Iterative Structural Inference of Directed Graphs

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this paper, we propose a variational model, *Iterative Structural Inference of Directed Graphs* (iSIDG), to infer the existence of *directed interactions* from observational agents' features over a time period in a dynamical system. First, the iterative process in our model feeds the learned interactions back to encourage our model to eliminate indirect interactions and to emphasize directional representation during learning. Second, we show that extra regularization terms in the objective function for smoothness, connectiveness, and sparsity prompt our model to infer a more realistic structure and to further eliminate indirect interactions. We evaluate iSIDG on various datasets including biological networks, simulated fMRI data, and physics simulations to demonstrate that our model is able to precisely infer the existence of interactions, and is significantly superior to baseline models.

1 Introduction

A wide range of dynamical systems in the real world can be viewed as the composition of interacting agents, including physical systems [23, 13], multi-agent systems [5, 26] and biological systems [46, 33]. Knowing the interactions of dynamical systems can promote our understanding of the intrinsic mechanisms of these systems, and further facilitate accurate prediction and control of dynamical systems. However, in many cases, unlike the states of the agents, the interactions are usually not directly observable or measurable. Therefore, it is necessary to uncover the underlying interactions of dynamical systems based on the observational states of the agents. In dynamical systems, the states of an agent are affected by the interactions, and the states are usually recorded as a set of continuous variables, which make it difficult to uncover the interactions based on the similarity between the agents. It is also worth mentioning that most dynamical systems are characterized by *directed interactions*, such as citation networks [42], social ego networks [30], and gene regulatory networks [33]. We address the revealing of the existence of interactions in the dynamical systems as the problem of structural inference, and model the dynamical system as a directed graph, in which the nodes represent the agents of the system, and the directed edges denote the interactions between the agents. Thus, the structural inference problem is transformed into the inference of the asymmetric adjacency matrix of the graph based on observational nodes' features (dynamics).

While several models have been proposed to address this problem with unsupervised variational generative frameworks [21, 49, 29], most of them only deal with undirected interactions, and almost none of them explains why the variational method succeeds in inferring the structure. In this paper, based on the theory of *Information Bottleneck* (IB) [45, 1], which states the training of variational auto-encoder (VAE) follows the two-step procedure of deep learning: label-fitting and representation-compression, we explain the intrinsic mechanism of structural inference with VAEs, and propose a novel VAE based machine learning approach, namely **i**terative **S**tructural **I**nference of **D**irected **G**raphs (iSIDG), for iterative inference of the directed graph structure based on observational node features. Our iSIDG model utilizes a graph neural network (GNN) as encoder and infers adjacency matrix of the graph in its latent space.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

The key rationale of our iSIDG model is the iterative process for structural inference, which is designed based on the Variational Information Bottleneck [1]. The iterative process not only feeds the learned structure backward with direction information, but also creates tighter bounds and relaxation during the training process. The direction information contributes to a more diverse edge representations in the encoder and latent space. The iterative process connects several rounds of training a VAE into a loop, and joints the second phase: representation-compression with the label-fitting phase of the next round, which creates a tighter bound and a relaxation, to learn a more precise disentanglement between indirect connections and direct ones. Unlike previous works [21, 49, 29] that totally rely on the bottleneck structure of VAE to infer structure, iSIDG has extra terms to regularize the learned features in the latent space, such as sparsity and connectiveness, which carefully takes the properties of (asymmetric) adjacency matrix into account. Experimental results show that iSIDG outperforms or matches the state-of-the-art baselines on datasets with directed or undirected graphs. More importantly, iSIDG can distinguish indirect connections from direct ones more precisely.

2 Related Work

Kipf et al. [21] propose the framework of neural relational inference (NRI), which is the first to utilize a VAE to address the problem of structural inference based on observational node features. NRI conducts message-passing operations on a fixed fully connected graph structure in its encoder and infers graph structure in its latent space. Based on NRI, Webb et al. [49] propose factorized neural relational inference (fNRI), which extends NRI to multi-interaction systems. However, fNRI relies on ground-truth graph structure to classify results for each type of interaction. To deal with more complex systems, Li et al. [28] and Chen et al. [9] take various prior knowledge into account, such as sparsity and the distribution of node degrees, to increase the accuracy of structural inference. Unfortunately, it is quite difficult to acquire prior knowledge in many problem settings. Alet et al. [2] propose a modular meta-learning-based framework that jointly infers the structure with higher data efficiency. However, the framework has to be trained on various datasets beforehand to reach its best performance. From the aspect of Granger-causality, Löwe et al. [29] propose amortized causality discovery (ACD) method, which infers a latent posterior graph from temporal conditional dependence. Despite its high accuracy, the ACD method suffers from indirect interactions. Compared with this line of work, iSIDG introduces an iterative structural inference process to create a tighter bound by updating the fully connected graph structure in the encoder, and encourages edge representations to be diverse with the integrated direction information.

In addition to the aforementioned models to infer the structure of static graphs, several frameworks can also infer the structure of dynamic graphs. Ivanovic and Pavone [15] present a graph-structured model which combines recurrent sequence modeling and variational deep generative modeling to infer dynamic structure and predict future trajectories. Li et al. [25] propose a generic trajectory prediction forecasting network to provide multi-modal trajectory hypotheses and auxiliary structural inference. The framework in [12] utilizes sequential latent variable models to infer and predict separate relation graphs for every single time step. Moreover, the generic generative neural system proposed in [26] predicts trajectories based on dynamic graph representation and scene context information. The frameworks that can infer dynamic graph structures are mostly application-focused, and the structural inference is auxiliary to the task of trajectory prediction.

Besides the models based on variational interface, we would like to mention a few works which deal with the problem of structural inference as well, but based on different design methodologies. Casadiego et al. [6] propose the ARNI framework to infer the latent structure based on regression analysis and a careful choice of basis functions. Mutual information is also utilized to determine the existence of causal links and thus can infer the structure of dynamical systems [50, 38]. Some approaches fit a dynamics model and then produce a causal graph estimate of the model by using recurrent models [43, 18], or infer the structure by generating edges sequentially [17, 27] and others independently prune the generated edges from an over-complete graph [39]. Although these methods can provide insights about the correlations between adjacent nodes, such as mutual information and causal relation, they either require prior knowledge, or demand the comparison of features from all possible node-pairs, or are sensitive to indirect interactions in the graphs.

There exists another branch of research called Graph Structure Learning, which aims to jointly learn an optimized graph structure and corresponding graph representations for downstream tasks [57, 11, 16]. However, these methods adopt similarity-based methods to optimize the noisy graph structure, which

cannot be applied to dynamical systems. Furthermore, the main focus of these methods is the downstream task, not the reconstruction of the graph structure. Besides that, there are also a series of work to reconstruct the structure of directed acyclic graphs [56, 54, 36, 55]. However, these works have difficulties to work on graphs with cycles (and self-loops), which are commonly observed among dynamical systems. Last but not least, in the research field of biology, there are some work which can infer the structure of gene regulatory networks [24, 19, 7]. But these works either require prior assumption or cannot be extended to dynamical systems whose agents have multi-dimensional features. Unlike these works, iSIDG focuses on the structural inference of dynamical systems, and can deal with cycles in the system and high-dimensional node features.

3 Preliminaries

Notations and problem formulation. We view a dynamical system as a directed graph, where the agents of the system are represented as the nodes of the graph, and the directed interactions between the agents as the edges. We note the directed graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in which $\mathcal{V}$ represents the set of n nodes: $\{v_i, 1 \leq i \leq n\}$, and $\mathcal{E}$ represents the set of edges: $(v_i, v_j) \in \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Based on the set of edges, we derive to an asymmetric adjacency matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, in which $\mathbf{a}_{ij} \in \{0, 1\}$ indicates the presence ($\mathbf{a}_{ij} = 1$) of the edge between v_i and v_j or not ($\mathbf{a}_{ij} = 0$). The edges in our work are assumed to be static and do not evolve with time. The feature recordings of the nodes over a time period are represented as $\mathcal{V} = \{V^t, 0 \leq t \leq T\}$, where T is the total number of time steps , and V^t is the set of features of all the n nodes at time step t: $V^t = \{v_0^t, v_1^t, \ldots, v_n^t\}$. Given M sets of feature recordings $\{\mathcal{V}_k, 0 \leq k \leq M\}$, the structural inference problem we consider in this paper is to reconstruct the asymmetric adjacency matrix $\mathbf{A}$ of the graph in an unsupervised way.

We state the problem of structural inference as searching for a combinatorial distribution to describe the existence of the edges between any of the node pairs in the graph. As shown in Figure 1, we utilize a GNN based VAE [20] as the cornerstone, which can be trained to approximate complicated and high-dimensional probability distributions from the sampled node features [35]. We design an iterative process based on the IB theory, and feed the direction information, which is computed from the learned structure in the past training procedures, back to the input side, to encourage a more diverse learned representations, and to create a tighter bound as well as a relaxation of the training.

Graph neural networks. Graph neural networks (GNNs) are powerful deep learning algorithms to learn the representations of nodes and graphs. Most modern GNNs are designed based on the neighbourhood aggregation strategy [14], where the representation of a node is updated by aggregating the representations of its neighbours. In brief, we denote $h_i^{(k)}$ as the representation of node v_i at the k-th layer, and the mechanism of these GNNs can be described as:

$$h_i^{(k)} = \texttt{UPDATE}^{(k)}\big(h_i^{(k-1)}, \mathbf{m}_{\mathcal{N}(v_i)}^{(k-1)}\big)\text{ , where }\ \mathbf{m}_{\mathcal{N}(v_i)}^{(k-1)} = \texttt{AGGREGATE}^{(k)}(\{v_j \in \mathcal{N}(v_i)\})\,, \tag{1}$$

where $\mathcal{N}(v_i)$ denotes the set of chosen nodes related to v_i (e.g., its neighbours), `UPDATE` denotes the arbitrary function in the layer to update node representation, and `AGGREGATE` represents the aggregation function in the layer.

GCN [22] updates node representation based on one-hop neighbours and also its own representation in the previous layer, and calculates new representations with ReLU. GIN [51] aggregates the representations of adjacent nodes and node's own representation in the previous layer as a weighted summation, and utilizes multilayer perceptron to update node representations. GAT [47] introduces attention mechanism to implicitly specify different weights to adjacent nodes, and then updates representations upon the weighted summation. GNNs are efficient in learning rich representations for nodes and graphs. As a result, we utilize GNNs in the encoder to firstly learn representations of the nodes, and later on learn edge representations from the node representations in iSIDG.

Information bottleneck. The theory of *Information Bottleneck* (IB) provides an explanation and understanding of learning with deep neural networks [45, 44, 40]. In general, for the input data X and its label Y, the IB aims to learn the minimal sufficient representation $Z = \arg\min_Z I(Z; X) - \mathfrak{u} \cdot I(Z; Y)$, where $\mathfrak{u}$ is the Lagrangian multiplier to balance sufficiency and minimality. The training epochs of standard deep learning can be divided into two phases: label-fitting and representation-compression. These two phases can be characterized by the mutual information between the input features and the learned representation of a layer $I(X; Z)$, as well as between learned representation and the label variables $I(Z; Y)$. In the label-fitting phase, both $I(X; Z)$ and $I(Z; Y)$ increase, but

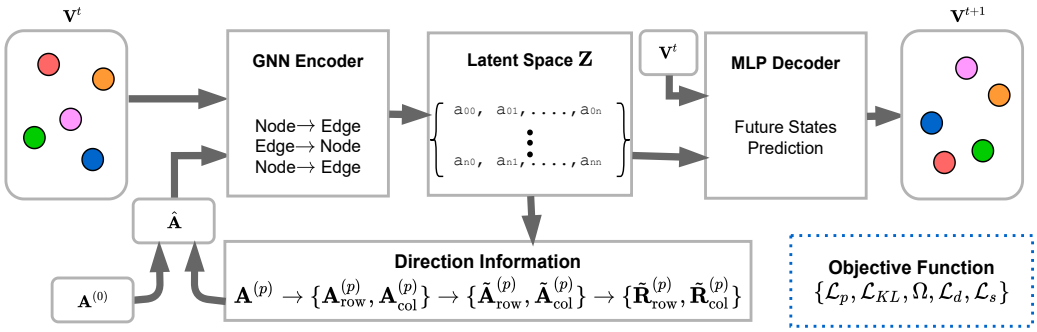

Figure 1: Overall architecture of the proposed iSIDG framework. iSIDG takes the fully-connected graph structure for the first run, but later on it takes the learned adjacency matrix with direction information as input. Regularization terms including smoothness, connectiveness and sparsity encourage learned features in latent space to be realistic and eliminate inferred indirect interactions.

in the representation-compression phase, $I(X;Z)$ decreases while $I(Z;Y)$ can either decrease or increase, depending on the amount of data [40].

Alemi et al. [1] present a variational approximation to the IB theory (VIB), and prove VAE objective is a special case of the variational approximation. Under a fully unsupervised setting, VAEs learn the minimal sufficient statistics from the input features, which are sufficient to derive the output features, and keep the most compressed and abstractive representations in their latent space. We utilize a VAE to infer the structure of the graph in its latent space, which is minimally sufficient to predict the future states of the nodes together with present node states.

It is suggested that IB iterations with stochastic relaxation methods may boost training [40]. It is also mentioned that a relaxation process is helpful to reduce local ambiguity and achieve global consistency [53], as already proved in previous work [8, 52]. Therefore, we design an iterative process with stochastic gradient descent (SGD) based optimizer to create a tighter bound and a relaxation for the training, and to encourage iSIDG to further disentangle dynamics from the adjacency matrix. In order to verify the effectiveness of this iterative process, we conduct a case study in Section D.3.

4 Model Design

4.1 Structural Inference by Information Bottleneck

Together with node features, graph structure characterizes the basic information of a graph, which represents a dynamical system. The goal of structural inference is to accurately reconstruct the adjacency matrix $\mathbf{A}$ of the directed graph based on the recordings of node features: $\mathbf{A} \leftarrow \{V^t, 0 \leq t \leq T\}$. Interestingly, for dynamical systems, the future features of the nodes are deterministic by their previous features and the connectivity: $V^{t+1} \leftarrow \{V^0, V^1, \ldots, V^t; \mathbf{A}\}$. We assume the Markovian assumption works in our case, and thus previous statement can be simplified to: $V^{t+1} \leftarrow \{V^t; \mathbf{A}\}$. Thus, we derive to an assumption: the information about the adjacency matrix is implicitly included in the node features over a time period, which also conforms to the IB theory.

Based on VIB, we derive the *Information Bottleneck for Structural Inference*, which aims at inferring the adjacency matrix $\mathbf{A}$ in the latent space $\mathbf{Z}$:

$$\mathbf{Z} = \arg\min_{\mathbf{Z}} I(\mathbf{Z}; V^t, \mathbf{A}) - \mathfrak{u} \cdot I(\mathbf{Z}; V^{t+1}), \tag{2}$$

Interestingly, $\mathbf{A}$ is also required in the first term in Equation 2, which is unrealistic in our problem setting. Thus, we assume a fully connected graphs structure $\mathbf{A}^{(0)}$ as input, which is not less informative than the actual sparse adjacency matrix:

$$\mathbf{Z} = \arg\min_{\mathbf{Z}} I(\mathbf{Z}; V^t, \mathbf{A}^{(0)}) - \mathfrak{u} \cdot I(\mathbf{Z}; V^{t+1}). \tag{3}$$

Following the statement in VIB [1], we adopt a VAE [20] to deal with the problem described in Equation 3, where an encoder $q_\phi(\mathbf{Z}|V^t, \mathbf{A}^{(0)})$ approximates $I(\mathbf{Z}; V^t, \mathbf{A}^{(0)})$, and a decoder

$p_\theta(\hat{V}^{t+1}|\mathbf{Z}, V^t)$ approximates $I(\mathbf{Z}; V^{t+1})$. ϕ and θ represent the parameters of encoder and decoder, respectively. We set the latent space of the VAE with the size of $n \times n$. The fixed dimension of the latent space restricts the size of the search space for the adjacency matrix. We utilize a weighted ELBO with multiple integrated regularization terms to optimize model parameters $\{\phi, \theta\}$.

4.2 Encoder

The encoder in our model is tasked with inferring the existence of directed interactions $\mathbf{z}_{ij}$ between all node-pairs by extracting the most compressed representation given the node features $\mathcal{V}$.

The encoder $q_\phi(\mathbf{Z}|V^t, \mathbf{A}^{(0)})$ returns a set $\{\mathbf{z}_{ij}|v_i, v_j \in \mathcal{V}\}$, which features the distribution to describe how probable the edge from node j to node i exists: $p(\mathbf{z}_{ij}) \in [0, 1]$. The prior $p(\mathbf{Z}) = \prod_{i,j\in\mathcal{V}} p(\mathbf{z}_{ij})$ is a factorized uniform distribution. We utilize GNNs to extract features in the encoder. But the GNNs learn new representation based on both node features and the topology of the graph $f_{\text{GNN}}(\mathcal{V}, \mathbf{A})$, in which the adjacency matrix $\mathbf{A}$ is our learning target. To encounter this problem, we impose an iterative learning process on the adjacency matrix, and we discuss the details in Section 4.4. For simplicity, in this section we use $\hat{\mathbf{A}}$ to represent the adjacency matrix. In general, the encoder:

$$q_\phi = \text{softmax}\left(\mathbf{f}_{\text{enc}}(\mathcal{V}, \hat{\mathbf{A}})\right), \tag{4}$$

where f_{enc} represents the encoder. The basic setup of our encoder is similar to that in [21], which applies two rounds of node-to-edge and one round of edge-to-node message passing operations to learn the edge embeddings $\mathbf{h}_{ij}$:

$$\text{Node Embedding:} \quad \mathbf{e}_j^1 = f_{\text{embed}}^{(1)}(v_j), \tag{5}$$

$$\text{Node-to-edge:} \quad \mathbf{e}_{ij} = f_e^1([\mathbf{e}_i^1, \mathbf{e}_j^1]), \text{ if } a_{ij} = 1 \text{ in } \hat{\mathbf{A}}, \tag{6}$$

$$\text{Edge-to-node:} \quad \mathbf{e}_j^2 = f_v\left(\sum \mathbf{e}_{ij}\right), \tag{7}$$

$$\text{Node-to-edge:} \quad \mathbf{h}_{ij} = f_e^2([\mathbf{e}_i^2, \mathbf{e}_j^2]), \text{ if } a_{ij} = 1 \text{ in } \hat{\mathbf{A}}, \tag{8}$$

where $f_{\text{embed}}^{(1)}$ is the embedding network for input features, f_e^1 and f_e^2 are node-to-edge message-passing networks, and f_v is the edge-to-node operation. We observe that Equations 5 - 7 is a round of message passing for node features in most GNNs, as a result, we substitute Equations 5 - 8 as:

$$\mathbf{h}_{ij} = f_e^2\left(f_{\text{GNN}}(\mathcal{V}, \hat{\mathbf{A}}), \hat{\mathbf{A}}\right), \tag{9}$$

so that we can integrate state-of-the-art GNNs such as GCN, GAT and GIN into our encoder.

As a consequence, our encoder is designed to learn informative features from input node features and compress them to fit the restricted search space, which has the dimension of a fully connected graph. By extracting information from the high-dimensional but low-level node features to the restricted and low-dimensional latent space, our encoder disentangles the dynamics and approximates the minimally sufficient statistics, the adjacency matrix. After that, we model the existence of edges as $\mathbf{Z} \in \mathbb{R}^{n\times n}$, where its component $\mathbf{z}_{ij}$ comes from $q_\phi(\mathbf{z}_{ij}|V^t, \hat{\mathbf{A}}) = \text{softmax}(\mathbf{h}_{ij})$.

4.3 Decoder

The task of the decoder is to predict the future states V^{t+1} based on the samplings from latent space $\mathbf{Z}$ and the present node features V^t. From the IB theory, if the encoder derives minimally sufficient statistics as the adjacency matrix, the decoder can precisely predict the future node features. However, there remains the problem of how to design the decoder, so that it can further encourage the encoder to disentangle dynamics and to infer the graph structure only. In order to deal with this problem, we design the decoder to imitate a single message-passing operation. We feed our decoder with the present node features besides the samplings from the latent space to predict the future node features. As a consequence, the node features are disentangled from the compressed information in the latent space during training. In short, the decoder deals with the following problem: $p_\theta(V^{t+1}|V^t, \mathbf{Z})$. We use a two-stage decoder, which consists of a message-passing layer and an embedding layer:

$$\hat{\mathbf{h}}_j^{t+1} = v_j^t + f_{\text{embed}}^{(3)}\left(\sum_{\mathbf{z}_{ij}>0} \mathbf{z}_{ij} f_{\text{embed}}^{(2)}([v_i^t, v_j^t])\right), \tag{10}$$

where $f_{\text{embed}}^{(2)}$ and $f_{\text{embed}}^{(3)}$ are multilayer perceptrons (MLPs) to embed the features.

4.4 Iterative Learning with Direction Information

The motivation of integrating iterative learning process into the structural inference consists of three aspects: (1) to create a tighter bound for Equation 3; (2) to create a relaxation for the training process; (3) to emphasize direction information. In order to formulate the ideas more precisely, we would like to firstly describe the iterative process with direction information in our model.

From the very beginning, we have no prior knowledge about the structure of the graph. In order to utilize GNNs to extract and compress the features into a restricted search space for adjacency matrix, we feed the encoder with a fully connected graph $\mathbf{A}^{(0)} = \mathbf{1} \in \mathbb{R}^{n \times n}$. After η rounds of training, we obtain an asymmetric adjacency matrix $\mathbf{A}^{(p)}$ from the latent space, which is calculated from the averaged result of the entire training samples:

$$\mathbf{A}^{(p)} = \frac{1}{m} \sum_{k \in M} \mathbf{Z}_{\text{k}} \,, \tag{11}$$

where m is the total number of training samples, and M is the set of training data. The full-batch average operation not only preserves the most information from the training set, but also makes the $\mathbf{A}^{(p)}$ to be invulnerable to outliers. We then normalize $\mathbf{A}^{(t)}$ in both row-wise and column-wise:

$$\mathbf{A}_{\text{row}}^{(p)} = \mathbf{D}_{\text{in}}^{-1} \mathbf{A}^{(p)} \text{ , and } \mathbf{A}_{\text{col}}^{(p)} = \mathbf{A}^{(p)} \mathbf{D}_{\text{out}}^{-1} \,, \tag{12}$$

where $\mathbf{D}_{\text{in}}^{-1}$ and $\mathbf{D}_{\text{out}}^{-1}$ are in-degree and out-degree matrices which are calculated based on $\mathbf{A}^{(p)}$, respectively. We name Equations 12 as *direction information*. After that, we combine the direction information matrices with the normalized initial fully-connected adjacency matrix, respectively:

$$\tilde{\mathbf{A}}_{\text{row}}^{(p)} = \lambda \mathbf{L} + (1-\lambda) \mathbf{A}_{\text{row}}^{(p)} \text{ , and } \tilde{\mathbf{A}}_{\text{col}}^{(p)} = \lambda \mathbf{L} + (1-\lambda) \mathbf{A}_{\text{col}}^{(p)} \,, \tag{13}$$

where $\mathbf{L}$ is the normalized adjacency matrix of the initial fully-connected graph $\mathbf{A}^{(0)}$, and λ is the combination coefficient. Since $\mathbf{A}^{(0)}$ is symmetric, we can use either column-wise or row-wise normalization. We map the results in Equation 13 to binary sets with the characteristic function:

$$\tilde{\mathbf{R}}_{\kappa}^{(p)} = \mathbb{1}_{\xi}(\tilde{\mathbf{A}}_{\kappa}^{(p)}) = \begin{cases} 1 & \text{if } \mathbf{a}_{ij} \geq \xi \text{ for } \mathbf{a}_{ij} \text{ in } \tilde{\mathbf{A}}_{\kappa}^{(p)} \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

where $\kappa \in \{\text{row}, \text{col}\}$, and ξ is the threshold value. Finally, we feed $\tilde{\mathbf{R}}_{\kappa}^{(p)}$ back to the input side of encoder mentioned in Section 4.2 by substituting the $\hat{\mathbf{A}}$ in Equations 6 and 8. More formally, the two feature vectors which are concatenated in Equations 6 and 8 represent the sender and receiver of the edge, respectively. We rewrite the equations for the iterative process according to $\tilde{\mathbf{R}}_{\kappa}^{(p)}$:

$$\mathbf{k}_{ij} = \begin{cases} f_e^{\text{n}}([\mathbf{k}_i, \mathbf{k}_j]) & \text{if } \mathbf{r}_{ij} = 1 \text{ in } \tilde{\mathbf{R}}_{\text{col}}^{(p)} \text{ and } \tilde{\mathbf{R}}_{\text{row}}^{(p)} \\ f_e^{\text{n}}([\mathbf{k}_i,\ 0\,]) & \text{if } \mathbf{r}_{ij} = 1 \text{ only in } \tilde{\mathbf{R}}_{\text{col}}^{(p)} \\ f_e^{\text{n}}([\,0,\ \mathbf{k}_j]) & \text{if } \mathbf{r}_{ij} = 1 \text{ only in } \tilde{\mathbf{R}}_{\text{row}}^{(p)} \,, \end{cases} \tag{15}$$

where $\mathbf{k}_{ij} \in \{\mathbf{e}_{ij}, \mathbf{h}_{ij}\}$, $f_e^{\text{n}} \in \{f_e^1, f_e^2\}$, and $\mathbf{k}_n \in \{\mathbf{e}_n^1, \mathbf{e}_n^2\}$ in Equations 6 and 8, respectively. After the iteration begins, we introduce another inner iteration with the current updated adjacency matrices $\tilde{\mathbf{R}}_{\kappa}^{(p)}$. The stop condition for inner iteration is:

$$\|\mathbf{A}^{[i]} - \mathbf{A}^{[i-1]}\|_F^2 < \delta \|\mathbf{A}^{(0)}\|_F^2 \,, \tag{16}$$

where superscript $[i]$ represents the rounds within inner iteration. Once this condition is satisfied, the adjacency matrix at the input side of the encoder will be updated according to Equations 11 - 15.

Tighter bound. During training, the initial fully connected adjacency matrix $\mathbf{A}^{(0)}$ is updated to sparser matrices $\{\hat{\mathbf{A}}_{\kappa}^{(p)}, \kappa \in \{\text{row}, \text{col}\}\}$. (We omit the description for κ in this paragraph for clearer statement). The model is able to roughly detect which edges are not beneficial to the future state prediction, and eliminates these edges in the new representation. As a consequence, any of the matrices in $\{\hat{\mathbf{A}}_{\kappa}^{(p)}\}$ is sparser than $\mathbf{A}^{(0)}$. Thus the first term in Equation 3 is re-framed as

$I(\mathbf{Z}; V^t, \{\hat{\mathbf{A}}_\kappa^{(p)}\})$, which obviously supports: $I(\mathbf{Z}; V^t, \{\hat{\mathbf{A}}_\kappa^{(p)}\}) \leq I(\mathbf{Z}; V^t, \mathbf{A}^{(0)})$, and results in a tighter bound for $\mathbf{Z}$ in Equation 3.

Relaxation. Besides that, our iterative training process imposes relaxation to connect the two phases, i.e., label-fitting and representation-compression, of training a deep learning network as a cycle. In the first phase, our model for structural inference tries to fit the target for prediction, when the training errors become small, it turns to the second phase and compresses the representation of any layer to be minimally sufficient to fit the target. In the second phase, the compression process affects the adjacency matrix in latent space as well, and due to the difficulty of training in an unsupervised manner, it is very apt to reach a local optimal. The iterative process, which feeds the learned adjacency matrix back to the encoder, stops the phase of representation-compression, and starts the next round of label-fitting and representation-compression. At this round, the combination of direction information and the initial adjacency matrix together creates a different adjacency matrix for the encoder compared with the previous round, which will relax the learning of adjacency matrix back from the local optimal. We verify the effectiveness of the iterative process through mutual information in Section D.3.

Direction information. One of the main differences between undirected and directed graphs is that the former can be described with a unified degree matrix, but the latter requires two matrices for in-degree and out-degree. As a result, we introduce direction information by integrating the in- and out-degrees of every node with the new adjacency matrices, which contributes to a bifurcated training process and encourages the learned embeddings for edges to be more diverse. The integration of direction information is beneficial to the structural inference for directed graphs, which is validated experimentally in Section 5.1.

4.5 Training with a Hybrid Loss

Although the iterative process is an effective way to learn the adjacency matrix of directed graphs, adequate loss function can promote the training process as well. Firstly, the following is the loss function for prediction:

$$\mathcal{L}_p = \mathbb{E}_{q_\phi(\mathbf{Z}|V,\mathbf{A})}[\log p_\theta(V \mid \mathbf{Z})] \,. \tag{17}$$

Then the Kullback–Leibler (KL) divergence acts as a regularization for the latent space:

$$\mathcal{L}_{KL} = -\mathrm{KL}[q_\phi(\mathbf{Z} \mid V) \| p_\theta(\mathbf{Z})] \,. \tag{18}$$

We omit the superscripts of vectors V and v in Equations 17 - 21, the actual correspondence refers to Sections 4.2 and 4.3.

In addition to the aforementioned terms which are often utilized as the objective function for VAEs, we introduce extra regularization terms. The regularization term of signal *smoothness* takes both inferred structure and node features into consideration to eliminate any connection which does not fulfill the smoothness assumption of adjacent node features, such as indirect interactions. We consider input features V as graph signals, and the assumption for graph signals also works here: values change smoothly across adjacent nodes [10]. Therefore, we introduce Dirichlet energy [3] to measure the smoothness between the signals:

$$\Omega(\mathbf{Z}, V) = \frac{1}{n^2} \sum_{i,j} \mathbf{Z}_{ij} \|v_i - v_j\|^2 \,. \tag{19}$$

It is worth mentioning that in most cases the adjacency matrix is asymmetric, we cannot use Laplacian matrices to simplify the formulation.

Moreover, we introduce more regularization terms [3] to feature the learned features in latent space with the properties of realistic adjacency matrices, such as *connectiveness* $\mathcal{L}_d$ and *sparsity* $\mathcal{L}_s$. Training the network with these regularization terms encourages our model to further disentangle node representations from the latent space and distinguish indirect interactions from direct ones:

$$\mathcal{L}_d = -\frac{1}{n} \mathbf{1}^\mathsf{T} \log(\mathbf{Z}\mathbf{1}) \,, \text{ and } \; \mathcal{L}_s = \frac{1}{n^2} \|\mathbf{Z}\|_F^2 \,, \tag{20}$$

where the logarithmic barrier in $\mathcal{L}_d$ forces the model to learn a connected adjacency matrix, but lacks the regularization of sparsity. In consequence, we introduce $\mathcal{L}_s$ to regularize its sparsity.

Table 1: AUROC values (%) of iSIDG and baselines on synthetic networks

METHOD	LI	LL	CY	BF	TF	BF-CV
iSIDG	**86.2**± 4.2	**88.1**± 4.9	**79.5**± 3.8	**68.3**± 7.4	**60.2**± 9.5	**70.7**± 6.0
NRI	70.5± 5.0	75.0± 4.7	64.5± 6.2	59.0± 3.6	55.1± 7.6	59.2± 7.4
fNRI	73.0± 5.3	77.6± 4.9	67.1± 7.0	63.8± 6.9	59.0± 7.1	64.8± 8.2
MPIR	46.8± 5.6	49.0± 6.4	31.7± 10.9	47.5± 8.0	40.9± 6.1	49.2± 4.0
ACD	65.0± 5.2	68.4± 4.3	62.9± 3.7	59.8± 6.7	57.2± 7.0	55.8± 6.7

To conclude, we train our iSIDG with the hybrid loss $\mathcal{L}$:

$$\mathcal{L} = \mathcal{L}_p + \mu \cdot \mathcal{L}_{KL} + \alpha \cdot \Omega(\mathbf{Z}, V) + \beta \cdot \mathcal{L}_d + \gamma \cdot \mathcal{L}_s \,, \tag{21}$$

where $\{\mu, \alpha, \beta, \gamma\}$ are hyperparameters. The first two terms in Equation 21 formulate a weighted ELBO, which is the training objective for VAEs. The rest of the terms are regularization for the learned adjacency matrix in latent space. Together with the training cycles of fitting and compression, which are connected with relaxations, the hybrid loss function further encourages our iSIDG model to reconstruct the adjacency matrix precisely. We give a detailed discussion on the design methodology of iSIDG on how to effectively eliminate indirect connections in Section A.

5 Experiments

We test our iSIDG model on three different datasets, and present its ablation study on one of the datasets. Implementation details and further experimental results can be found in Sections B - D.

5.1 Structural Inference on Different Systems

Datasets. We test our model on the six directed synthetic biological networks [33], namely Linear (LI), Linear Long (LL), Cycle (CY), Bifurcating (BF), Trifurcating (TF), and Bifurcating Converging (BF-CV) networks. These networks are essential components that lead to a variety of different trajectories that are commonly observed in differentiating and developing cells [37]. We use BoolODE [33] to simulate the process of developing cells with these synthetic networks, and obtain six groups of trajectories. The features at every node are one-dimensional, i.e., the level of mRNA expression.

We also test our model on NetSim datasets [41] of simulated fMRI data. The NetSim datasets consist of simulated blood-oxygen-level-dependent imaging data across different regions within the human brain, which leads to asymmetric relation networks. The node features are one-dimensional.

The last datasets to be mentioned here are three physical simulations mentioned in [21], namely springs, charged particles and phase-coupled oscillators (Kuramoto model). We keep the symmetric interactions in the setting of these networks. However, different from the setting mentioned in that work, we sample the trajectories with fixed interactions between the agents in the system, but with different initial conditions, in order to simulate the real application scenario on the observational data from a specific unknown system. The features at every node are 4-dimensional.

Baselines and metrics. We compare iSIDG with the state-of-the-art models for structural inference:

- NRI [21]: a VAE-based model for unsupervised relational inference.
- fNRI [49]: an NRI-based model with additional latent space for every factorized interaction type.
- MPIR [50] a model based on minimum predictive information regularization.
- ACD [29]: a variational model that leverages shared dynamics to infer causal relations.

We describe the implementation details of the baseline methods in Section B.2. We demonstrate our evaluation results with the following metrics: the area under the receiver operating characteristic (AUROC), the area under the precision-recall curve (AUPRC) and the Jaccard similarity index (JI). We also provide results of two DAG-based structure learning methods on our datasets in Section F.

Results. The experimental results of our iSIDG model and baseline methods are summarized in Tables 1 and 2 (also Tables 5 - 8 in Appendix), which consist of mean values and 95% intervals of AUROC, AUPRC and JI from 15 experiments on each, respectively. Due to the alternative simulation pipelines between ours and other literature, we obtain different results of ACD and MPIR on physical simulations from other literature. iSIDG outperforms or matches the baseline methods. In the datasets of synthetic networks and three NetSim datasets, which consist of directed graphs and low

Table 2: AUROC values (%) of iSIDG and baselines on physical simulations and NetSim datasets.

METHOD	Springs	Particles	Kuramoto	NetSim1	NetSim2	NetSim3
iSIDG	**90.5**± **3.9**	**78.1**± **4.9**	79.8± 5.5	**75.5** ± **5.2**	**72.4** ± **5.2**	**71.0** ± **4.2**
NRI	89.2± 2.6	75.1± 2.8	78.1± 4.3	72.1 ± 3.6	72.0 ± 4.1	68.2 ± 3.3
fNRI	90.4± 1.9	77.0± 3.0	79.5± 5.2	72.5 ± 4.6	71.2 ± 5.0	69.6 ± 3.9
MPIR	59.5± 3.2	55.7± 4.9	50.2± 6.3	47.2 ± 3.1	46.0 ± 3.8	44.3 ± 1.7
ACD	90.1± 4.1	77.9± 3.6	**80.5**± **5.6**	66.7 ± 3.6	64.0 ± 3.1	62.9 ± 2.5

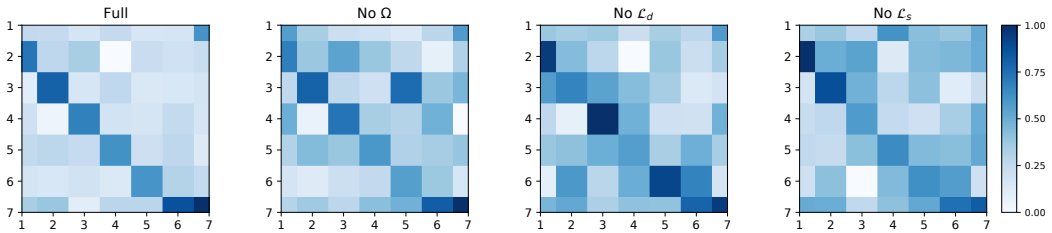

Figure 2: Reconstruction results of iSIDG with different objective functions on LI dataset.

dimensional input features, iSIDG outperforms baseline models by at most 13.2 percent in terms of AUROC, showing that it can infer structure more precisely than the baseline methods. These results highlight the validity of integrating the iterative training process and the direction information on structural inference problem for directed graphs. It is also remarkable that iSIDG matches or is slightly inferior to the baseline methods in the datasets of physical simulations, especially fNRI and ACD. The reason may be that high dimensional input features (4-dimensional features for physical simulation datasets) contain richer dynamics for fNRI and other methods to infer from. It is worth mentioning that fNRI method utilizes ground truth to distinguish every latent space and match them to the edge type with the highest accuracy, which results in higher AUROC, AUPRC and JI scores. Moreover, since ACD relies on the information shared by input features, the richer information contained in high dimensional features promotes a more promising result.

5.2 Ablation Study

We conduct ablation studies on the effectiveness of regularization terms in objective function: signal smoothness Ω, connectiveness $\mathcal{L}_d$ and sparsity $\mathcal{L}_s$, and the reconstruction results are shown in terms of heat maps in Figure 2. The heat maps visualize the average results of 15 experiments of iSIDG with different objective functions by deleting one regularization term, respectively. As shown in the second plot in Figure 2, without considering signal smoothness, the indirect interactions emerge again in the results of structural inference with the correctly inferred edges. The third plot in Figure 2 shows that without considering the connectiveness, the model correctly infers only a few direct interactions but as isolated “islands", and connects these islands with numerous indirect interactions. The last plot in Figure 2 demonstrates the results that without the terms to regularize the sparsity, many direct interactions are substituted by the indirect ones. Therefore, the extra regularization terms in the objective function of iSIDG effectively encourage the model to infer realistic graph structures and to eliminate the influence of indirect interactions. More ablation studies can be found in Section D.4.

6 Future Work

We so far only consider the structural inference for static and (relatively) small graphs, though we tested the scalability of iSIDG on graphs of 100 nodes (see Section E for the results). Future research includes introducing dynamic interactions and more scalable model design which remains as a critical challenge in structure learning for dynamical systems. Furthermore, we also want to integrate prior knowledge such as partial structural information to further promote the performance of iSIDG.

References

[1] A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy. Deep variational information bottleneck. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[2] F. Alet, E. Weng, T. Lozano-Pérez, and L. P. Kaelbling. Neural relational inference with fast modular meta-learning. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 11804–11815, 2019.

[3] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of the 15th International Conference on Neural Information Processing Systems (NIPS)*, pages 585–591, 2001.

[4] F. H. Biase, X. Cao, and S. Zhong. Cell fate inclination within 2-cell and 4-cell mouse embryos revealed by single-cell rna sequencing. *Genome Research*, 24(11):1787–1796, 2014.

[5] G. Brasó and L. Leal-Taixé. Learning a neural solver for multiple object tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6247–6257, 2020.

[6] J. Casadiego, M. Nitzan, S. Hallerberg, and M. Timme. Model-free inference of direct network interactions from nonlinear collective dynamics. *Nature Communications*, 8, 12 2017.

[7] T. E. Chan, M. P. Stumpf, and A. C. Babtie. Gene regulatory network inference from single-cell data using multivariate information measures. *Cell Systems*, 5(3):251–267.e3, 2017.

[8] P. Chaudhari, A. Oberman, S. Osher, S. Soatto, and G. Carlier. Deep relaxation: partial differential equations for optimizing deep neural networks. *Research in the Mathematical Sciences*, 5(3):1–30, 2018.

[9] S. Chen, J. Wang, and G. Li. Neural relational inference with efficient message passing mechanisms. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, pages 7055–7063, 2021.

[10] Y. Chen, L. Wu, and M. Zaki. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. In *Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[11] B. Fatemi, L. E. Asri, and S. M. Kazemi. SLAPS: self-supervision improves structure learning for graph neural networks. In *Proceedings of the 35th Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 22667–22681, 2021.

[12] C. Graber and A. G. Schwing. Dynamic neural relational inference for forecasting trajectories. In *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 4383–4392, 2020.

[13] S. Ha and H. Jeong. Unraveling hidden interactions in complex systems with deep learning. *Scientific Reports*, 11(1):1–13, 2021.

[14] W. L. Hamilton. *Graph Representation Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2020.

[15] B. Ivanovic and M. Pavone. The Trajectron: Probabilistic multi-agent trajectory modeling with dynamic spatiotemporal graphs. In *Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2375–2384, 2019.

[16] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang. Graph structure learning for robust graph neural networks. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 66–74. ACM, 2020.

[17] D. D. Johnson. Learning graphical state transitions. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[18] S. Khanna and V. Y. F. Tan. Economy statistical recurrent units for inferring nonlinear granger causality. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

[19] S. Kim. Ppcor: An r package for a fast calculation to semi-partial correlation coefficients. *Communications for Statistical Applications and Methods*, 22:665–674, 11 2015.

[20] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[21] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel. Neural relational inference for interacting systems. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 2688–2697. PMLR, 2018.

[22] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

[23] J. Kwapień and S. Drożdż. Physical approach to complex systems. *Physics Reports*, 515(3): 115–226, 2012.

[24] A. Lamere and J. Li. Leap: Constructing gene co-expression networks for single-cell rna-sequencing data using pseudotime ordering. *Bioinformatics*, 33(5):764–766, 2016.

[25] J. Li, F. Yang, M. Tomizuka, and C. Choi. Evolvegraph: Multi-agent trajectory prediction with dynamic relational reasoning. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[26] J. Li, H. Ma, Z. Zhang, J. Li, and M. Tomizuka. Spatio-temporal graph dual-attention network for multi-agent prediction and tracking. *arXiv preprint arXiv:2102.09117*, 2021.

[27] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. W. Battaglia. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

[28] Y. Li, C. Meng, C. Shahabi, and Y. Liu. Structure-informed graph auto-encoder for relational inference and simulation. In *Proceedings of ICML Workshop on Learning and Reasoning with Graph-Structured Data*, page 2, 2019.

[29] S. Löwe, D. Madras, R. Zemel, and M. Welling. Amortized causal discovery: Learning to infer causal graphs from time-series data. *arXiv preprint arXiv:2006.10833*, 2020.

[30] J. J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 548–556, 2012.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 8024–8035, 2019.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[33] A. Pratapa, A. P. Jalihal, J. N. Law, A. Bharadwaj, and T. Murali. Benchmarking algorithms for gene regulatory network inference from single-cell transcriptomic data. *Nature Methods*, 17(2): 147–154, 2020.

[34] B. C. Ross. Mutual information between discrete and continuous data sets. *PLOS ONE*, 9:1–5, 2014.

[35] L. Ruthotto and E. Haber. An introduction to deep generative modeling. *GAMM-Mitteilungen*, page e202100008, 2021.

[36] B. Saeed, S. Panigrahi, and C. Uhler. Causal structure discovery from distributions arising from mixtures of dags. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 8336–8345. PMLR, 2020.

[37] W. Saelens, R. Cannoodt, H. Todorov, and Y. Saeys. A comparison of single-cell trajectory inference methods. *Nature Biotechnology*, 37(5):547–554, 2019.

[38] T. Schreiber. Measuring information transfer. *Physical Review Letters*, 85(2):461, 2000.

[39] R. Selvan, T. N. Kipf, M. Welling, J. H. Pedersen, J. Petersen, and M. de Bruijne. Graph refinement based tree extraction using mean-field networks and graph neural networks. *arXiv preprint arXiv:1811.08674*, 2018.

[40] R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[41] S. M. Smith, K. L. Miller, G. Salimi-Khorshidi, M. Webster, C. F. Beckmann, T. E. Nichols, J. D. Ramsey, and M. W. Woolrich. Network modelling methods for FMRI. *Neuroimage*, 54(2): 875–891, 2011.

[42] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 990–998. ACM, 2008.

[43] A. Tank, I. Covert, N. Foti, A. Shojaie, and E. Fox. Neural Granger Causality. *arXiv preprint arXiv:1802.05842*, 2018.

[44] N. Tishby and N. Zaslavsky. Deep learning and the information bottleneck principle. In *Proceedings of 2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2015.

[45] N. Tishby, F. Pereira, and W. Biale. The information bottleneck method. In *Proceedings of the 37th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 368–377. IEEE, 1999.

[46] M. Tsubaki, K. Tomii, and J. Sese. Compound–protein interaction prediction with end-to-end learning of neural networks for graphs and sequences. *Bioinformatics*, 35(2):309–318, 2019.

[47] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, 2018.

[48] M. J. Vowels, N. C. Camgoz, and R. Bowden. D'ya like dags? a survey on structure learning and causal discovery. *arXiv preprint arXiv:2103.02582*, 2021.

[49] E. Webb, B. Day, H. Andres-Terre, and P. Lió. Factorised neural relational inference for multi-interaction systems. *arXiv preprints arXiv:1905.08721*, 2019.

[50] T. Wu, T. Breuel, M. Skuhersky, and J. Kautz. Discovering nonlinear relations with minimum predictive information regularization. *arXiv preprint arXiv:2001.01885*, 2020.

[51] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.

[52] P. Yin, S. Zhang, J. Lyu, S. Osher, Y. Qi, and J. Xin. Binaryrelax: A relaxation approach for training deep neural networks with quantized weights. *SIAM Journal on Imaging Sciences*, 11 (4):2205–2223, 2018.

[53] S.-S. Yu and W.-H. Tsai. Relaxation by the Hopfield neural network. *Pattern Recognition*, 25 (2):197–209, 1992.

[54] Y. Yu, J. Chen, T. Gao, and M. Yu. DAG-GNN: DAG structure learning with graph neural networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 7154–7163. PMLR, 2019.

[55] Y. Yu, T. Gao, N. Yin, and Q. Ji. DAGs with No Curl: An efficient DAG structure learning approach. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 12156–12166. PMLR, 2021.

[56] X. Zheng, B. Aragam, P. Ravikumar, and E. P. Xing. DAGs with NO TEARS: continuous optimization for structure learning. In *Proceedings of the 32nd Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[57] Y. Zhu, W. Xu, J. Zhang, Q. Liu, S. Wu, and L. Wang. Deep graph structure learning for robust representations: A survey. *arXiv preprint arXiv:2103.03036*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See abstract and last paragraph of Section 1.
   (b) Did you describe the limitations of your work? [Yes] Due to page limitation, we describe the limitation of our work with a few words in Section 6, and in Section E.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section G.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We have read the ethics review guidelines and ensured that our paper conforms to them.

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Sections 3 and 4.1.
   (b) Did you include complete proofs of all theoretical results? [Yes] The proofs are quite intuitive, see Section 4.1.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We attach the link to our code and data in the supplementary documents. We also give instructions about how to reproduce the main experimental results in Section B.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Sections 5 and B.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Instead of error bars, we report our experimental results with the mean value and 95% intervals of the results from running the experiments several rounds. We describe the method to obtain the data in Section 5, and the results in Tables 1, 2 and 5- 8.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section B.1.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 5.1.
   (b) Did you mention the license of the assets? [Yes] We mention the license in our code.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include new assets, and they can be found via a URL.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] The datasets are generated with simulations, which has nothing to do with personal data.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] The data is neutral to biological human being.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Discussion

A.1 Indirect connections

Originally, iSIDG starts training with a fully connected graph structure: $\mathbf{A}^{(0)}$. Based on the problem setting of dynamical systems, we assume $\mathbf{A}^{(0)}$ can be decomposed as: $\mathbf{A}^{(0)} = \{\mathbf{A}_D, \mathbf{A}_{IN}, \mathbf{A}_U\}$, where $\mathbf{A}_D$ represents the actual edges in the graph (directed connections), $\mathbf{A}_{IN}$ denotes the indirect connections between the two actual connected nodes, and $\mathbf{A}_U$ is the set of non-connections.

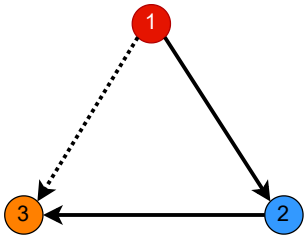

Figure 3: A draft of the direct connections and indirect connections in a three node set. Edges (v_1, v_2) and (v_2, v_3) are direct connections, and edge (v_1, v_3) is the possible indirect connection inferred by structural inference methods.

Previous structural inference methods, such as NRI, fNRI and ACD, are good at eliminating $\mathbf{A}_U$ in the inference results. However, as shown in Figure 3, these methods may falsely reconstruct the structure with indirect connections. It is interesting that the indirect connections resulted from the transmission of signals between nodes. In Figure 3, there exists a chain connection from node v_1 to v_3 through v_2, and in a dynamical system, the changes in v_1 will finally affect v_3, which would mislead the structural inference methods to falsely infer an edge from v_1 to v_3. Moreover, for a sparse graph, it is possible that $\mathbf{A}_{IN} \cap \mathbf{A}_U \neq \emptyset$, which challenges structural inference methods, such as NRI, fNRI and ACD.

A.2 Solution for Eliminating Indirect Connections

Our solution to eliminate indirect connections in the inference results consists of three methodologies: (1) **VAE**; (2) **Iterative process**; and (3) **Regularization terms**.

VAE. As shown in previous works [21, 49, 29], the bottleneck structure of VAEs can correctly identify many non-connections in the reconstructed adjacency matrix. Thus, we also rely on the VAE to firstly identify as many non-connections as possible before the first round of iterative process, and sample $\{\hat{\mathbf{R}}^{(p)}_{\text{row}}, \hat{\mathbf{R}}^{(p)}_{\text{col}}\}$ according to operations shown in Equations 11 - 14.

Iterative process. The iterative process follows the methodology of VIB, and is mentioned in Section 4.4 with a new objective function:

$$\mathbf{Z} = \arg\min_{\mathbf{Z}} I(\mathbf{Z}; V^t, \{\hat{\mathbf{A}}^{(p)}_{\kappa}\}) - \mathfrak{u} \cdot I(\mathbf{Z}; V^{t+1}). \tag{22}$$

The iterative process is characterized by a main advantage: the first term in Equation 22 $I(\mathbf{Z}; V^t, \{\hat{\mathbf{A}}^{(p)}_{\kappa}\}) < I(\mathbf{Z}; V^t, \mathbf{A}^{(0)})$, which results in a tighter bound than Equation 3. In other words, after feeding the direction information back to the input side of the encoder, the new search space excludes the non-connections, which are found by VAE in the previous training epochs. With a smaller search space, iSIDG can focus on the differentiation between misleading indirect connections and the real connections.

Regularization terms. Although feeding the learned adjacency matrix back to the encoder can eliminate the indirect connections, we need regularization terms to secure and accelerate this process. The sparsity term $\mathcal{L}_S$ encourages iSIDG to choose the more solid path from every node, to meet the minimal sufficient statistics assumption of IB. For example, in Figure 3, at node 1, iSIDG has to choose between the path $1 \to 3$ and $1 \to 2 \to 3$ based on the sparsity regularization terms and the minimal sufficient information for dynamics prediction in decoder.

One may argue that it is also possible that iSIDG will falsely think there are edges from node 1 to node 2, and from node 1 to node 3. However, this does not conform to the future state prediction. The task of the decoder is trying to approximate:

$$v_i^{t+1} = v_i^t + \Delta \cdot \sum_{j \in \mathcal{N}_i} f\Big(||v_j^t, v_i^t||_\alpha\Big), \tag{23}$$

where $f(\cdot)$ denotes the interaction from node v_j to node v_i. We may imagine the interaction could be RNA activation level in GRN, spring force in spring-balls system, and the phase-couple of oscillators. As a result, the interactions between nodes are assumed to be the same, and described with the same function. Yet node 1 can only affect node 3 through node 2, which results in a superposition of functions: $f(f(\cdot))$. Therefore, both edges from node 1 to node 2, and from node 1 to node 3 cannot predict the future state of node 3 correctly, and VAE will search for other possible edges.

B Further Implementation Details

B.1 Implementation details of iSIDG

We summarize the described architecture of iSIDG and present the pipeline of training iSIDG in Algorithm 1.

Algorithm 1 General Framework for iSIDG

Input: V, number of nodes n
Parameters: δ, σ, η, μ, α, β, γ, ξ, batch size m and time steps T
Model Weights: ϕ, θ
Output: $\mathbf{A}^{(t)}$
$i \leftarrow 1$
StopCond $\leftarrow \|\mathbf{A}^{[i]} - \mathbf{A}^{[i-1]}\|_F^2 < \delta \|\mathbf{A}^{(0)}\|_F^2$
$\mathbf{A}^{(0)} = \mathbb{1} \in n \times n$
$\mathbf{A}^{(i+1)} = \mathbf{A}^{(0)}$
Split V into $\mathbf{V}^T = \{V^0, \ldots, V^T\}$ and $\mathcal{V}^{T+1} = \{V^1, \ldots, V^{T+1}\}$
while $i < MaxEpoch$ **do**
  $\mathbf{Z} \leftarrow$ Encoder($\mathbf{V}^T$, $\mathbf{A}^{(i)}$, ϕ)
  $\hat{\mathbf{V}}^{T+1} \leftarrow$ Decoder($\mathbf{V}^T$, $\mathbf{Z}$, θ)
  if $i > \eta$ and *StopCond* **then**
    $\tilde{\mathbf{R}}_\kappa^{(i)} \leftarrow \{\mathbf{Z}, \mathbf{A}^{(0)}, \xi\}$ using Eqs. 11 - 15
    $\mathbf{A}^{(i)} \leftarrow \tilde{\mathbf{R}}_\kappa^{(i)}$
    $\mathcal{L}_p \leftarrow \{\hat{\mathbf{V}}^{T+1}, \mathbf{V}^{T+1}, \delta\}$
    $\mathcal{L}_{KL}, \mathcal{L}_d, \mathcal{L}_s \leftarrow \{\mathbf{Z}, n\}$
    $\mathcal{L}^{(i)} \leftarrow \mathcal{L}_p + \mu\mathcal{L}_{KL} + \alpha\Omega(\mathbf{A}^{(t)}, V) + \beta\mathcal{L}_d + \gamma\mathcal{L}_s$
    $\mathcal{L} \leftarrow \mathcal{L}^{(i)}/T$
    Back-propagate $\mathcal{L}$ to update model weights ϕ and θ
    $i \leftarrow i + 1$
  end if
end while
Return: $\mathbf{A}^{(t)}$

Algorithm 2 A MLP block.

Input: features $input$
x = elu(Linear1($input$))
x = dropout(x)
x = elu(Linear2(x))
out = batch_norm(x)
Return: out

Algorithm 3 Pseudocode for MPM encoder.

Input: features $input$
x = mlp1($input$)
x = node2edge(x)
x = mlp2(x)
x_skip = x
x = edge2node(x)
x = mlp3(x)
x = node2edge(x)
x = concatenation([x, x_skip])
x = mlp4(x)
out = fully connected out(x)
Return: out

Basic settings. We implement our iSIDG model in PyTorch [31] with the help of Scikit-Learn [32] to calculate various metrics. We run experiments on a single NVIDIA Tesla V100 SXM2 graphic card, which has 32 GB graphic memory and 5120 NVIDIA CUDA Cores. We attach our pseudocode and implementation as the supplementary document to this paper. During training, we set batch size as 128 for datasets which have less than 10 nodes, for those more than 10 nodes, we set batch size as

64. We train our iSIDG model with 2000 epochs on every dataset, which takes around 30 - 42 hours for every training session.

Loss Functions. The hybrid loss function shown in Equation 21 consists of five different terms to be calculated: $\{\mathcal{L}_p, \mathcal{L}_{KL}, \Omega(\mathbf{Z}, V), \mathcal{L}_d, \mathcal{L}_s\}$. We would like to describe the actual implementations of $\mathcal{L}_p$ and $\mathcal{L}_{KL}$. Since the target of the decoder is the prediction of future node features based on present states, $\mathcal{L}_p$ is estimated by:

$$-\sum_j \sum_{t=2}^{T} \frac{\|v_j^t - \hat{\mathbf{h}}_j^t\|^2}{2\sigma^2} + \text{const}\,, \tag{24}$$

where the variance σ is set as a fixed value, and the const is e^{-16}.

And the calculation of KL-divergence $\mathcal{L}_{KL}$ over a uniform prior can be simplified as the calculation of entropy H:

$$\sum_{i,j} H(q_\theta(z_{ij}|v)) + \text{const}\,. \tag{25}$$

Hyper parameters. The choice of most hyper parameters in our iSIDG model (such as those mentioned in Algorithm 1) may refer to Table 3. In addition to the hyper parameters mentioned in the table, we set the combination coefficient λ in Equation 13 as 0.3 for the training on all of the datasets. We argue that λ can be set as varying values according to the training epochs, and decrease during training, which may be helpful for faster converge. We would like to carry on more research onto this hypothesis in the future. Besides that, we describe the choice of time step length T in Section C. Time step length T decides the amount of samplings along a trajectory.

Table 3: Hyper parameter choices for every dataset.

DATASET	δ	σ	η	μ	α	β	γ	ξ
LI	0.00001	0.00008	150	200	50	10	20	0.5
LL	0.00001	0.00008	150	400	80	20	30	0.5
CY	0.00001	0.00008	150	300	50	10	20	0.5
BF	0.00001	0.00008	150	400	80	20	40	0.4
TF	0.00001	0.00008	150	500	70	20	50	0.4
BF-CV	0.00001	0.00008	150	400	80	20	40	0.4
NetSim1	0.00001	0.00008	180	300	50	10	20	0.5
NetSim2	0.00001	0.00008	180	400	70	20	30	0.5
NetSim3	0.00001	0.00008	180	500	80	30	50	0.5
Springs	0.0001	0.00005	100	100	50	10	20	0.5
Particles	0.0001	0.00005	100	100	50	10	20	0.5
Kuramoto	0.0001	0.00005	100	200	50	10	20	0.5

Ablation study with different GNNs. The main difference between the utilization of different GNNs takes place in the encoder of iSIDG. For the implementation of **MLP** encoder, we use three stacked multi-layer perceptron blocks, with each block consists of the units shown in Algorithm 2. We place a concatenation operation between Block 1 and 2 to get edge representations. The last block is attached with another Linear layer with its dimension matching the number of edges in a full-graph setting.

Table 4: Number of units in the Linear layers in MLP encoder.

BLOCK	Linear 1	Linear 2
1	256	256
2	512	256
3	256	256

The number of units in the Linear layers in the **MLP** encoder is shown in Table 4. The dropout rates in all of the blocks are set as 0.5.

For the simple message-passing mechanism encoder (**MPM** encoder), we follow the design of "MLP Encoder" in NRI [21]. And can be summarized as the pseudocode shown in Algorithm 3.

Algorithm 4 Pseudocode for GIN encoder.

1: **Input:** features $input$
2: x = mlp1($input$)
3: x_skip = node2edge(x)
4: x_skip = mlp2(x_skip)
5: x = f_{GIN}(x)
6: x = mlp3(x)
7: x = node2edge(x)
8: x = concatenation([x, x_skip])
9: x = mlp4(x)
10: out = fully connected out(x)
11: **Return:** out

Algorithm 5 Pseudocode for GAT encoder.

1: **Input:** features $input$
2: x = mlp1($input$)
3: x_skip = node2edge(x)
4: x_skip = mlp2(x_skip)
5: x = f_{GAT}(x)
6: x = mlp3(x)
7: x = node2edge(x)
8: x = concatenation([x, x_skip])
9: x = mlp4(x)
10: out = fully connected out(x)
11: **Return:** out

Based on the design of Graph Identity Networks and the **MPM** encoder, we combine these idea together into the **GIN** encoder, by replacing the operations from line 3 to line 6 in Algorithm 3 with the GIN operation mentioned in [51], but still keep the skip connection for edges representations, and the pseudocode is shown in Algorithm 4. Similarly, if we replace the GIN operation in line 5 of Algorithm 4 with GCN operation, we obtain the **GCN** encoder. Moreover, the design of **GAT** encoder is similar to that of **GIN** encoder and is shown as the pseudocode shown in Algorithm 5.

B.2 Implementation details of baselines

NRI. We use the official implementation code by the author from `https://github.com/ethanfetaya/NRI` with customized data loader for our chosen datasets: synthetic networks, NetSims and physical simulations. We add our metric-evaluation in "test" function, after the calculation of accuracy in the original code.

fNRI. We use the official implementation code by the author from `https://github.com/ekwebb/fNRI` with customized data loader for our chosen three datasets. We add our metric-evaluation in "test" function, after the calculation of accuracy and the selection of correct order for the representations in latent spaces in the original code.

MPIR. We use the official implementation code from `https://github.com/tailintalent/causal` as the code for MPIR. We run the code by customized data loader for the chosen three datasets. After the obtain of the results, we run another script to calculate the metrics.

ACD. We follow the official implementation code by the author as the framework for ACD (`https://github.com/loeweX/AmortizedCausalDiscovery`). We run the code with customized data loader for the chosen three datasets. We implement the metric-calculation pipeline in the "forward_pass_and_eval()" function.

C Further Details about Datasets

C.1 Synthetic networks

The six directed Boolean networks (LI, LL, CY, BF, TF, BF-CV) are the most often observed fragments in many gene regulatory networks, each has 7, 18, 6, 7, 8 and 10 nodes, respectively. Thus by carrying out experiments on these networks, we can get acknowledge about the performance of the chosen methods on the structural inference of real-world biological networks. We collect the six ground-truth directed Boolean networks from [33] and simulate the single cell evolving trajectories with BoolODE [33] `https://github.com/Murali-group/BoolODE` with default settings mentioned in that paper for every network. We sample 12000 trajectories and group them into three datasets: for training, for validation, and for testing, each with the number of 8000, 2000 and 2000, respectively. Then we sample 49 snapshots with equal time interval in every trajectory and save them as ".npy" files for data loading.

C.2 NetSim datasets

The NetSim datasets simulate blood-oxygen-level-dependent imaging data across different regions within the human brain and is described in [41] and `https://www.fmrib.ox.ac.uk/datasets/netsim/`. We target at inferring the existence of directed connections between different brain areas. Among the total 28 datasets in NetSim, we choose the first three datasets (NetSim1, NetSim2 and NetSim3) which have 5, 10, and 15 nodes, respectively. We sample 49 snapshots on each trajectory with equal interval and randomly group them into three sets for training, validation and testing with the ratio of 8: 2: 2, respectively.

C.3 Physical simulations

To generate these physical simulations (springs, charged particles and phase-coupled oscillators), we follow the description of the data in [21] but with fixed interactions. To be specific, at the beginning of the data generation for each physical simulation, we randomly generate a ground truth graph and then simulate 12000 trajectories on the same ground truth graph, but with different initial conditions. The rest settings for the simulations are the same as that mentioned in [21]. It is worth mentioning that the connections in the physical simulations are indirected, which are different from those in the synthetic networks and NetSim datasets. We collect the trajectories and randomly group them into three sets for training, validation and testing with the ratio of 8: 2: 2, respectively.

D Further Details about Experiments

D.1 Metrics

We choose the three most representative metrics from 0-1 classification for the evaluation of structural inference results, namely the area under the receiver operating characteristic (AUROC), the area under the precision-recall curve (AUPRC) and Jaccard index (JI). All of the metrics are calculated from the inferred structure and the full-patch of ground-truth.

AUROC. The area under the receiver operating characteristic (AUROC) is a performance metric to evaluate the result of classification tasks. It demonstrates the model's ability to discriminate between cases (positive examples) and non-cases (negative examples). For our case, it is used to make clear the model's ability to distinguish actual edges and empty interactions. It is calculated as the area under the receiver operating characteristic curve (ROC), which sets false positive rate (FPR) as x-axis and true positive rate (TPR) as y-axis at various threshold settings. In the ROC space, the best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 100% no false negatives and 100% no false positives. Then a model outputs more precise results with many TPR yields a higher AUROC value, and vice versa.

AUPRC. The area under the precision-recall curve (AUPRC) is a useful performance metric for imbalanced data in a problem setting since we care a lot about finding the existence of interactions between nodes. It shows the ability of the model to find the existence of all ground-truth edges without accidentally marking any non-interaction as existence. The AUPRC is calculated as the area under the Precision-recall (PR) curve. The x-axis of a PR curve is the TPR and the y-axis is the precision. This is in contrast to ROC curves, where the y-axis is the TPR and the x-axis is FPR. Similar to calculate AUROC, AUPRC is also calculated from various threshold settings.

Jaccard index. The Jaccard index (sometimes called the Jaccard similarity coefficient) (JI) compares the inferred edges and ground-truth to see the shared results and the distinct results. Jaccard index is a measure of similarity for the two sets of data, with a range from 0% to 100%. The higher the percentage, the more similar the two populations. It is calculated as "Intersection of two sets/Union of two sets". Although JI is easy to calculate and interpret, the result is extremely sensitive to small samples sizes and may produce erroneous results, especially when the number of samples is small or the data sets contain missing observations.

Table 5: AUPRC values (%) of iSIDG and baselines on synthetic networks.

METHOD	LI	LL	CY	BF	TF	BF-CV
iSIDG	**40.3**± 3.4	**18.3**± 2.9	**42.1**± 5.7	**38.0**± 5.0	**46.6**± 5.6	**32.8**± 4.4
NRI	27.3± 3.1	7.30± 1.3	31.1± 4.7	28.2± 3.1	32.7± 3.6	17.1± 3.6
fNRI	29.0± 3.4	8.59± 1.9	25.8± 5.7	29.8± 4.3	33.2± 4.9	20.1± 3.5
MPIR	15.6± 2.6	5.78± 1.4	16.7± 2.5	23.7± 3.3	31.3± 2.1	17.8± 2.0
ACD	21.9± 3.5	14.7± 2.5	29.9± 5.4	23.2± 4.8	34.3± 4.0	24.8± 3.6

Table 6: JI values (%) of iSIDG and baselines on synthetic networks.

METHOD	LI	LL	CY	BF	TF	BF-CV
iSIDG	**45.4**± 3.9	**49.5**± 5.0	**47.2**± 5.5	**46.8**± 4.8	**46.0**± 4.3	**46.6**± 3.6
NRI	41.6± 2.9	37.2± 4.4	36.5± 4.8	38.0± 4.1	35.9± 4.3	40.6± 3.5
fNRI	41.8± 3.0	38.5± 5.7	33.5± 6.2	32.4± 5.1	35.0± 5.0	37.4± 3.4
MPIR	11.9± 1.7	3.70± 0.8	13.3± 1.4	18.4± 2.6	24.5± 2.0	15.6± 1.6
ACD	22.6± 4.5	11.0± 2.6	20.2± 2.8	15.0± 2.2	18.1± 2.5	14.0± 2.8

Table 7: AUPRC and JI values (%) of iSIDG and baselines on NetSim datasets.

	AUPRC			JI		
METHOD	NetSim1	NetSim2	NetSim3	NetSim1	NetSim2	NetSim3
iSIDG	**40.0** ± 4.9	**36.3** ± 4.5	**34.8** ± 5.0	**49.1** ± 4.8	**47.6** ± 5.0	**44.2** ± 4.5
NRI	34.5 ± 2.9	32.4 ± 3.0	32.1 ± 3.3	43.9 ± 3.5	42.3 ± 2.8	41.2 ± 2.9
fNRI	33.1 ± 3.7	30.8 ± 3.8	30.2 ± 4.1	44.1 ± 4.2	40.9 ± 4.0	41.7 ± 3.6
MPIR	25.2 ± 2.2	23.7 ± 3.1	21.9 ± 3.0	25.4 ± 1.3	24.1 ± 2.6	23.5 ± 2.9
ACD	32.9 ± 2.6	30.7 ± 3.5	30.2 ± 2.1	26.8 ± 1.7	25.6 ± 2.4	23.7 ± 3.5

Table 8: AUPRC and JI values (%) of iSIDG and baselines on physical simulations.

	AUPRC			JI		
METHOD	Springs	Particles	Kuramoto	Springs	Particles	Kuramoto
iSIDG	**80.7**± 6.7	70.6± 4.9	71.8± 5.4	81.8± 4.2	72.6± 4.1	71.3± 3.7
NRI	79.8± 4.4	69.4± 2.1	71.7± 4.6	80.4± 2.8	72.2± 2.6	71.3± 3.3
fNRI	**80.7**± 5.7	**72.6**± 3.2	**73.9**± 6.8	**82.2**± 2.8	**74.6**± 2.4	**73.4**± 3.7
MPIR	35.5± 5.2	32.4± 3.9	39.0± 3.4	33.7± 4.8	30.4± 4.2	37.3± 3.1
ACD	80.2± 6.1	58.4± 3.6	70.4± 5.7	80.0± 2.6	57.3± 4.7	69.4± 5.8

D.2 Further experimental results

In this section, we demonstrate additional experimental results as the supplement to Section 5. We present AUPRC and JI values as well as a case study about the structural inference results of iSIDG and baseline methods.

AUPRC and JI results. We present the AUPRC and JI results of iSIDG and baseline methods on synthetic networks, NetSim datasets and physical simulations in Tables 5 - 8. The results are the mean values and 95% intervals of 15 experiments, respectively. Similar to the AUROC results presented in Section 5.1, iSIDG performs the best among all of the chosen methods in datasets with directed graphs, and can match the best baseline method in datasets with indirected graphs.

Besides that, we conduct a case study of iSIDG with different objective functions, and calculate the average accumulated path lengths based on 15 experiments of each on Linear network dataset. The results are shown in Figure 4. It is clear that our choice of objective function with regularization terms of smoothness, connectiveness and sparsity successfully encourages iSIDG model to infer the structure of the system with less indirect interactions and more accurate results.

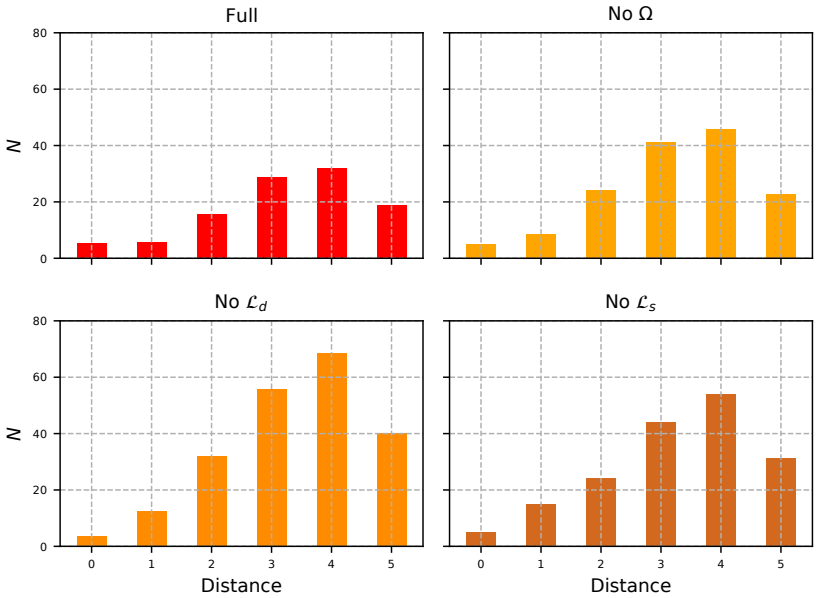

Figure 4: Averaged accumulated length of the connections between nodes of the inference results on Linear network dataset with different objective functions. The y-axis represents the averaged counts of paths according to their length, and the x-axis denotes the path length between two nodes. The results are the mean values of 15 experiments on Linear network dataset with different objective functions.

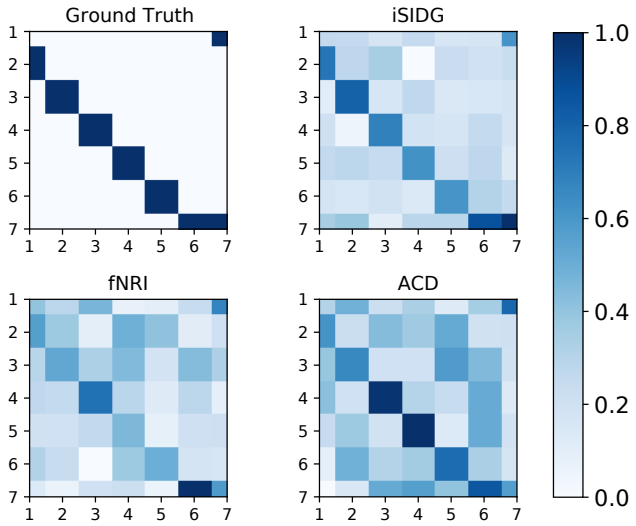

Figure 5: Reconstruction results of different methods on Linear network dataset.

Case study. To gain an intuitive understanding of the results of the structural inference of directed graphs, we conduct a case study on the Linear network dataset. The reconstructed adjacency matrices of our iSIDG, fNRI and ACD methods on Linear network dataset are visualized in Figure 5, as well as the ground-truth. The results are the mean results of 15 experiments of each method and are shown in terms of distributions of reconstructed relations. In the figure, 0.0 at a point $[x, y]$ denotes the absence of interaction from x to y, and 1.0 represents in an opposite way. As shown in Figure 5, compared with the ground-truth, the structure reconstructed by ACD contains many indirect

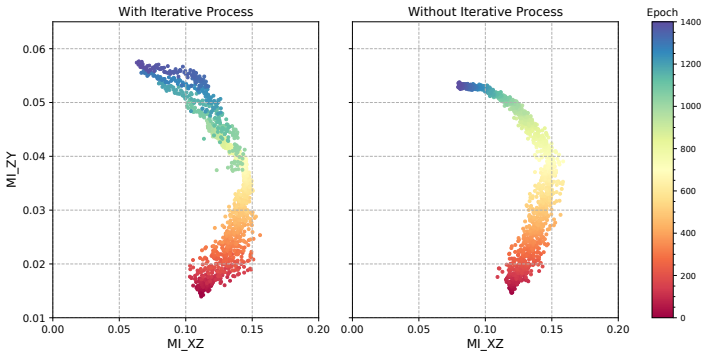

Figure 6: Mutual information comparison between methods with or without iterative process on Linear network dataset.

connections. It is possibly the result from the fact that ACD infers a latent posterior graph from the input features based on hidden confounding [48]. For Linear network dataset, which has a “chain" structure, it may confuse ACD method as it contains many indirect interactions, and thus leads to inaccurate inference results. Moreover, fNRI utilizes a factorized latent space for every possible interaction types. For the case here, it even reserves a latent space for “Non-interaction" type, which further disentangles possible indirect interactions from direct ones, and therefore slightly promotes its accuracy. In contrast, the iterative training process in iSIDG creates relaxation of the gradients during training. As a result, it can distinguish most indirect interactions from the direct ones and reconstruct the adjacency matrix more precisely than the other methods on Linear network dataset.

D.3 Mutual information

We conduct another case study to demonstrate the difference of the mutual information between our iSIDG with and without the iterative process on Linear network dataset with 10 experiments of every model with 1400 training epochs. We utilize nearest-neighbor MI estimator [34] and the plots are shown in Figure 6. The plots show the mutual information between input features and embeddings in latent space MI_XZ for every edge, as well as the mutual information between embeddings in latent space and the output features MI_ZY for every edge. Since the input consists of node features, we concatenate them according to [sender, receiver] to represent each edge for the calculation of mutual information, respectively.

As shown in Figure 6, in both methods, the two types of mutual information increase in the first phase, and then after around 800 epochs of training, the mutual information between embeddings in latent space and output features decreases, while the other one keeps growing. These phenomena verify the two procedures in the training of deep neural networks as mentioned in the IB theory [40, 1], namely label-fitting and representation-compression.

In the phase of representation-compression, when the stop condition is reached, the method with the iterative process stops the current training round, feeds the obtained adjacency matrix from latent space back to the input side of encoder, and starts another round of training with the obtained adjacency matrix. The phase-transition is demonstrated by the decrease of both mutual information and start with another round of label-fitting and representation-compression. Therefore, the iterative training process creates several rounds of relaxation during training, and encourages our iSIDG model to learn a more compressed representation in the latent space. As shown in Figure 5, the compressed representation in the latent space eliminates most of the indirect interactions and promotes the structural inference accuracy of our iSIDG model.

D.4 Ablation studies on different encoders and decoders

We conduct another series of ablation studies to demonstrate the different choices of encoders and decoders in our iSIDG model. We choose four methods as encoders for our ablation study,

Table 9: AUROC values (%) of iSIDG with different encoders and decoders on LI dataset.

	MLP	MULTI	RNN
MLP	64.8 ± 4.9	60.6 ± 4.9	60.9 ± 4.8
MPM	75.5 ± 3.3	70.4 ± 5.4	71.6 ± 3.8
GCN	81.0 ± 4.7	76.5 ± 4.0	76.3 ± 4.6
GIN	**86.2 ± 4.2**	79.8 ± 4.4	79.5 ± 4.8
GAT	65.9 ± 3.8	63.0 ± 4.3	62.7 ± 3.9

Table 10: AUROC values (%) of iSIDG and baselines on "Springs100" and ESC datasets.

METHOD	Springs100	ESC
iSIDG	**72.7± 6.8**	**56.7± 5.1**
NRI	67.1± 5.9	39.2± 5.0
fNRI	68.4± 5.5	39.8± 4.5
MPIR	35.6± 3.2	30.6± 6.2
ACD	66.0± 5.7	38.7± 4.2

namely: multi-layer-perceptrons (MLP), simple message-passing mechanism (MPM), GCN, GIN, and GAT. We also take three different decoders into account: MLP, avoiding degenerate decoder (MULTI) [21][49], and recurrent decoder (RNN) [21]. The results are shown in Table 9, where the rows denote the choice of encoder, the columns for decoder, and the results are the average values and 95% intervals of 15 experiments with every combination of encoder and decoders. As shown in Table 9, the methods with GNNs as encoder are greatly more accurate than the ones with simple MLP encoder. The only exception are the ones with GAT encoder, which are only marginally superior to the ones with MLP encoder. The reason may be that GAT is prone to over-fitting and the attention weights lack supervision in the cases of structural inference. When it comes to GIN as encoder, the results are superior to methods with other encoders by at least 3.2 percent of AUROC. Surprisingly, methods with MLP decoder, which is the simplest decoder among the three, output the most accurate reconstruction results. We argue that it may be due to the simple task of the iSIDG: only focusing on structural inference instead of both structural inference and dynamic prediction (as that of NRI).

E Experiments on Scalability

Although iSIDG works well on small datasets, such as synthetic networks, physical simulations and NetSim datasets, we are curious about the performance of iSIDG on large datasets. For this reason, we create another "springs" dataset with 100 nodes (we name it as "Springs100"), and with the same settings as the details mentioned in Section C.3. Besides that, we also test our model and baselines on the ESC dataset [4], which is a GRN with 98 nodes and 409 edges. We ran the experiments for ten rounds, and summarize the AUROC results in Table 10. As shown in the table, our iSIDG still outperforms baseline methods on these two large-scale datasets.

However, we think that iSIDG cannot prove its scalability in a broader view. The reasons are the following:

1. Despite the iSIDG can work better than baseline methods on "Springs100" dataset, the AUROC only values 72.7%, which makes the inferred results to be highly unauthentic for downstream tasks.
2. On ESC dataset, iSIDG can only work better than guessing the existence of edges with equal probability (either exist or not). So that the results are not reliable.
3. Compared with the results shown in Section 5, we observe a significant decline, which suggests that we can extrapolate an even worse inference result on datasets with more nodes.
4. Because the latent space in iSIDG has a fixed size, which means that we have to adopt a larger latent space for larger graphs. Thus the scalability of the iSIDG is limited by VAE as well.

Therefore, we suggest that iSIDG can be utilized to infer the structure of dynamical systems with tens of nodes. We will investigate the problem of scalability in structural inference in our future work, which is also interesting and of practical value.

Table 11: AUROC values (%) of DAG-GNN and DAG-NoCurl on synthetic networks

METHOD	LI	LL	CY	BF	TF	BF-CV
DAG-GNN	42.3± 2.0	40.5± 3.1	42.9± 4.5	43.2 ± 3.4	40.8 ± 2.8	40.5 ± 3.7
DAG-GNN*	40.0± 5.1	46.8± 3.6	44.0± 5.7	50.3± 4.5	50.2± 3.6	51.7± 4.1
DAG-NoCurl	43.0± 2.7	43.2± 3.3	49.5± 4.4	51.8± 3.2	39.4± 5.5	51.1± 3.2

Table 12: AUROC (%) of DAG-GNN and DAG-NoCurl on physical simulations and NetSim datasets.

METHOD	Springs	Particles	Kuramoto	NetSim1	NetSim2	NetSim3
DAG-GNN	48.2± 2.3	40.0± 3.4	45.5± 4.5	43.1 ± 3.4	42.4 ± 4.9	40.9 ± 3.6
DAG-GNN*	39.5± 2.7	38.4± 2.1	38.5± 3.3	35.4 ± 3.4	34.0 ± 4.6	33.2 ± 3.9
DAG-NoCurl	47.4± 2.6	44.8± 3.6	47.0± 4.7	49.5 ± 3.1	47.2 ± 3.6	46.5 ± 3.3

F Do DAG-structure Learning Methods Work in Our Problem Setting?

We observe that there is a series of works, which deal with the directed acyclic graphs (DAG) structure learning [56, 54, 55]. These works formulate the problem as a continuous optimization with a structural constraint that ensures acyclicity [56], with explicit structural constraints [54], or with implicit mechanism to force the acyclicity [55]. Although it is claimed by the authors that these work deal with DAG structure learning, which is different from our problem setting, we are still curious about the performance of these works on the problem of structural inference of dynamical systems.

We choose DAG-GNN [54] and DAG-NoCurl [55], which are the representative in the research field of DAG structure learning. We follow the official implementations of these models:

- DAG-GNN: `https://github.com/fishmoon1234/DAG-GNN`;
- DAG-NoCurl: `https://github.com/fishmoon1234/DAG-NoCurl`;

and we only change the data loader modules to load physical simulations, synthetic networks and NetSims datasets, respectively. We ran the experiments for ten rounds, and summarize the AUROC results in Tables 11 and 12, where we note DAG-GNN* as the DAG-GNN with acyclicity constraint set as zero. As shown in the tables, DAG-GNN, DAG-GNN* and DAG-NoCurl fail to infer the existence of interactions in dynamical systems. The reason may be that the existence of cycles in the systems, which is common observed among dynamical systems, violates the acyclicity assumptions of these methods. In particularly, although the acyclicity constraint in DAG-GNN* is set as zero, DAG-GNN* is unable to correctly infer the structure of our datasets, and (un)surprisingly performs even worse than DAG-GNN.

G Broader Impact

Methods such as iSIDG for structural inference of dynamical systems allow for numerous researchers in the field of physics, chemistry and biology to study the interactions inside the systems. We have shown that iSIDG works well on either one-dimensional node features or multi-dimensional features, where the features are continuous variables, which proves its wide application. While the emergence of the structural inference technology may be extremely helpful for many, it has the potential for misuse. Potentially, iSIDG can be extended to infer the online social connections via measuring mutual information, which could erode privacy.