantages of our method in different scenarios. And can conclude that, we can count our approach as a good alternative to the traditional prediction-based and statistical methods, can tackle some limitations of them, also may have limitations in more complex problems such as computational complexity, scaling, and additional parameter tuning.

5.0.3 Future work

After the detailed explanation of the method we have applied for network reconstruction and its advantages and disadvantages, we can give some directions that can be taken related to our work to improve the current approach or address different aspects of the work that are not included in the thesis. One of the possible improvements that can be made is related to the implementation of the method. As we proposed a framework where all MLP models are trained independently in a sequence, one can implement the joint training of the MLPs to reduce computational expenses. This can be done by assigning a common loss function, where all models are optimized to minimize the overall loss instead of their independent losses, and an appropriate sensitivity analysis method can be chosen to measure the importance of nodes, which is similar to the related work [51], where they are measuring the casualty with regularization techniques on weights. But, it can also bring some limitations, such as identifying heterogeneous behaviors. Secondly, future work can be done for improving the work to identify the direction of the relationships, where we would be able to reconstruct directed networks with high accuracy too. As previously mentioned, more advanced global sensitivity analysis methods can be investigated, such as Sobol SA (especially in continuous state spaces) [15], together with ways of reducing computational time, which may lead to more appropriate and robust results about the individual and group interactions of the nodes and also better capture non-linearities. In our work, we mainly focus on approximating the total effect of nodes on each other based on the whole subset of nodes, but one can investigate the nodes' effects on each other based on their contribution of them in different subsets with the shapely values method from game theory [43]. However, this is a computationally expensive method; the SHAP method [36] for the machine learning and deep learning models provides efficient approximations of the shapely values and can be used in the network reconstruction process to distinguish individual and interaction effects and may provide more accurate results. Related to the scalability issues, dimensionality reduction techniques can be investigated if they may provide an accurate approximation of the node's importance, which may also reduce the computational time in more extensive networks.

6 Conclusions

In this work, we addressed the reconstruction of networks' underlying structure based on their components' time series data without ground truth information about the network and its dynamical rules governing the system. We tried to propose a prediction-based approach by using deep learning and tackle the basic limitations of traditional methods. We used sensitivity analysis methods to infer the underlying relationships between inputs and the target node of the respective predictor model which is built on the purpose to predict the target node's future state based on other nodes' previous state.

Our hypothesis in this work was "nodes that affect the respective nodes' future behavior are more likely to be its direct neighbors", and this paradigm lead us to find important features of the models and assign them as neighbors of the predicted node, which revealed the whole structure of the network. By using deep learning, we could capture the group interactions, non-linear effects, and other basic limitations of traditional methods.

Furthermore, we explained and analyzed the three different sensitivity analysis techniques in that manner, explaining their advantages and disadvantages in the network reconstruction problem. We also tried to address the heterogeneous behaviors of the nodes, which is the case in most real-world scenarios. As well as, we analyzed our approach in different scenarios, such as the node's effect on the prediction of its next state and how it helps us in terms of finding its direct neighbors. We used local clustering to turn the weighted adjacency matrix into a binary and symmetric matrix, which lead us to reconstruct indirect networks without any supervision. We also provided and tested alternative models such as LSTM to give appropriate predictions in non-markovian dynamical models where sequential information is needed and a model-agnostic way of analyzing their important features. We tested our method on continuous time models and tested the effect of different sampling rates on prediction accuracy and reconstruction accuracy. We also tried to reconstruct a directed network with our approach. In all cases, we observed successful reconstruction accuracy, as well as analyzed and explained the limitations of the approach by comparing it with different methods, together with giving some directions that can be done in the future related to this topic.

In conclusion, we observed that our hypothesis about the network structure gave us

enough information to reconstruct the networks' structure. In terms of the sensitivity analysis, we could conclude that methods such as input permutation which provide better approximation about the global behaviors of the inputs are more appropriate than local interpretation methods in network reconstruction to capture interactions, and non-linear relationships, and better able to distinguish additional effects which reduce the false-positives in reconstruction.

Bibliography

[1] https://github.com/julianzimmerlin/network-reconstruction/blob/master/Thesis_Main.pdf. [Online; Last accessed 02-May-2023] (cit. on pp. 21, 25).

[2] http://proceedings.mlr.press/v70/shrikumar17a/shrikumar17a-supp.pdf. [Online; Last accessed 02-May-2023] (cit. on p. 33).

[3] https://github.com/gerritgr/GINA. [Online; Last accessed 25-April-2023] (cit. on p. 83).

[4] https://github.com/joshgun11/Network_reconstruction_Thesis. [Online; Last accessed 02-May-2023] (cit. on p. 84).

[5] Vida Abedi, Raquel Hontecillas, Adria Carbo, et al. *Chapter 8 - Multiscale Modeling: Concepts, Technologies, and Use Cases in Immunology*. Academic Press, 2016, pp. 145–173. DOI: 10.1016/B978-0-12-803697-6.00008-4. URL: https://www.sciencedirect.com/science/article/pii/B9780128036976000084 (cit. on p. 26).

[6] Ildefons Magrans de Abril, Junichiro Yoshimoto, and Kenji Doya. "Connectivity inference from neural recording data: Challenges, mathematical bases and research directions". In: *Neural Networks* 102 (2018), pp. 120–137. DOI: https://doi.org/10.1016/j.neunet.2018.02.016. URL: https://www.sciencedirect.com/science/article/pii/S0893608018300704 (cit. on pp. 7, 8, 56).

[7] Réka Albert and Albert-László Barabási. "Statistical mechanics of complex networks". In: *Rev. Mod. Phys.* 74 (1 Jan. 2002), pp. 47–97. DOI: 10.1103/RevModPhys.74.47. URL: https://link.aps.org/doi/10.1103/RevModPhys.74.47 (cit. on p. 39).

[8] A. Antonioni and M. Tomassini. "Degree correlations in random geometric graphs". In: *Phys. Rev. E* 86 (3 Sept. 2012), p. 037101. DOI: 10.1103/PhysRevE.86.037101. URL: https://link.aps.org/doi/10.1103/PhysRevE.86.037101 (cit. on p. 40).

[9] Niklas Boers, Bedartha Goswami, Aljoscha Rheinwalt, et al. “Complex networks reveal global pattern of extreme-rainfall teleconnections”. In: *Nature* 566.7744 (2019), pp. 373–377. DOI: 10.1038/s41586-018-0872-x. URL: https://doi.org/10.1038/s41586-018-0872-x (cit. on p. 3).

[10] Léon Bottou. *Online Algorithms and Stochastic Approximations*. Ed. by David Saad. revised, oct 2012. Cambridge, UK: Cambridge University Press, 1998. URL: http://leon.bottou.org/papers/bottou-98x (cit. on p. 22).

[11] Saiful Bukhori. “Forest Fire Model”. In: *Forest Fire*. Ed. by Janusz Szmyt. Rijeka: IntechOpen, 2017. Chap. 4. DOI: 10.5772/intechopen.72591. URL: https://doi.org/10.5772/intechopen.72591 (cit. on p. 16).

[12] Maosen Cao, Nizar F. Alkayem, Lixia Pan, and Drahomír Novák. *Advanced Methods in Neural Networks-Based Sensitivity Analysis with their Applications in Civil Engineering*. IntechOpen, 2016. Chap. 13. DOI: 10.5772/64026. URL: https://doi.org/10.5772/64026 (cit. on p. 26).

[13] Yehuda Dar, Vidya Muthukumar, and Richard G. Baraniuk. *A Farewell to the Bias-Variance Tradeoff? An Overview of the Theory of Overparameterized Machine Learning*. 2021. DOI: 10.48550/ARXIV.2109.02355. URL: https://arxiv.org/abs/2109.02355 (cit. on p. 23).

[14] Laura L. Elo, Henna Järvenpää, Matej Orešič, Riitta Lahesmaa, and Tero Aittokallio. “Systematic construction of gene coexpression networks with applications to human T helper cell differentiation process”. In: *Bioinformatics* 23.16 (2007), pp. 2096–2103. DOI: 10.1093/bioinformatics/btm309 (cit. on p. 35).

[15] Thomas Fel, Remi Cadene, Mathieu Chalvidal, et al. “Look at the Variance! Efficient Black-box Explanations with Sobol-based Sensitivity Analysis”. In: (2021). DOI: 10.48550/ARXIV.2111.04138. URL: https://arxiv.org/abs/2111.04138 (cit. on p. 68).

[16] Sebastian Gerwinn, Jakob Macke, and Matthias Bethge. “Bayesian inference for generalized linear models for spiking neurons”. In: *Frontiers in Computational Neuroscience* 4 (2010). DOI: 10.3389/fncom.2010.00012. URL: https://www.frontiersin.org/articles/10.3389/fncom.2010.00012 (cit. on p. 7).

[17] Muriel Gevrey, Ioannis Dimopoulos, and Sovan Lek. “Review and comparison of methods to study the contribution of variables in artificial neural network models”. In: *Ecological Modelling* (2003), pp. 249–264. DOI: 10.1016/S0304-3800(02)00257-0. URL: https://www.sciencedirect.com/science/article/pii/S0304380002002570 (cit. on p. 27).

[18] E. N. Gilbert. “Random Graphs”. In: *The Annals of Mathematical Statistics* 30.4 (1959), pp. 1141–1144. DOI: 10.1214/aoms/1177706098. URL: https://doi.org/10.1214/aoms/1177706098 (cit. on p. 40).

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016, pp. 164–172 (cit. on p. 22).

[20] C. W. J. Granger. “Investigating Causal Relations by Econometric Models and Cross-spectral Methods”. In: *Econometrica* 37.3 (1969), pp. 424–438. URL: http://www.jstor.org/stable/1912791 (visited on Mar. 7, 2023) (cit. on p. 9).

[21] Gerrit Großmann and Luca Bortolussi. *Reducing Spreading Processes on Networks to Markov Population Models*. 2019. DOI: 10.48550/ARXIV.1906.11508. URL: https://arxiv.org/abs/1906.11508 (cit. on pp. 13, 14).

[22] Gerrit Großmann, Julian Zimmerlin, Michael Backenköhler, and Verena Wolf. *GINA: Neural Relational Inference From Independent Snapshots*. 2021. DOI: 10.48550/ARXIV.2105.14329. URL: https://arxiv.org/abs/2105.14329 (cit. on pp. 10, 14, 83).

[23] Silvia de Haan-Rietdijk, Manuel C. Voelkle, Loes Keijsers, and Ellen L. Hamaker. “Discrete- vs. Continuous-Time Modeling of Unequally Spaced Experience Sampling Method Data”. In: *Frontiers in Psychology* 8 (2017). DOI: 10.3389/fpsyg.2017.01849. URL: https://www.frontiersin.org/articles/10.3389/fpsyg.2017.01849 (cit. on p. 13).

[24] L Harrison, W.D Penny, and K Friston. “Multivariate autoregressive modeling of fMRI time series”. In: *NeuroImage* 19.4 (2003), pp. 1477–1491. DOI: https://doi.org/10.1016/S1053-8119(03)00160-5. URL: https://www.sciencedirect.com/science/article/pii/S1053811903001605 (cit. on p. 7).

[25] Richard A. Holley and Thomas M. Liggett. “Ergodic Theorems for Weakly Interacting Infinite Systems and the Voter Model”. In: *The Annals of Probability* 3.4 (1975), pp. 643–663. DOI: 10.1214/aop/1176996306. URL: https://doi.org/10.1214/aop/1176996306 (cit. on p. 14).

[26] Vân Anh Huynh-Thu and Guido Sanguinetti. *Gene regulatory network inference: an introductory survey*. 2018. DOI: 10.48550/ARXIV.1801.04087. URL: https://arxiv.org/abs/1801.04087 (cit. on p. 3).

[27] E. M. Izhikevich, J. H. Conway, and A. Seth. “Game of Life”. In: *Scholarpedia* 10.6 (2015). revision #150735, p. 1816. DOI: 10.4249/scholarpedia.1816 (cit. on p. 15).

[28] Ameya D. Jagtap and George Em Karniadakis. *How important are activation functions in regression and classification? A survey, performance comparison, and future directions*. 2022. DOI: 10.48550/ARXIV.2209.02681. URL: https://arxiv.org/abs/2209.02681 (cit. on p. 22).

[29] Fikret Emre Kapucu, Jarno M. A. Tanskanen, Francois Christophe, Tommi Mikkonen, and Jari A. K. Hyttinen. "Evaluation of the effective and functional connectivity estimators for microelectrode array recordings during in vitro neuronal network maturation". In: *EMBEC & NBC 2017*. Ed. by Hannu Eskola, Outi Väisänen, Jari Viik, and Jari Hyttinen. Singapore: Springer Singapore, 2018, pp. 1105–1108 (cit. on p. 9).

[30] Wasiur R. KhudaBukhsh, Arnab Auddy, Yann Disser, and Heinz Koeppl. *Approximate lumpability for Markovian agent-based models using local symmetries*. 2018. DOI: 10.48550/ARXIV.1804.00910. URL: https://arxiv.org/abs/1804.00910 (cit. on p. 13).

[31] Sunyong Kim, Seiya Imoto, and Satoru Miyano. "Dynamic Bayesian network and nonparametric regression for nonlinear modeling of gene networks from time series gene expression data". In: *Biosystems* 75.1 (2004). Computational Systems Biology, pp. 57–65. DOI: https://doi.org/10.1016/j.biosystems.2004.03.004. URL: https://www.sciencedirect.com/science/article/pii/S0303264704000383 (cit. on p. 8).

[32] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. *Neural Relational Inference for Interacting Systems*. 2018. DOI: 10.48550/ARXIV.1802.04687. URL: https://arxiv.org/abs/1802.04687 (cit. on p. 10).

[33] Mikhail Krechetov. *Game of Life on Graphs*. 2021. DOI: 10.48550/ARXIV.2111.01780. URL: https://arxiv.org/abs/2111.01780 (cit. on p. 16).

[34] Ellen Kuhl. "The classical SIS model". In: *Computational Epidemiology: Data-Driven Modeling of COVID-19*. Cham: Springer International Publishing, 2021, pp. 33–40. DOI: 10.1007/978-3-030-82890-5_2. URL: https://doi.org/10.1007/978-3-030-82890-5_2 (cit. on p. 15).

[35] Tim Loossens, Francis Tuerlinckx, and Stijn Verdonck. "A comparison of continuous and discrete time modeling of affective processes in terms of predictive accuracy". In: *Scientific Reports* 11.1 (2021), p. 6218. DOI: 10.1038/s41598-021-85320-4. URL: https://doi.org/10.1038/s41598-021-85320-4 (cit. on p. 13).

[36] Scott Lundberg and Su-In Lee. *A Unified Approach to Interpreting Model Predictions*. 2017. DOI: 10.48550/ARXIV.1705.07874. URL: https://arxiv.org/abs/1705.07874 (cit. on pp. 33, 68).

[37] Feng Luo, Yunfeng Yang, Jianxin Zhong, et al. "Constructing gene co-expression networks and predicting functions of unknown genes by random matrix theory". In: *BMC Bioinformatics* 8 (2007), pp. 299–299 (cit. on p. 35).

[38] Robert M. May and George F. Oster. "Period doubling and the onset of turbulence: An analytic estimate of the Feigenbaum ratio". In: *Physics Letters A* 78.1 (1980), pp. 1–3. DOI: https://doi.org/10.1016/0375-9601(80)90788-4. URL: https://www.sciencedirect.com/science/article/pii/0375960180907884 (cit. on p. 18).

[39] George Michailidis and Florence d'Alché-Buc. "Autoregressive models for gene regulatory network inference: Sparsity, stability and causality issues". In: *Mathematical Biosciences* 246.2 (2013), pp. 326–334. DOI: https://doi.org/10.1016/j.mbs.2013.10.003. URL: https://www.sciencedirect.com/science/article/pii/S0025556413002393 (cit. on pp. 7, 8).

[40] Md Mehedi Hassan Onik, Shakhawat Ahmmed Nobin, Adnan Ferdous Ashrafi, and Tareque Mohmud Chowdhury. *Prediction of a Gene Regulatory Network from Gene Expression Profiles With Linear Regression and Pearson Correlation Coefficient*. 2018. DOI: 10.48550/ARXIV.1805.01506. URL: https://arxiv.org/abs/1805.01506 (cit. on p. 8).

[41] J. Pizarroso, J. Portela, and A. Muñoz. *NeuralSens: Sensitivity Analysis of Neural Networks*. 2020. DOI: 10.48550/ARXIV.2002.11423. URL: https://arxiv.org/abs/2002.11423 (cit. on p. 30).

[42] Lukasz Romaszko. "Signal Correlation Prediction Using Convolutional Neural Networks". In: *Proceedings of the Neural Connectomics Workshop at ECML 2014*. Ed. by Demian Battaglia, Isabelle Guyon, Vincent Lemaire, and Jordi Soriano. Vol. 46. Proceedings of Machine Learning Research. PMLR, 15 Sep 2015, pp. 45–56. URL: https://proceedings.mlr.press/v46/romaszko15.html (cit. on p. 9).

[43] Benedek Rozemberczki, Lauren Watson, Péter Bayer, et al. *The Shapley Value in Machine Learning*. 2022. DOI: 10.48550/ARXIV.2202.05594. URL: https://arxiv.org/abs/2202.05594 (cit. on p. 68).

[44] Jakob Runge, Jobst Heitzig, Vladimir Petoukhov, and Jürgen Kurths. "Escaping the Curse of Dimensionality in Estimating Multivariate Transfer Entropy". In: *Phys. Rev. Lett.* 108 (25 June 2012), p. 258701. DOI: 10.1103/

PhysRevLett.108.258701. URL: https://link.aps.org/doi/10.1103/PhysRevLett.108.258701 (cit. on p. 9).

[45] Shaeke Salman and Xiuwen Liu. *Overfitting Mechanism and Avoidance in Deep Neural Networks*. 2019. DOI: 10.48550/ARXIV.1901.06566. URL: https://arxiv.org/abs/1901.06566 (cit. on p. 23).

[46] K. Sasirekha and P. Baby. "Agglomerative Hierarchical Clustering Algorithm-A Review". In: 2013 (cit. on p. 35).

[47] Alex Sherstinsky. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404 (2020), p. 132306. DOI: https://doi.org/10.1016/j.physd.2019.132306. URL: https://www.sciencedirect.com/science/article/pii/S0167278919305974 (cit. on p. 20).

[48] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning Important Features Through Propagating Activation Differences". In: (2017). DOI: 10.48550/ARXIV.1704.02685. URL: https://arxiv.org/abs/1704.02685 (cit. on pp. 32, 33).

[49] Antonio Sutera, Arnaud Joly, Vincent Franois-Lavet, et al. "Simple Connectome Inference from Partial Correlation Statistics in Calcium Imaging". In: *Neural Connectomics Challenge*. Springer International Publishing, 2017, pp. 23–36. DOI: 10.1007/978-3-319-53070-3_2. URL: https://doi.org/10.1007%2F978-3-319-53070-3_2 (cit. on p. 8).

[50] György Szabó and Gábor Fáth. "Evolutionary games on graphs". In: *Physics Reports* 446.4 (2007), pp. 97–216. DOI: https://doi.org/10.1016/j.physrep.2007.04.004. URL: https://www.sciencedirect.com/science/article/pii/S0370157307001810 (cit. on p. 16).

[51] A. Tank, I. Covert, N. Foti, A. Shojaie, and E. B. Fox. "Neural Granger Causality". In: *IEEE Transactions on Pattern Analysis amp; Machine Intelligence* 44.08 (Aug. 2022), pp. 4267–4279. DOI: 10.1109/TPAMI.2021.3065601 (cit. on pp. 7, 9–11, 68).

[52] Raul Vicente, Michael Wibral, Michael Lindner, and Gordon Pipa. "Transfer entropy—a model-free measure of effective connectivity for the neurosciences". In: *Journal of Computational Neuroscience* 30.1 (2011), pp. 45–67. DOI: 10.1007/s10827-010-0262-3. URL: https://doi.org/10.1007/s10827-010-0262-3 (cit. on p. 9).

[53] Tatsuo Yanagita. "Phenomenology of boiling: A coupled map lattice model." In: *Chaos* 2 3 (1992), pp. 343–350 (cit. on p. 17).

[54] Jian-Bo Yang, Kai-Quan Shen, Chong-Jin Ong, and Xiao-Ping Li. "Feature selection for MLP neural network: the use of random permutation of probabilistic outputs". In: *IEEE transactions on neural networks* 20.12 (2009), pp. 1911–1922. DOI: 10.1109/tnn.2009.2032543. URL: https://doi.org/10.1109/TNN.2009.2032543 (cit. on pp. 27, 28).

[55] Zhang Zhang, Yi Zhao, Jing Liu, et al. "A general deep learning framework for network reconstruction and dynamics learning". In: *Applied Network Science* 4.1 (2019), p. 110. DOI: 10.1007/s41109-019-0194-4. URL: https://doi.org/10.1007/s41109-019-0194-4 (cit. on pp. 2, 3, 10, 18).

[56] Jie Zhou, Ganqu Cui, Shengding Hu, et al. *Graph Neural Networks: A Review of Methods and Applications*. 2018. DOI: 10.48550/ARXIV.1812.08434. URL: https://arxiv.org/abs/1812.08434 (cit. on p. 9).

List of Figures

List of Tables

Appendix

A

A.1 Implementation and experimental details

Model architecture

In our work, the MLP model of each node is in the same structure as below:

- Classification model: Input layer, 3 Hidden layers, Output layer

  Input layer - Number of nodes
  1st hidden layer - 100 neurons
  2nd hidden layer - 64 neurons
  3rd hidden layer - 32 neurons
  Output layer - Number of possible states

  The activation function is ReLu for each hidden layer and Softmax for the output layer.

- Regression model: Input layer, 7 Hidden layers, Output layer

  Input layer - Number of nodes
  1st hidden layer - 128 neurons
  2nd hidden layer - 128 neurons
  3rd hidden layer - 128 neurons
  4th hidden layer - 128 neurons
  5th hidden layer - 128 neurons
  6th hidden layer - 64 neurons
  7th hidden layer - 64 neurons
  Output layer - 1 neuron

  The activation function is ReLu for each hidden layer and Linear activation on the output layer.

For the LSTM models:

- Classification and Regression: Input layer, 1 LSTM layer, 2 Hidden layers, Output layer

  Input layer - Number of nodes
  LSTM layer - 100 neurons
  1st hidden layer - 32 neurons
  2nd hidden layer - 16 neurons
  Output layer - Number of possible states

  The activation function is ReLu for each LSTM and hidden layer, and Softmax for the output layer in classification, Linear in regression.

Parameters are chosen with random search empirically based on all used dynamical models. These structures are for a single model. If there are N nodes, there are N models.

Training

We mentioned before that prediction models are trained independently in a sequence, where one model finishes training and the other starts, and each has its own independent weights and loss function. Details below are for the training procedure of a single model:

- Classification MLP: Number of maximum epochs = 100
  Early stopping patience = 10
  Optimization algorithm = SGD
  Loss function = Categorical cross entropy
  Learning rate = 0.01
  Batch size = 64
  Train-Test split = 80-20

- Regression MLP: Number of maximum epochs = 100
  Early stopping patience = 10
  Optimization algorithm = SGD
  Loss function = MSE
  Learning rate = 0.01
  Weight decay = 0.001
  Momentum = 0.9
  Batch size = 64
  Train-Test split = 80-20

- Classification LSTM: Number of maximum epochs = 50
  Early stopping patience = 5
  Optimization algorithm = SGD
  Loss function = Categorical cross entropy
  Learning rate = 0.01
  Batch size = 64
  Train-Test split = 80-20

- Regression LSTM: Number of maximum epochs = 50
  Early stopping patience = 5
  Optimization algorithm = SGD
  Loss function = MSE
  Learning rate = 0.01
  Batch size = 64
  Train-Test split = 80-20

While the maximum number of epochs is defined, based on the complexity of the data, models can stop training early to prevent overfitting, and consequently, the run time of the whole process is not the same for each data even if the size is the same.

MLP models are built on *Pytorch*, and LSTM models on *Keras*. Experiments have been conducted on a *MacBook Pro* with processor: 1,4 GHz. Quad-Core Intel Core i5, and memory: 16 GB 2133 MHz.

Data generation

For generating synthetic data with the dynamical rules, the code base of the GINA [22] has been used for all dynamical models. For the CML dynamics, the code has been modified to get continuous values. You can visit the *github link* [3] for the code.

SA implementation

- Input permutation: For the implementation of the input permutation, we have used our own implementation. For the permutation, we have used the uniform permutation method of *Numpy*. We apply a single permutation to each feature of the test data.

- Partial derivatives: To calculate the partial derivatives of the output values with respect to the input values, we have used *Autograd* package of *Pytorch*.

- DeepLift: For the implementation of the DeepLift, we have used *Captum* library with linear and re-scale rules.

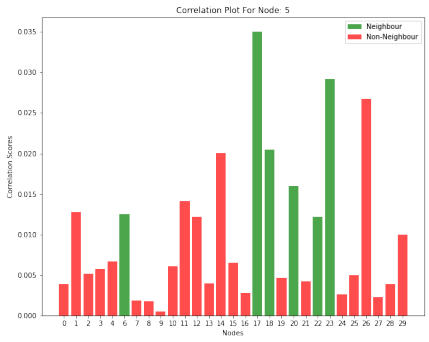
Baselines

Correlation, Partial correlation, Mutual Information, and Granger casualty baselines are implemented with *Netrd* library. Not that, resulted weights are taken with absolute values, and for thresholding, clustering locally is applied to the weight matrices as same in our method.

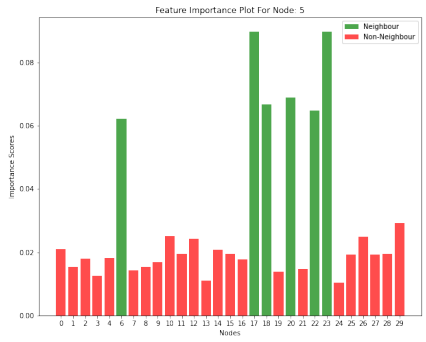
You can review our code base and implementation from the GitHub page of the work [4].

A.2 Additional visualizations

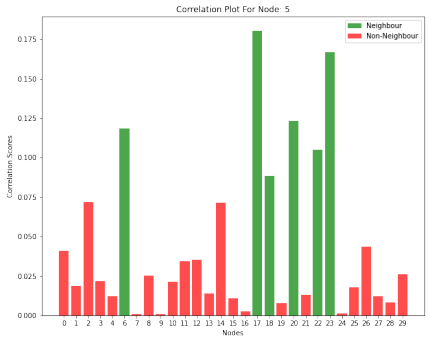
In this section, you can review the differences between our method and statistical method (Correlation) weights for example nodes visually, some visualizations about the relationship between the distance to the target node and importance scores, and some reconstructed network examples in the next pages.



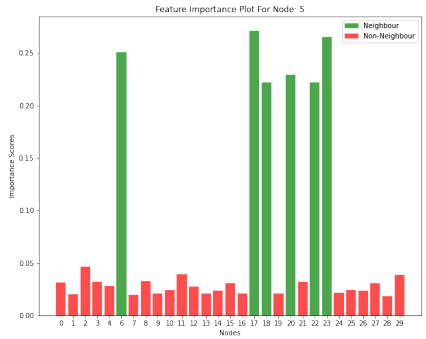
(a) Correlation scores for Forest Fire model



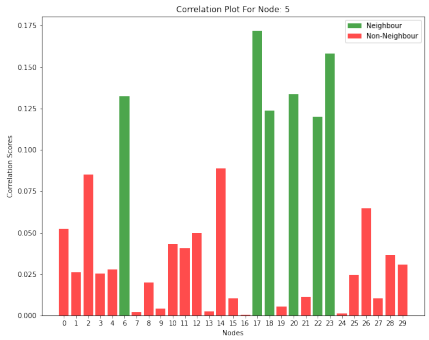
(b) Importance scores for Forest Fire model



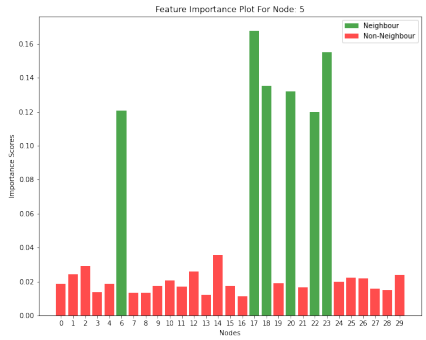
(c) Correlation scores for RPS model



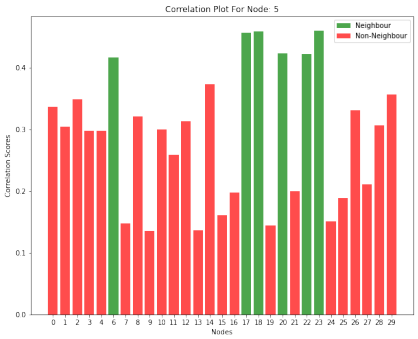
(d) Importance scores for RPS model



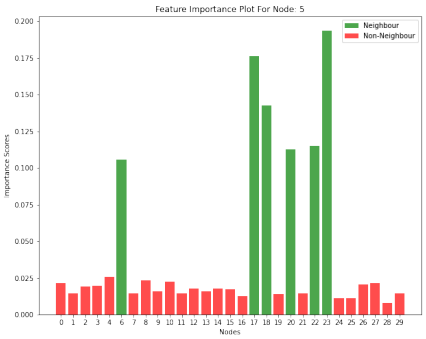
(e) Correlation scores for Game of Life model



(f) Importance scores for Game of Life model

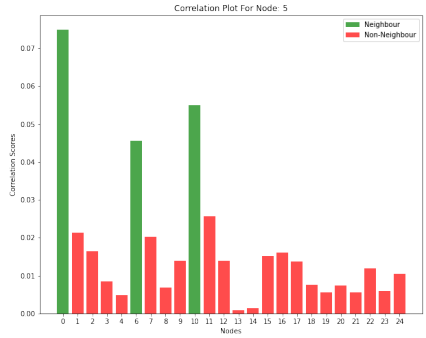


(g) Correlation scores for Voter model

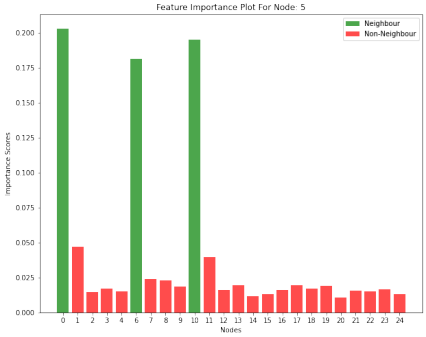


(h) Importance scores for Voter model

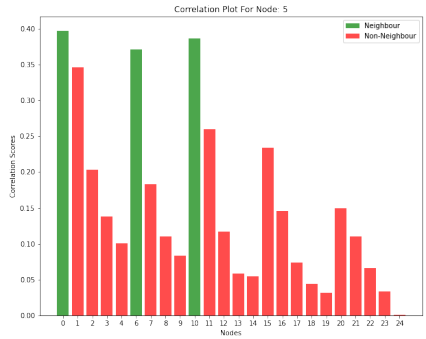
Fig. A.1.: Comparison of importance scores with correlation scores on 30 nodes random geometric graph



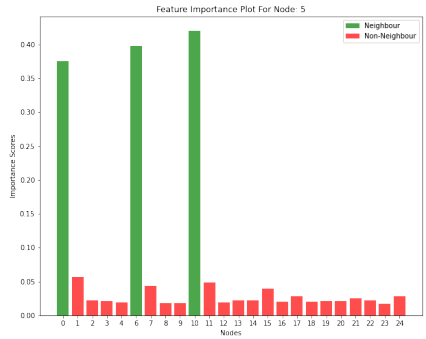
(a) Correlation scores for Forest Fire model



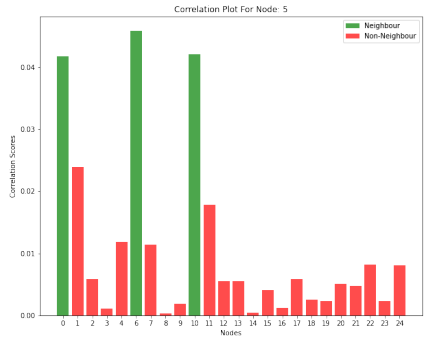
(b) Importance scores for Forest Fire model



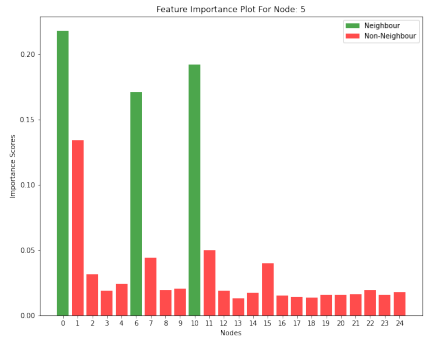
(c) Correlation scores for RPS model



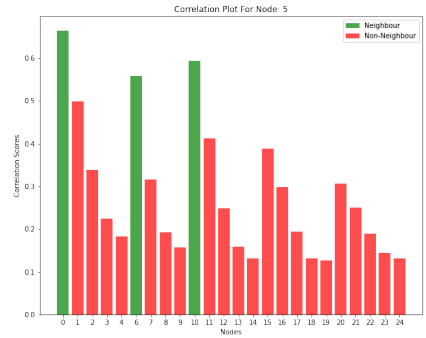
(d) Importance scores for RPS model



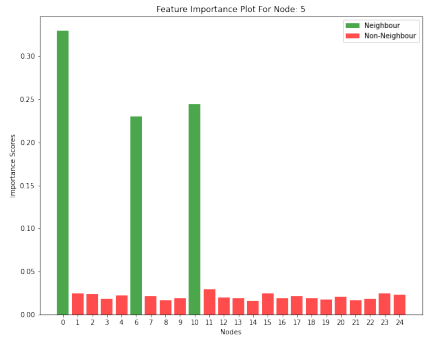
(e) Correlation scores for Game of Life model



(f) Importance scores for Game of Life model



(g) Correlation scores for Voter model



(h) Importance scores for Voter model

Fig. A.2.: Comparison of importance scores with correlation scores on 25 nodes Grid graph

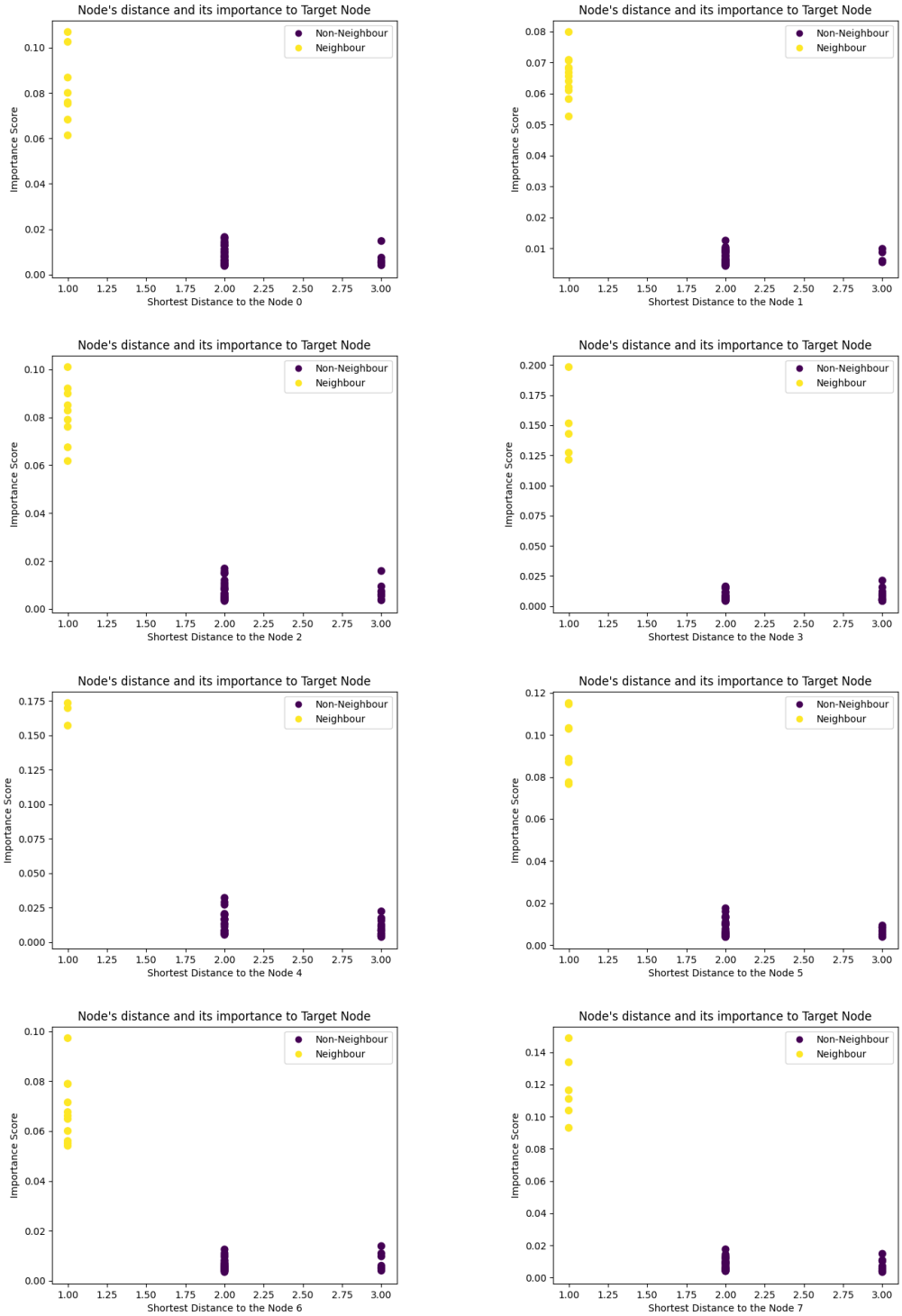


Fig. A.3.: Relationship between distance to node and importance scores

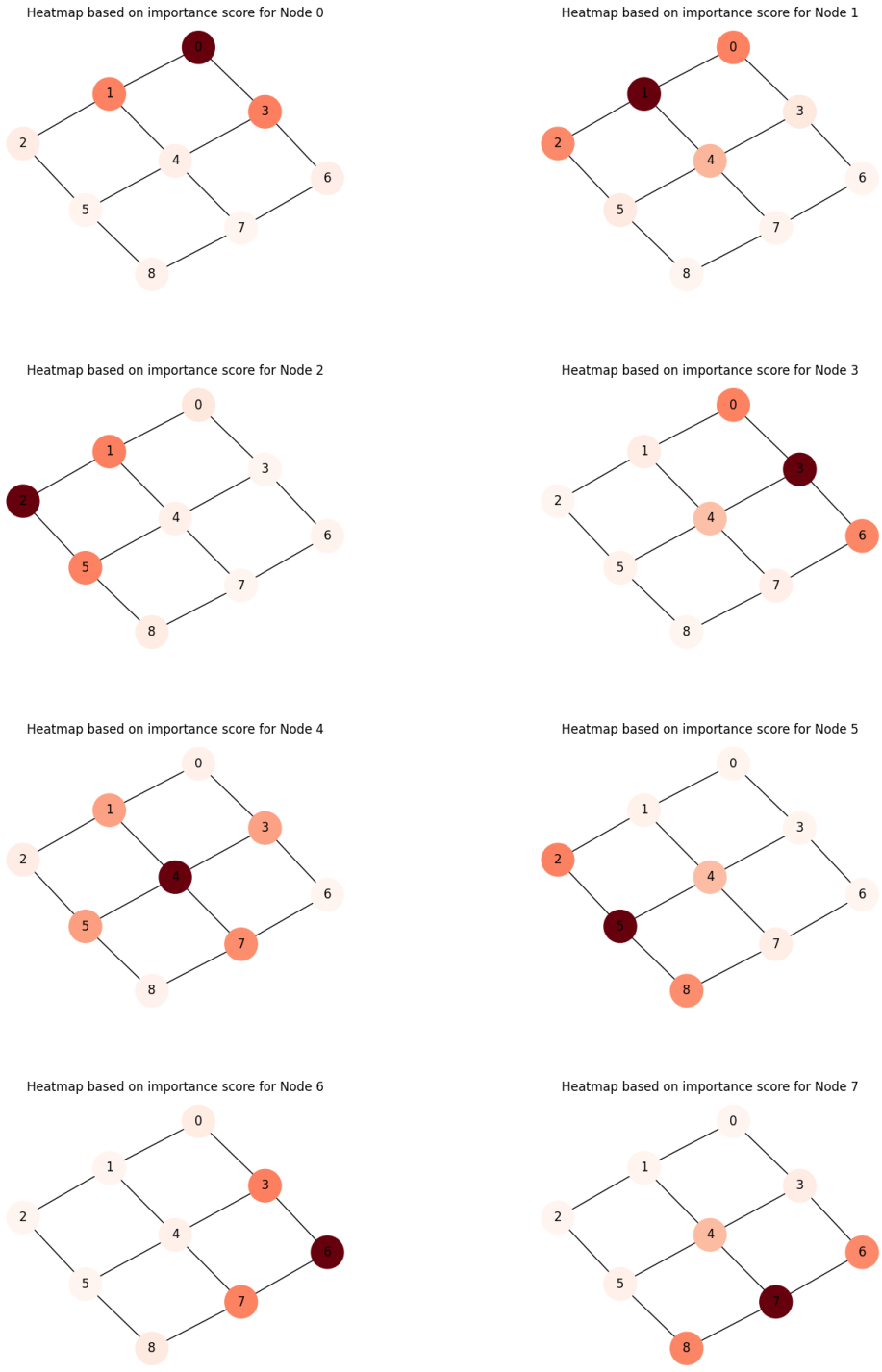


Fig. A.4.: Heatmap of importance scores

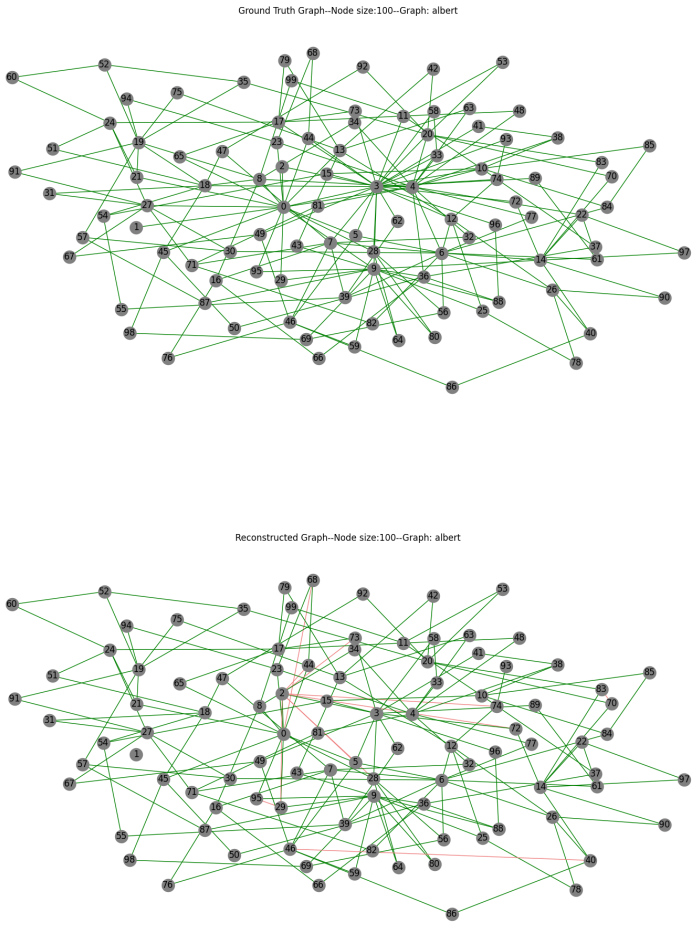


Fig. A.5.: Ground truth and reconstructed Barbasi-Albert network with 100 nodes and game of life dynamics. Red edges indicate false positives.

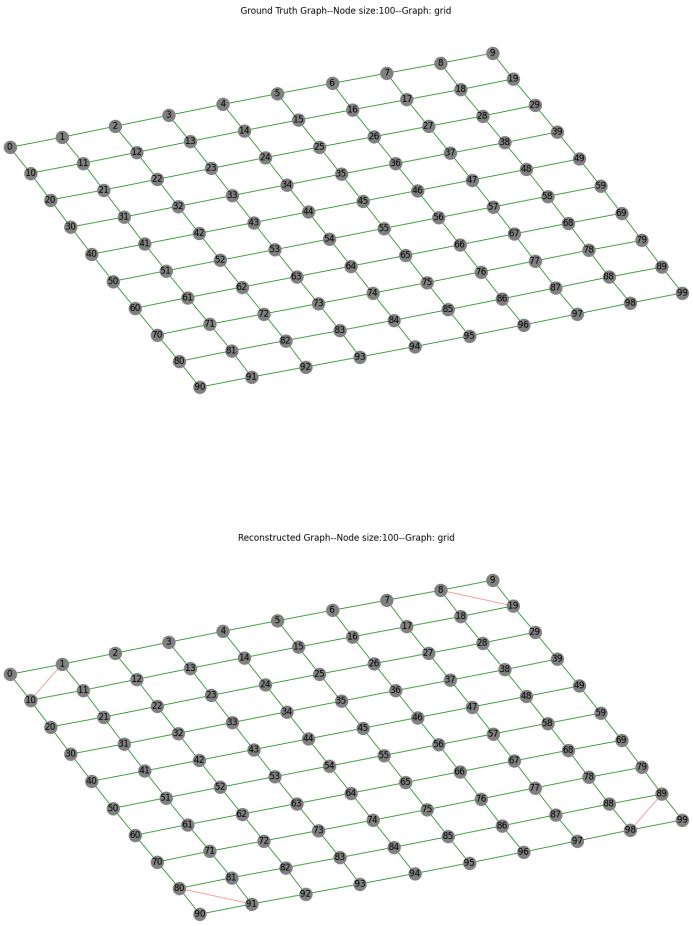


Fig. A.6.: Ground truth and reconstructed Grid network with 100 nodes and game of life dynamics. Red edges indicate false positives.

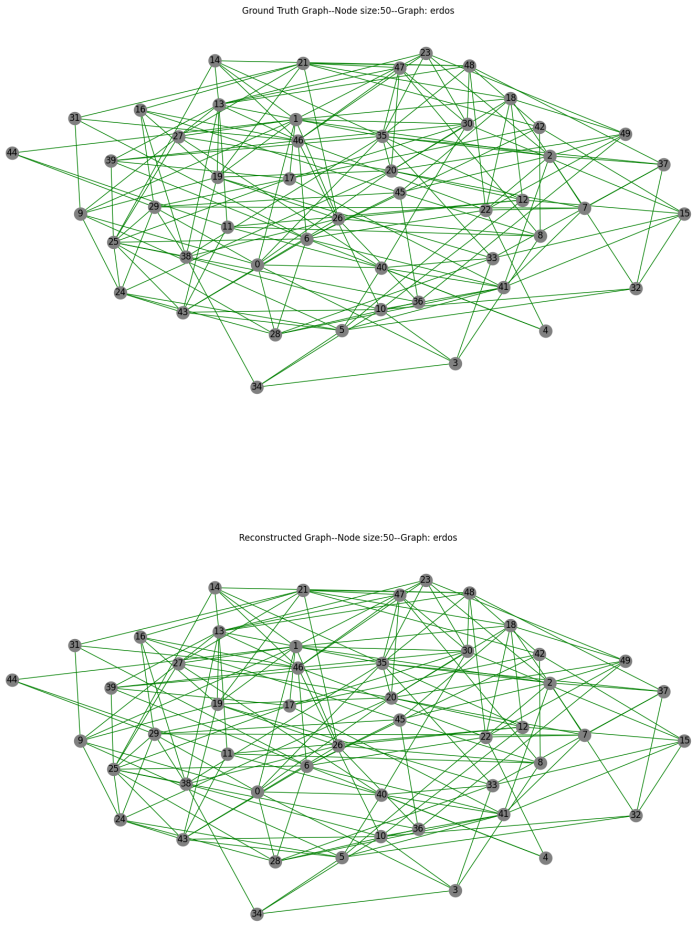


Fig. A.7.: Ground truth and reconstructed Erdos-Renyi network with 50 nodes and game of life dynamics. Red edges indicate false positives.

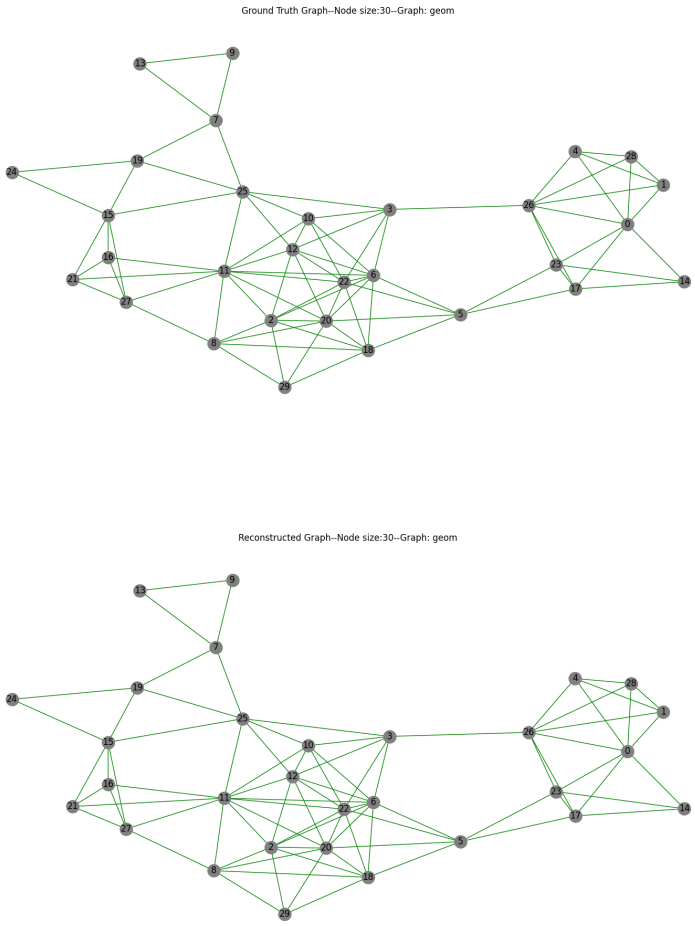


Fig. A.8.: Ground truth and reconstructed Random geometric network with 30 nodes and game of life dynamics. Red edges indicate false positives.