Link Level Implicit Graph Neural Networks

Anonymous Authors[1]

Abstract

We propose link level implicit graph neural networks (LIGNN), a deep equilibrium model that utilizes node pairs as message passing units to predict link properties at its fixed point. Given the input graph, LIGNN recursively invokes a contractive linear mapping to compute the fixed point, where the space occupation grows linearly w.r.t. the graph size and the amortized computation time for each link grows linearly w.r.t. average node degrees. We evaluate the performance by applying it to simulate the naïve evaluation of Datalog programs and various general link prediction benchmarks. LIGNN shows competitive performance for various link prediction tasks, and is capable for capturing long-range and recursive dependencies.

1. Introduction

Link prediction is a fundamental task in graph machine learning fields, and has been applied for solving various problems including knowledge graph completion (KGC) (Nickel et al., 2015), recommender systems (Koren et al., 2009), drug discovery (Ioannidis et al., 2020), influence detection (Nguyen et al., 2021), etc. Often, the solutions are implied by some latent mechanisms, e.g., KGC can often be described with a set of fuzzy logic rules, and molecular properties are implied by its chemical structures. Recently, researchers employ deep learning methods to learn such latent mechanisms, and have achieved impressive success with methods including graph neural networks (GNNs) (Schlichtkrull et al., 2017; Zhang & Chen, 2018; Teru et al., 2019; Liu et al., 2021), knowledge graph embedding (KGE) (Bordes et al., 2013; Yang et al., 2014; Sun et al., 2018), etc.

However, there are some defects with the previous efforts of deep learning methods for link predictions, that is, the inability of capturing recursive dependencies and equilibrium point properties. For example, logical properties can be described with Datalog programs (Abiteboul et al., 1994), whose solution is given by the fixed point of the minimal model (Ceri et al., 1989); Molecular structured are determined by solving the quantum mechanical equations (González et al., 2019). Since GNNs for link prediction heavily relies on explicitly defined layers and can only approximate a fixed number of iterations, they are unsuitable to such complex reasoning tasks. Besides, KGE cannot be applied to inductive setting due to the utilization of node embeddings. Rule-based methods cannot learn real-valued edge features.

To overcome the deficiencies, we propose link level graph neural networks (LIGNN), a deep learning link prediction model that can learn such latent mechanisms implied in the graphs. LIGNN is able to capture recursive dependencies (the solution incorporates recursive computation of immediate results), long-range dependencies (source and target nodes are more than 10-hops away) via fixed point iterations. To summarize, our main contributions are as follows:

- We propose a graph neural network layer with node pairs as message passing units. The required space grows linearly w.r.t. the number of nodes and edges in the graph, and the amortized computational time for each link grows linearly w.r.t. average node degrees.
- We extend the ideas of implicit layers to link level GNNs. With implicitly defined layers, our model is able to capture long-range dependencies and the convergency is guaranteed.
- We discuss some invariant properties of LIGNN, and show that LIGNN produces consistent predictions under different graph settings.

Paper outline In Section 2, we introduce the background and some notations. In Section 3, we introduce how we implement the trainable layers of the model. In Section 4, we introduce the LIGNN model and discuss how we compute the forward / backward pass. In Section 5 we discuss about some properties of the model. In Section 6, we discuss some related works including GNNs, implicit models, etc. In Section 7, we empirically test our model with both synthetic and real-world datasets.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2. Preliminaries

2.1. Problem Statement

We study the problem of link prediction. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$, the target is to predict the link property $\boldsymbol{y}_{s,t}$ of any node pair $(s,t) \in \mathcal{V} \times \mathcal{V}$. We assume each edge $(s,t) \in \mathcal{E}$ is assigned with a feature vector $\boldsymbol{x}_{s,t} \in \mathbb{R}^d$ as inputs (e.g., a one-hot vector indicating the edge type). For $(s,t) \notin \mathcal{E}$ absent in the graph, we simply set $\boldsymbol{x}_{s,t} = \boldsymbol{0}^d$. We assume nodes are indexed with integers $\mathcal{V} = \{1, 2, ..., N\}$ and let $M = |\mathcal{E}|$ be the number of edges.

We use bold uppercases $\boldsymbol{A}, \boldsymbol{B}, ...$ to represent matrices and $\mathbf{A}, \mathbf{B}, ...$ for tensors. Given a matrix $\boldsymbol{A}$ / tensor $\mathbf{A}$, we denote $\boldsymbol{A}[i,j]$ as the (i,j)-th element of $\boldsymbol{A}$. Tensor slices are expressed with colons, for example $\boldsymbol{A}[i, j:k] = [\boldsymbol{A}[i,j], \boldsymbol{A}[i,j+1], ..., \boldsymbol{A}[i,k]]^T \in \mathbb{R}^{k-j+1}$ is a vector, and $\mathbf{A}[i, :j, :k] = \mathbf{A}[i, 1:j, 1:k] \in \mathbb{R}^{j \times k}$ is a matrix.

2.2. Message passing for link prediction

We briefly explain a general iterative mechanism for computing link properties $\{\boldsymbol{y}_{s,t}\}$. The general scheme of iterative methods follows the message passing procedure, and is given as follows:

$$\begin{aligned}
\boldsymbol{z}_{s,t}^{(0)} &= f_I(\boldsymbol{x}_{s,t}), \\
\boldsymbol{z}_{s,t}^{(l+1)} &= \mathcal{U}\left(\boldsymbol{z}_{s,t}^{(l)}, \boldsymbol{z}_{N(s,t)}^{(l)}\right), && (1) \\
\boldsymbol{z}_{N(s,t)}^{(l)} &= \mathcal{A}\left(\left\{\left\{\boldsymbol{z}_{N_e(s,t)}^{(l)} \mid e \in \mathcal{V}\right\}\right\}\right), && (2) \\
\boldsymbol{z}_{N_e(s,t)}^{(l)} &= \mathcal{M}\left(\boldsymbol{z}_{s,e}^{(l)}, \boldsymbol{z}_{e,t}^{(l)}\right), && (3) \\
\boldsymbol{y}_{s,t} &= \lim_{l \to \infty} f_O(\boldsymbol{z}_{s,t}^{(l)})
\end{aligned}$$

where $\boldsymbol{z}_{N(s,t)}^{(l)}$ is the representation of $N(s,t)$, which is the neighbors of the node pair (s,t). $\boldsymbol{z}_{N_e(s,t)}^{(l)}$ is the representation of $N_e(s,t)$, which is the e-th neighbor of (s,t). These notations follows the 2-FWL setting (Cai et al., 1989) (see Appendix H for more explanation). f_I, f_O are problem-specific encoding and decoding functions respectively, $\mathcal{U}, \mathcal{A}, \mathcal{M}$ are problem-specific iterative mappings that update the hidden representations $\{\boldsymbol{z}_{s,t}^{(l)}\}$. They are such functions that for all $s, t \in \mathcal{V}$, the infinite sequence $(\boldsymbol{z}_{s,t}^{(l)})$ converges to the fixed point $\boldsymbol{z}_{s,t}^*$, and the solutions $\{\boldsymbol{y}_{s,t}\}$ are obtained by decoding these fixed points. In real world, many graph-related problems are solved with such mechanisms. For example, the 2-FWL graph isomorphism test which shares the same discriminative power with 3-WL, is an instance of the mechanism when all functions are injective (Grohe, 2017; Maron et al., 2019). Also, by specifying different iterative mappings, this mechanism can also express concepts including graph connectivity, shortest path, logical reasoning, etc.

3. Implementation of trainable layers

Our proposed LIGNN model is an instantiation of the above iterative mechanism where the problem-specific functions $f_I, f_O, \mathcal{U}, \mathcal{A}, \mathcal{M}$ are learnt from the inputs $\{\boldsymbol{x}_{s,t}\}$ and the fixed point solutions $\{\boldsymbol{y}_{s,t}\}$. We first discuss how we implement the iterative mappings $\mathcal{U}, \mathcal{A}, \mathcal{M}$ as trainable layers.

3.1. Parameterizing iterative mappings

To design a trainable layer based on Eq. 1-3, we implement $\mathcal{U}, \mathcal{A}, \mathcal{M}$ with trainable parameters. Previous studies on node level GNNs have brought up extensive implementations of these iterative functions, and we extend these ideas to link level where node pairs are message passing units. We focus on the simplicity and intuition of the approach.

For $\mathcal{U}$, we borrow ideas from previous node level GNNs such as GCN (Kipf & Welling, 2016a) and SGC (Wu et al., 2019). A simple and common method is to initialize $\mathcal{U}$ as a weighted sum:

$$\mathcal{U}\left(\boldsymbol{z}_{s,t}^{(l)}, \boldsymbol{z}_{N(s,t)}^{(l)}\right) = \boldsymbol{W}_1 \boldsymbol{z}_{N(s,t)}^{(l)} + \boldsymbol{W}_2 \boldsymbol{z}_{s,t}^{(l)}, \quad (4)$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{m \times r}, \boldsymbol{W}_2 \in \mathbb{R}^{m \times d}$ are weight matrices. $\boldsymbol{z}_{N(s,t)}^{(l)} \in \mathbb{R}^r$ is the hidden representation for all the neighbors of the tuple (s,t). Note that for GCNs, the term $\boldsymbol{z}_{s,t}^{(l)}$ can be absorbed into the $\boldsymbol{z}_{N(s,t)}^{(l)}$ term by adding self-loops, but we explicitly write it down here due to the inherently different structures of $\boldsymbol{z}_{s,t}^{(l)}$ and each of its neighbors $\boldsymbol{z}_{N_e(s,t)}^{(l)}$.

For $\mathcal{A}$, it is common to implement it as a summation. Though this implementation is not injective, it is applied in most GNN frameworks and has shown great efficiency:

$$\mathcal{A}\left(\left\{\left\{\boldsymbol{z}_{N_e(s,t)}^{(l)} \mid e \in \mathcal{V}\right\}\right\}\right) = \sum_{e \in \mathcal{V}} \boldsymbol{z}_{N_e(s,t)}^{(l)}. \quad (5)$$

The tricky part is the implementation of $\mathcal{M}$. Unlike node level GNNs, each neighbor of $\boldsymbol{z}_{s,t}^{(l)}$ is now a pair $(\boldsymbol{z}_{s,e}^{(l)}, \boldsymbol{z}_{e,t}^{(l)})$. Previous studies on knowledge graph embeddings (KGE) have brought up plenty of implementations of combining link representations, and many (e.g. DistMult(Yang et al., 2014), ComplEx(Trouillon et al., 2016), RotatE(Sun et al., 2018), etc.) choose the Hadamard production due to its inherent logical expressiveness:

$$\mathcal{M}\left(\boldsymbol{z}_{s,e}^{(l)}, \boldsymbol{z}_{e,t}^{(l)}\right) = \left(\boldsymbol{W}_3 \boldsymbol{z}_{s,e}^{(l)}\right) \odot \left(\boldsymbol{W}_4 \boldsymbol{z}_{e,t}^{(l)}\right), \quad (6)$$

where $\odot$ is the Hadamard production, and weight matrices $\boldsymbol{W}_3, \boldsymbol{W}_4 \in \mathbb{R}^{r \times m}$ are used to obtain head and tail representations, respectively. Putting these together, we obtain

$$\boldsymbol{z}_{s,t}^{(l+1)} = \boldsymbol{W}_1 \sum_{e \in \mathcal{V}} \left(\boldsymbol{W}_3 \boldsymbol{z}_{s,e}^{(l)}\right) \odot \left(\boldsymbol{W}_4 \boldsymbol{z}_{e,t}^{(l)}\right) + \boldsymbol{W}_2 \boldsymbol{z}_{s,t}^{(l)}. \quad (7)$$

which is a nonlinear system that implements $\mathcal{U}, \mathcal{M}, \mathcal{A}$ with trainable weight matrices $\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3, \boldsymbol{W}_4$.

3.2. Linearizing the layers

Eq. 7 provides a link-level GNN layer for computing representations $\boldsymbol{z}_{s,t}^{(l)}$ for each node pair (s,t), but there are some deficiencies of this formulation. For one, the computational complexity is too high: the algorithm takes $\mathcal{O}(N^2)$ memory and the amortized computation time for each link is $\mathcal{O}(N)$. For another, $\mathcal{M}$ involves quadratic terms of inputs and is less stable during training. To overcome the deficiencies, we need to simplify the layers. Since we do not want the simplification to cause the loss of the discriminative power, we first describe the concept of discriminative power, and then we propose our simplification method to linearize the layers and show that this change does not lose its maximum expressiveness.

Definition 3.1. Given a graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ and any $s,t\in\mathcal{V}$, f is a link encoder if $f(\mathcal{G},s,t)\in\mathbb{S}$ maps each node pair in a graph to the embedding space $\mathbb{S}$. Given two link encoders f_1, f_2, if the following statement holds for any graphs $\mathcal{G}_1=(\mathcal{V}_1,\mathcal{E}_1)$, $\mathcal{G}_2=(\mathcal{V}_2,\mathcal{E}_2)$ and node pairs $s,t\in\mathcal{V}_1, p,q\in\mathcal{V}_2$:

$$f_1(\mathcal{G}_1,s,t)=f_1(\mathcal{G}_2,p,q)\Rightarrow f_2(\mathcal{G}_1,s,t)=f_2(\mathcal{G}_2,p,q),$$

then we say the discriminative power of f_1 is greater than or equal to f_2, denoted as $f_1\succeq f_2$. f_1 and f_2 share the equal discriminative power if $f_1\succeq f_2$ and $f_2\succeq f_1$.

The discriminative power indicates the model's ability of detecting different graph structures. A question is, can we simplify the parameterization in Eq. 3 while preserving its discriminative power? The solution is as follows:

Theorem 3.2. *Consider the nonlinear system in Eq. 7. Let $\boldsymbol{z}_{s,t}^{(0)}=f_I(\boldsymbol{x}_{s,t})$ for any s,t be the initial inputs at layer 0. Then, the outputs at each layer correspond to link encoders $f_1^{(0)}, f_1^{(1)}, \ldots$ defined in Definition 3.1, where we have $f_1^{(l)}(\mathcal{G},s,t)=\boldsymbol{z}_{s,t}^{(l)}$ for any $\mathcal{G},s,t$.*

Similarly, let $f_2^{(0)}, f_2^{(1)}, \ldots$ be the encoders induced by a linear system where each layer is defined by

$$\boldsymbol{z}_{s,t}^{(l+1)}=\boldsymbol{W}_1'\sum_{e\in\mathcal{V}}\left(\boldsymbol{W}_3'\boldsymbol{z}_{s,e}^{(l)}\right)\odot(\boldsymbol{W}_4'\boldsymbol{x}_{e,t})+\boldsymbol{W}_2'\boldsymbol{z}_{s,t}^{(l)}. \quad (8)$$

Then, the upper bound of discriminative powers of $\{f_1^{(L)}\},\{f_2^{(l)}\}$ are equivalent. That is, for any $f_1^{(l)}$, there always exists $f_2^{(L)}$, such that $f_2^{(L)}\succeq f_1^{(l)}$, and vice versa.

To prove the theorem and provide more insights of the algorithm, we can show that the predictions of both systems in Eq. 7 and 8 can be explained by a set of weighted horn clauses, and thus the maximum discriminative powers of $\{f_1^{(l)}\},\{f_2^{(l)}\}$ are the same as all such clauses. See Appendix A for the complete proof. Therefore, Eq. 8 is the linear mapping that preserves the expressiveness of the nonlinear system 7.

4. The proposed model

Now we discuss how we design LIGNN with the linear system in Eq. 8. LIGNN is a function with $\{\boldsymbol{x}_{s,t}\}$ as inputs, meaning that it always produces the same fixed point outputs $\{\boldsymbol{y}_{s,t}\}$ with arbitrary choice of initial hidden representations $\{\boldsymbol{z}_{s,t}^{(0)}\}$ (Later in Corollary 4.4 we show this property).

4.1. A compact formulation

A benefit of linearizing the layers is that they are now equivalent to a general feature propagation procedure. Let $\mathbf{A}\in\mathbb{R}^{N\times N\times D}$ denote the augmented edge feature tensor, where $D=d+1$ and

$$\mathbf{A}[s,t,:d]=\boldsymbol{x}_{s,t},\quad \mathbf{A}[s,t,D]=\begin{cases}1, & s=t,\\ 0, & \text{else.}\end{cases}$$

Let $\boldsymbol{Z}_s^{(l)}\in\mathbb{R}^{N\times m}$ be the collection of all $\boldsymbol{z}_{s,t}^{(l)}$ with the same source s, $\boldsymbol{Z}_s^{(l)}[t,:]=\boldsymbol{z}_{s,t}^{(l)}$, then we can directly derive from Eq. 8 that

$$\boldsymbol{Z}_s^{(l+1)}=\sum_{i=1}^{D}\boldsymbol{A}_i\boldsymbol{Z}_s^{(l)}\hat{\boldsymbol{\Theta}}_i,$$

where $\boldsymbol{A}_i=\mathbf{A}[:,:,i]$, $\boldsymbol{w}_i=\boldsymbol{W}_4'[:,i]$, and

$$\hat{\boldsymbol{\Theta}}_i=\begin{cases}\boldsymbol{W}_1'\mathrm{Diag}(\boldsymbol{w}_i)\boldsymbol{W}_3', & i<D,\\ \boldsymbol{W}_2', & i=D.\end{cases} \quad (9)$$

This formulation implies a intuitive interpretation of LIGNN: at each layer, LIGNN propagates the tail nodes t of learnt link representations $\boldsymbol{z}_{s,t}^{(l)}$ through the graph edges. Assume the model is to predict all links with the same source $(s,?)$, then each layer is essentially equivalent to the evaluation of a GCN layer with multiple channels.

For more simplification, we can set each $\hat{\boldsymbol{\Theta}}_i=\boldsymbol{\Theta}_i$ to be directly trainable weight matrices, removing the redundant computation in Eq. 9. With the following proposition we can see that this difference does not change the discriminative power of the layers.

Proposition 4.1. *By setting $\{\hat{\boldsymbol{\Theta}}_i\}$ in Eq. 9 to be trainable weight matrices, the discriminative power upper bound of the model remains unchanged. Moreover, given any $\boldsymbol{W}_1'\in\mathbb{R}^{m\times r},\boldsymbol{W}_2'\in\mathbb{R}^{m\times m},\boldsymbol{W}_3'\in\mathbb{R}^{r\times m},\boldsymbol{W}_4'\in\mathbb{R}^{r\times d}$, there always exists $\{\boldsymbol{\Theta}_i\in\mathbb{R}^{m\times m}\}$, such that the simplified model (i.e. $\hat{\boldsymbol{\Theta}}_i=\boldsymbol{\Theta}_i$ are trainable parameters) and the original one (i.e., $\hat{\boldsymbol{\Theta}}_i$ defined by Eq. 9) produce the same infinite sequence $\boldsymbol{Z}_s^{(0)},\boldsymbol{Z}_s^{(1)},\ldots$. Vice versa.*

Thus, we can express an arbitrary dimension r of message units into d matrices of the size $m\times m$. The resulting model is similar with GCNs, as in each layer $\boldsymbol{z}_{s,t}^{(l)}$ propagates through the graph adjacency matrix as if $\boldsymbol{z}_{s,t}^{(l)}$ was the node representation for t.

4.2. Normalization

To ensure convergence, we introduce the concepts of normalizing the model. For a vector $\boldsymbol{a} \in \mathbb{R}^D$, let

$$\|\boldsymbol{a}\|_\Theta = \left\| \sum_{i=1}^{D} \boldsymbol{a}[i]\boldsymbol{\Theta}_i \right\|_F$$

denote the norm of $\boldsymbol{a}$ under the basis $\{\boldsymbol{\Theta}_1, ..., \boldsymbol{\Theta}_D\}$. Let

$$\tilde{\mathbf{A}}[s,t,:] = \frac{\mathbf{A}[s,t,:]}{\|\mathbf{A}[s,t,:]\|_\Theta + \epsilon}$$

with arbitrary small $\epsilon > 0$ to be the normalized edge features and denote $\tilde{\boldsymbol{A}}_i = \tilde{\mathbf{A}}[:,:,i]$. Let $\{\boldsymbol{S}_i\}$ denote the random walk normalized adjacency matrices

$$\boldsymbol{S}_i = \boldsymbol{D}^{-1}\tilde{\boldsymbol{A}}_i,$$

where $\boldsymbol{D} = \mathrm{Diag}([\deg_{\mathcal{O}}(1), ..., \deg_{\mathcal{O}}(\mathrm{N})])$ is the out-degree matrix. The convergency of the model is summarized as follows:

Proposition 4.2. *The infinite sequence $\{\boldsymbol{Z}^{(l)}\}_l$ induced by*

$$\boldsymbol{Z}^{(l+1)} = \sum_{i=1}^{D} \boldsymbol{S}_i \boldsymbol{Z}^{(l)} \boldsymbol{\Theta}_i + \boldsymbol{B}$$

is guaranteed to converge at the same fixed point $\boldsymbol{Z}^*$ given any bias matrix $\boldsymbol{B} \in \mathbb{R}^{N\times m}$, regardless of the initial point $\boldsymbol{Z}^{(0)}$.

Remark 4.3. Here, the normalized matrices $\{\boldsymbol{S}_i\}$ are relevant to the parameters $\{\boldsymbol{\Theta}_i\}$. There are also other methods to guarantee the model converges while keeping $\{\boldsymbol{S}_i\}$ and $\{\boldsymbol{\Theta}_i\}$ irrelevant, but they are not as efficient for capturing long-range dependencies. See Appendix F for more discussion.

4.3. The proposed LIGNN model

We are now ready to introduce the LIGNN model. From Proposion 4.2 we can see that the fixed point is irrelevant to the initial points. As a result, to incorporate information from edge features into the fixed point, we adopt a residual connection where each link representation $\boldsymbol{z}_{s,t}^{(l)}$ has direct access to their corresponding edge feature $\boldsymbol{x}_{s,t}$.

Let $\boldsymbol{y}_{s,t} \in \mathbb{R}^{d_{\text{out}}}$ be the pre-softmax output for (s,t). Let $\boldsymbol{Y}_s \in \mathbb{R}^{N\times d_{\text{out}}}$ be the collection of $\boldsymbol{y}_{s,e}$ with the same source s where $\boldsymbol{Y}_s[e,:] = \boldsymbol{y}_{s,e}$. We have:

$$\begin{aligned} \boldsymbol{Y}_s &= \boldsymbol{Z}_s \boldsymbol{F}, \\ \boldsymbol{Z}_s &= \gamma \sum_{i=1}^{D} \boldsymbol{S}_i \boldsymbol{Z}_s \boldsymbol{\Theta}_i + \boldsymbol{X}_s \boldsymbol{W}, \end{aligned} \tag{10}$$

with arbitrary small $\epsilon > 0$, and $\gamma \in (0,1]$. $\boldsymbol{F} \in \mathbb{R}^{m\times d_{\text{out}}}, \boldsymbol{W} \in \mathbb{R}^{D\times m}, \boldsymbol{\Theta}_i \in \mathbb{R}^{m\times m}$ are trainable weight matrices. $\boldsymbol{X}_s[e,:] = \boldsymbol{x}_{s,e}$. $\{\boldsymbol{S}_i\}$ are normalized adjacency matrices introduced in Section 4.2. With Proposition 4.2 we can immediately conclude that the solution $\boldsymbol{Z}_s$ exists and is unique:

Corollary 4.4. *The solution $\boldsymbol{Z}_s$ of Eq. 10 exists and is unique.*

4.4. Forward and backward pass

We now introduce the forward pass (i.e. compute the fixed points) and the backward pass (i.e. evaluate gradients) of the model. Since LIGNN is an instance of graph convolutional networks, the evaluation of $\boldsymbol{y}_{s,t}$ with different sources are separate and can be performed individually. To compute $\boldsymbol{y}_{s,t}$ for any (s,t), we first initialize $\boldsymbol{Z}_s = \boldsymbol{O}_{N\times m}$ to be a zero matrix and then apply Eq. 10 on $\boldsymbol{Z}_s$ until convergence. Thus, the additional space occupation is $\mathcal{O}(N+M)$, and the amortized computational time for each link property is $\mathcal{O}(M/N)$, where we omit terms irrelevant to the graph size. We leave a more detailed discussion of complexities in Appendix I.

For backward pass we need to compute gradients w.r.t. any parameters. Let $\mathcal{L}$ be some loss function. Often, $\mathcal{L}$ is a summation over distinct link labels,

$$\mathcal{L}(\boldsymbol{y}_{1,1}, ..., \boldsymbol{y}_{s,t}, ..., \boldsymbol{y}_{N,N}) = \sum_{s,t\in\mathcal{V}} \mathcal{L}_{st}(\boldsymbol{y}_{s,t}).$$

Then, the gradients can also be evaluated separately:

$$\frac{\partial \mathcal{L}}{\partial(\cdot)} = \sum_{s,t\in\mathcal{V}} \frac{\partial \mathcal{L}_{st}(\boldsymbol{y}_{s,t})}{\partial(\cdot)} = \sum_{s\in\mathcal{V}} \frac{\partial \mathcal{L}_s(\boldsymbol{Y}_s)}{\partial(\cdot)}.$$

where we can sum over all distinct batches of evaluations to obtain the global gradients. To compute the last term, we apply implicit differentiation (Bai et al., 2019):

$$\frac{\partial \mathcal{L}_s}{\partial(\cdot)} = \frac{\partial \mathcal{L}_s}{\partial \boldsymbol{Z}_s} \left(\boldsymbol{I} - \frac{\partial f(\boldsymbol{Z}_s^*, \boldsymbol{X}_s, \mathbf{A})}{\partial \boldsymbol{Z}_s^*} \right)^{-1} \frac{\partial f(\boldsymbol{Z}_s^*, \boldsymbol{X}_s, \mathbf{A})}{\partial(\cdot)},$$

where $\boldsymbol{Z}_s^*$ is the value of the equilibrium point of Eq. 10, $f(\boldsymbol{Z}_s^*, \boldsymbol{X}_s, \mathbf{A}) = \gamma \sum_{i=1}^{D} \boldsymbol{S}_i \boldsymbol{Z}_s^* \boldsymbol{\Theta}_i + \boldsymbol{X}_s \boldsymbol{W}$. We avoid the computation of the matrix inversion by applying another iterative process:

$$g = \left(\frac{\partial f(\boldsymbol{Z}_s^*, \boldsymbol{X}_s, \mathbf{A})}{\partial \boldsymbol{Z}_s^*} \right)^T g + \frac{\partial \mathcal{L}_s}{\partial \boldsymbol{Z}_s},$$

and with the solution g we obtain a Vector-Jacobian product estimation of the original gradients. Therefore, the backward pass takes the same time and space complexities with the forward pass, and the gradients can be evaluated using automatic differentiation. We provide the software implementation to illustrate the ideas in Appendix G.

5. Analysis

5.1. Expressiveness and invariants

With the utilization of graph convolutions, LIGNN (as well as the model introduced in Section 3) does not preserve the full discriminative power of 2-FWL tests due to the non-injective mean pooling aggregation function. We show that this loss of expressiveness actually leads to some benefits. We first discuss some invariant properties of LIGNN that benefit link prediction tasks.

Proposition 5.1 (Invariants)**.** *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, LIGNN maps all disconnected nodes $s, t \in \mathcal{V}$ with equal link representations $\boldsymbol{z}_{s,t} = \mathbf{0}$. Given another arbitrary graph $\mathcal{G}_1$, we can put $\mathcal{G}$ and $\mathcal{G}_1$ together to obtain $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$. Let $s, t \in \mathcal{V}$ and $s', t' \in \mathcal{V}'$ be the corresponding vertices of s, t in $\mathcal{G}'$, respectively. Then, LIGNN always maps (s, t) and (s', t') to the same link representation.*

Note that 2-FWL test does not preserve the above property. As a result, given a set of connected graphs $\mathcal{G}_1, \mathcal{G}_2, ...$, we can always view these graphs as one huge graph $\mathcal{G}$ (thus each subgraph is a connected component of $\mathcal{G}$), and on both settings, LIGNN produces exactly the same results. However, 2-FWL test can obtain different results when we view the graphs differently. In real world, there are many situations, such as molecular graphs, where the inputs are composed of many subgraphs. In this case, we wish the model to preserve the invariant properties (although slightly losing discriminative power) to make consistent predictions.

On the other hand, LIGNN preserves the 2-FWL discriminative power to some extent:

Proposition 5.2. *There are graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1), \mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ and $s, t \in \mathcal{V}_1, p, q \in \mathcal{V}_2$, such that LIGNN maps (s, t) and (p, q) to different representations, while the 1-WL test maps (s, t) and (p, q) to be equivalent.*

5.2. Relation with labeling trick

We compare LIGNN with labeling tricks, a technique that enhances node level GNNs for link prediction, and show that LIGNN can also be regarded as an extension of labeling tricks.

We first introduce the concept of labeling trick based on SEAL (Zhang & Chen, 2018; Li et al., 2020). Suppose we are to predict the link (s, t). We first assign additional labels to each node of the graph to mark s, t differently from the rest of the nodes. For SEAL, s and t are both labeled 1, and other nodes are labeled to be the sum of distances from s and t. Then, we apply a node level GNN on the augmented graph as an auto encoder and obtain the link representation of (s, t) as $f(\boldsymbol{z}_s, \boldsymbol{z}_t)$ where $\boldsymbol{z}_s, \boldsymbol{z}_t$ are the node representation of s, t respectively and f is some aggregation function. This technique enhances the link prediction power of node level GNNs.

There are some similarities between LIGNN and labeling trick, as Eq. 10 can also be seen as a node level GNN. The difference is that, we set node features to be $\boldsymbol{X}_s$ for predicting every link with the same source $(s, ?)$. This implies that LIGNN actually *only encodes the 1-hop neighbors of the source node*, which is more efficient in that (1) one-time encoding can be used for predicting N different links with the same head, (2) the overhead is low, because 1-hop neighbors of source node often comprise only a small portion of the graph, making simultaneously computing multi-source links possible, and (3) naturally implies the idea of subgraph extraction, because at layer l the model only incorporate the nodes at most l-hop away from s.

Besides, when forward propagation is completed, LIGCN does not require a decoding procedure to evaluate the link representations. Instead, we directly pick the node representation at t to be the link representation (s, t), where s is the source node.

5.3. Approximate inference

As discussed previously, LIGNN naturally implies subgraph extraction. Suppose we are to compute the link property of (s, t) in a sparse graph $\mathcal{G}$. By setting a small γ and cutting down the maximum iteration number to a feasible value T, we can approximate the inference of $\boldsymbol{y}_{s,t}$ with relative small errors (Liu et al., 2022a) and much less cost.

Due to the sparsity of $\boldsymbol{X}_s$, we can see that at iteration 1, only 1-hop neighbors of s are assigned with nonzero hidden representations. By induction it's straightforward to conclude that at iteration l only nodes at more l-hop away from s are nonzero. As a result, the evaluation of $\boldsymbol{y}_{s,t}$ only incorporates a subgraph $\mathcal{G}_{s,t}$ of $\mathcal{G}$, that is, for each node e in $\mathcal{G}_{s,t}$, we have

$$\mathrm{d}(s, e) + \mathrm{d}(e, t) \leq T,$$

where $\mathrm{d}(i, j)$ is the shortest distance from i to j. Therefore, to evaluate $\boldsymbol{y}_{s,t}$ in $\mathcal{G}$, we can instead run the iterations on the subgraph $\mathcal{G}_{s,t}$ for simplification.

6. Related Works

Implicit models Our work is an instance of implicit models. Generally, neural network models heavily relies on the concept of layers. Given the input data, these models compute outputs by propagating the inputs sequentially through their layers. Different from these methods, the outputs in implicit models are implicitly defined, often as a fixed-point solution of some equations. An advantage of the implicit models are that they can achieve constant space complexity regardless of the depth of the model. Bai et al. (2019) propose the deep equilibrium model (DEQ) and show how

they design a neural network with implicitly defined layers. Many recent works (Bai et al., 2020; Chen et al., 2018b; Bai et al., 2022) have shown the potential of implicit models on image classification, sequence modeling, etc. Besides the applications of implicit models, Geng et al. (2021) propose phantom gradients to accelerate the training of implicit models. Kawaguchi (2021) study the convergency of implicit models.

Graph neural networks GNNs are widely applied to graph-related machine learning tasks. Bruna et al. (2013) first extend the concept of convolutional networks to graphs. Defferrard et al. (2016) remove the expensive Laplacian eigendecomposition by using Chebyshev polynomials. After that, Kipf & Welling (2016a) propose graph convolutional networks (GCNs), which further simplify graph convolutions with a redefined message-passing scheme. At each layer GCN propagates node representations through a normalized adjacency matrix S. Based on GCNs, Chen et al. (2018a) propose a more efficient GCN variant based on sampling. Wu et al. (2019) remove all the non-linear parts of GCNs and obtain a extremely simple variant, namely SGC, which surprisingly show strong performance compared with non-linear GCNs. This also inspires our work, as it shows the strength of simple linear models in graph machine learning fields.

Besides GCNs, there are also many other GNN variants based on a more common message passing mechanism. To name a few, Hamilton et al. (2017) propose GraphSAGE, a inductive graph representation learning framework. Xu et al. (2018) studies the discriminative power of GNNs, and propose GIN which shares the same expressiveness with 1-WL test.

Higher-order GNNs Most GNNs focus on graph classification and node classification tasks, and they learn representations for each nodes (i.e., they are node level GNNs), thus their discriminative power are often restricted to 1-WL tests. Noticing the similarity between GNN message-passing mechanism and WL tests, Morris et al. (2018) propose a GNN model to parameterize the 2-WL and 3-WL tests. Similarly, Maron et al. (2019) propose to parameterize the 2-FWL test with GNNs. These GNNs learn high-order representations, i.e. they learn representations for node tuples. This makes higher-order GNNs inherently suitable for link prediction tasks. However, previous high-order GNNs are still rather complicated and inefficient.

Combining GNNs with implicit models Recently, there is a series of works trying to combine GNNs with implicit models. By doing so, the implicit GNNs obtain a new ability: since the fixed point of implicit models can be evaluated via an infinite sequence, implicit GNNs are now able to capture very long-range dependencies[1]. Gu et al. (2020) propose IGNN, an implicit graph neural network model. Yang et al. (2021) design GNN model to mimic the update rule of classical graph iterative algorithms. Liu et al. (2022b) utilize matrix decomposition to compute the fixed points without iterative solvers. Liu et al. (2022a) designs multi-scale connections to enhance the IGNN model. Park et al. (2021) propose CGS to solve graph value iteration problems. The previous efforts focus on node level GNNs for graph classification and node classification tasks, and higher-order GNNs for link prediction are not considered.

Link prediction Link prediction is a fundamental problem on graphs. Existing link prediction method can be generally classified into 3 types: rule-based, knowledge graph embedding and graph neural networks. Representative rule-based includes earlier works FOIL (Quinlan, 2004), MDIE (Muggleton, 2009), AMIE (Galárraga et al., 2015), etc. and recent neural-enhanced rule learning methods such as RLvLR (Omran et al., 2018), NeuralLP (Yang et al., 2017), DRUM (Rocktäschel & Riedel, 2017) and RNN-Logic (Qu et al., 2021). These methods generally learn probabilistic logic rules to predict the existence of the target links, and usually is applied to heterogeneous graphs such as knowledge graphs.

Knowledge graph embedding methods learn a distribution representation for each node and each type of edge in the graph by minimizing the distance between connected nodes. Representative methods include TransE (Bordes et al., 2013), DistMult (Yang et al., 2015), RotatE (Sun et al., 2018) and many other works that propose new scoring functions (Trouillon et al., 2016; Dettmers et al., 2017; Kazemi & Poole, 2018). These methods explicitly assigns embeddings for nodes, making them not applicable to the inductive setting.

Graph neural networks learn representations for each node. Early methods such as GAE (Kipf & Welling, 2016b) use GNN as an encoder and decode link representations as a function over node representation pairs. These methods are problematic in capturing complex graph structures, and might lead to poor performance. Later on, labeling trick was introduced by SEAL (Zhang & Chen, 2018) and adopted by GraIL (Teru et al., 2020), IGMC (Zhang & Chen, 2020), INDIGO (Liu et al., 2021), etc. These methods encode source and target nodes to mark them differently from the rest of the graph, and are proved to be more powerful than GAE. ID-GNN (You et al., 2021) and NBFNet (Zhu et al., 2021) shares some spirits with our approach, as they both augments GNNs with the identity of the source nodes, but our model is motivated by the 2-FWL framework that directly

[1] In fact, implicit GNNs are able to capture infinite-range dependencies in theory, but due to floating point errors they can only propagate messages to finite depths.

assigns node pairs as message passing units and has theoretical connections with higher-order GNNs, which can also be used for link prediction and are introduced in previous paragraphs. Besides, All-path (Toutanova et al., 2016) encodes relations as linear projections and proposes to efficiently aggregate all paths with dynamic programming. However, All-Path is restricted to bilinear models, has limited link prediction capability and is also not inductive. EdgeTransformer (Bergen et al., 2021) utilizes attention mechanism to learn representations for nodes and links. While it also follows the 2-FWL message passing procedure, it operates directly on fully-connected graphs and have no proposals for simplifications as we do, thus it is not scalable to larger graphs.

7. Experiments

We evaluate LIGNN on synthetic and real-word link prediction tasks to test the model's ability to capture long-range and recursive dependencies. Specifically, we conduct experiments on several synthetic relational reasoning tasks which require the model to learn recursive dependencies. We also conduct experiments on real-world molecular property prediction datasets recently proposed via Long Range Graph Benchmark (Dwivedi et al., 2022), and knowledge graphs including WN18RR (Dettmers et al., 2017), FB15k-237 (Toutanova & Chen, 2015) and NELL-995 (Xiong et al., 2017). Detailed model configurations are in Appendix E.

7.1. Evaluation on synthetic data

We first evaluate our model on synthetic data sets to test the ability of capturing long-range and recursive dependencies. The solutions (as well as the input graphs) can be expressed by a set of first-order logic formulas, for example consider the graph connectivity problem, whose solution is:

$$\text{Edge}(x, y) \Rightarrow \text{Connect}(x, y)$$
$$\text{Connect}(x, y), \text{Connect}(y, z) \Rightarrow \text{Connect}(x, z).$$

The problem is complex because we need to apply the formulas recursively to obtain the final solutions. Therefore, the model needs to decouple the implicit reasoning iterations to learn the latent mechanism for generating the target relation Connect. We provide several problem settings for testing the model's ability for learning complex relational reasoning. The tasks are designed as follows.

Graph connectivity The goal is to determine whether two nodes are connected in a directed chain graph. Training graph is a chain of length 10, and test graphs are of lengths $\{10, 20, ..., 200\}$. The model is asked to predict the connectivity of all node pairs.

Edge counting The goal is to determine whether two nodes in a directed chain are connected via an even number of edges. Training graph is a chain of length 10, and test graphs are generally longer, with lengths $\{10, 20, ..., 200\}$. The model is asked to predict all node pairs.

These tasks require the model to effectively captures very long-range dependencies. The solutions can be obtained with the fixed point of corresponding Datalog programs (Ceri et al., 1989), estimated by the iterative computation of the naïve evaluation. Since previous link-level GNNs are limited to small numbers of layers, we choose other implicit GNN models with infinite depths including EIGNN (Liu et al., 2022b) and CGS (Park et al., 2021). We exclude MGNNI (Liu et al., 2022a) due to the absence of official codes. We train these models as graph auto encoders with a labeling trick that marks sources 1 and tails 2. Then we obtain link representations as the inner production of source and target nodes. The results are shown in Figure 1, 2.

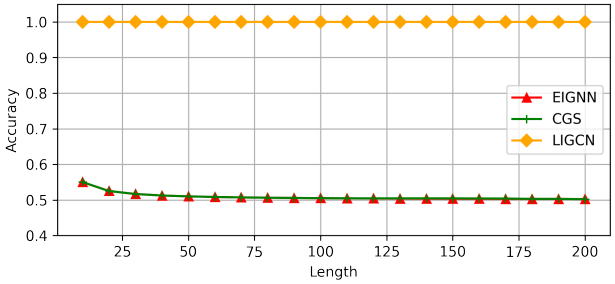

Figure 1. Accuracy of graph connectiveness task.

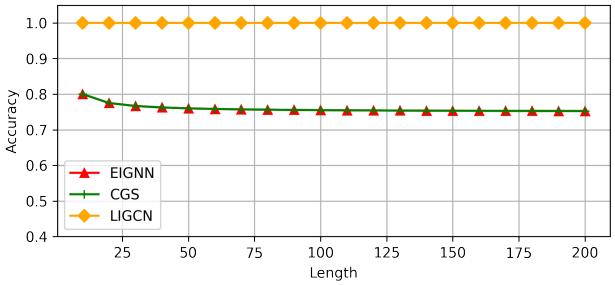

Figure 2. Accuracy of edge counting task.

Analysis From the results we can see that all three implicit models are able to capture recursive dependencies to some extent. As the problem is naturally expressed as a link prediction problem, LIGNN achieves the best performance with correctly classifying all target links.

7.2. Evaluation on long-range molecular graphs

Long range graph benchmark (LRGB) is recently proposed by (Dwivedi et al., 2022). The LRGB datasets includes four distinct graph benchmarks including graph classification, node classification and link classification tasks. Here, we consider the PCQM-Contact dataset, a molecular property prediction task, where the target is to predict whether pairs of distant nodes (more than 5 hops away from each other in a molecule graph) will be connecting with each other. Training and test data contains plenty of instances, and each data instance includes:

- A background molecular graph;
- Node and edge attribute vectors;
- A set of positive and negative target links.

To train our model on it, we first pool the node attributes together with edge attributes to obtain initial link representations. That is, suppose nodes s, t are connected where $\boldsymbol{v}_s, \boldsymbol{v}_t$ are node attributes of s, t and $\boldsymbol{e}_{s,t}$ is the edge attribute of the edge (s, t), respectively. Then, we have $\boldsymbol{x}_{s,t} = [\boldsymbol{v}_s^T, \boldsymbol{v}_t^T, \boldsymbol{e}_{s,t}^T]^T$. If s, t are not connected, we simply let $\boldsymbol{x}_{s,t} = \mathbf{0}$. We run our model on each molecular graph, and train it by maximizing the likelihood on the set of positive and negative target links.

The performance metrics are the same as in (Dwivedi et al., 2022), where we use the filtered ranking metrics (Hoyt et al., 2022). Given a query $(s, ?)$, we compute a score for all other nodes e as a tail (s, e). Then, we compute the rank of the true positive links. Based on the ranks, we compute standard ranking metrics Hits@1, Hits@3 and Mean Reciprocal Rank (MRR). The results are in Table 1.

The baselines and their experimental results are taken from (Dwivedi et al., 2022), including GCN (Kipf & Welling, 2016a), GCNII (Chen et al., 2020), GIN (Xu et al., 2018) and GGCN (Bresson & Laurent, 2017) from local message passing GNN class, and fully connected Transformer (Vaswani et al., 2017) with Laplacian PE (Dwivedi & Bresson, 2020) and SAN (Kreuzer et al., 2021). Previous link level GNNs are not suitable here because both node features and edge features are real-valued vectors.

Analysis We can see from Table 1 that Transformer based fully connected models generally outperform message passing GNN models. This is because that the target nodes are usually distinct from each other in the graphs, and GNNs with fixed layers are limited to only a few hops of information aggregation. Therefore, Transformer based models are able to capture global information of the graph and thus achieve better performances. Compared with these models, LIGNN shows the best results, indicating the importance of learning link representations that captures long-range dependencies. Moreover, LIGNN achieves the performance with much less parameters: only about 2.5K parameters are used in the model, which is less than one percent of the other baselines.

Table 1. Results on PCQM-Contact

MODEL	#PARAMS.	HITS@1	HITS@3	MRR
GCN	504K	13.21	37.91	0.3234
GCNII	501K	13.25	36.07	0.3161
GINE	517K	13.37	36.42	0.3180
GGCN	527K	12.79	37.83	0.3218
GGCN+RWSE	524K	12.88	38.08	0.3242
TF+LAPPE	502K	12.21	36.79	0.3174
SAN+LAPPE	499K	13.55	40.04	0.3350
SAN+RWSE	509K	13.12	40.30	0.3341
LIGNN	2.5K	**31.07**	**59.78**	**0.4835**

7.3. Evaluation on knowledge graphs

Following (Liu et al., 2021), we conduct experiments on several real-world knowledge graphs to test the model's ability to perform general link prediction tasks.

We assess the link prediction performance of LIGNN on 12 KG completion benchmarks: 4 benchmarks for each KG including WN18RR, FB15k-237, and NELL-995. We call them using the pattern *XXX*.vi, where *XXX* is the base and i the number of the version. Each of the benchmark contains:

- a KG $\mathcal{G}_{\text{train}}$ for training;
- an incomplete KG $\mathcal{G}_{\text{test}}$ and a set of test triples $\mathcal{T}_{\text{test}}$.

The target is, by training the model on $\mathcal{G}_{\text{train}}$ and then applying it on $\mathcal{G}_{\text{test}}$, to predict the existence of positive edges in $\mathcal{T}_{\text{test}}$ as well as rejecting negative ones. Note that $\mathcal{G}_{\text{train}}$ and $\mathcal{G}_{\text{test}}$ has no overlapped nodes.

We train our model as follows. Each edge in $\mathcal{G}_{\text{train}}$ is as a positive example, and we generate negative examples by randomly corrupting tail nodes. We run our model on $\mathcal{G}_{\text{train}}$ and obtain link representations, and train the model by maximizing the likelihood on positive and negative examples.

The performance metrics are standard classification metrics and ranking metrics. For standard classification metrics, we consider accuracy and AUC. For ranking metrics, we follow the settings in (Liu et al., 2021) to compute Hits@3 both by randomly corrupting heads / tails (e-Hits@3) and relations (r-Hits@3) of triples. A difference is that for each positive triple we only sample 50 negative triple. The results are in Table 2 and Table 3.

The baselines and their experimental results are taken from

Table 2. Link prediction on KG completion, part 1. R: R-GCN, G: GraIL, I: INDIGO, L: LIGNN.

Data		Accuracy				AUC			
		R	G	I	L	R	G	I	L
FB15K-237	v1	51.0	69.0	**84.3**	77.8	51.0	78.6	**93.4**	81.9
	v2	51.3	80.0	**89.3**	82.0	50.5	90.0	**96.3**	92.1
	v3	54.9	81.0	**89.0**	85.5	50.5	93.1	**96.6**	91.9
	v4	52.1	79.3	**87.8**	75.2	52.6	89.5	**95.8**	89.6
NELL-995	v1	63.7	**97.3**	85.6	90.5	75.0	98.8	94.5	**99.7**
	v2	52.0	68.5	**84.1**	80.0	50.4	89.7	**92.5**	90.1
	v3	52.3	74.3	**89.7**	85.6	52.0	95.4	**95.1**	91.1
	v4	53.6	49.7	**85.2**	82.2	51.0	65.8	**92.9**	89.2
WN18RR	v1	50.2	88.7	85.7	**88.8**	49.0	92.3	91.2	**97.5**
	v2	52.7	81.2	**85.8**	85.4	49.8	92.7	92.5	**95.8**
	v3	52.2	75.7	**84.3**	80.6	53.1	82.8	**92.4**	76.2
	v4	48.4	86.4	85.4	**88.5**	50.2	94.4	**94.7**	87.3

(Liu et al., 2021), including R-GCN (Schlichtkrull et al., 2017), GraIL (Teru et al., 2019) and INDIGO (Liu et al., 2021).

Analysis KG completion tasks are with less demands for capturing long-range dependencies, as using logical rules with length 3 (Qu et al., 2021) already produces competitive performance. Therefore, KG completion tests the model's general link prediction ability. From Table 2 3, LIGNN outperforms other baselines in predicting entities (e-Hits). This makes sense because the forward pass is naturally aligned with the e-Hits evaluation metric. Similarly, LIGNN also show good performance in predicting edge types r-Hits. LIGNN is less competitive on Accuracy and AUC metrics. A reason is that these two metrics compares links with totally different heads, relations and tails. Since the computation of all link representations in a graph by LIGNN includes N distinct rounds of evaluations, these metrics requires comparison between links representations that are computed from distinct runs. Even so, LIGNN is able to achieve consistently competitive performances on all metrics. R-GCN and GraIL are both poor at predicting edge types (r-Hits), and INDIGO is poor at predicting missing entities (e-Hits).

8. Discussion and Conclusion

Limitations. There are several limitations for LIGNN. First, although we aim to design GNNs that directly learns link representations to enhance the link discriminative power, LIGNN is not as expressive as 2-FWL test. Second, for sufficiently large and dense graphs, LIGNN cannot keep a relatively small memory consumption while capturing long-range dependencies. We might need to cut down the maximum depth of LIGNN to a relative small value.

Table 3. Link prediction on KG completion, part 2.

Data		E-Hits@3				R-Hits@3			
		R	G	I	L	R	G	I	L
FB15K-237	v1	16.1	43.4	45.1	**60.6**	2.4	1.0	53.1	**57.4**
	v2	18.3	68.5	36.2	**77.1**	3.4	0.4	67.6	**76.5**
	v3	14.1	71.2	33.9	**75.2**	3.5	6.6	66.5	**73.6**
	v4	16.1	61.8	37.1	**75.9**	3.3	3.0	66.3	**68.3**
NELL-995	v1	7.5	51.0	39.5	**54.3**	26.0	0.0	80.0	**95.0**
	v2	12.2	76.5	44.2	**85.4**	0.8	7.4	56.9	**64.6**
	v3	15.5	84.4	45.0	**87.7**	1.4	2.5	64.4	**73.4**
	v4	9.3	56.0	52.3	**83.9**	3.0	0.5	45.7	**62.5**
WN18RR	v1	20.1	82.7	12.5	**86.7**	2.1	0.6	98.4	**99.5**
	v2	18.1	81.5	18.8	**83.5**	11.0	10.7	97.3	**97.5**
	v3	16.9	55.5	33.1	**62.2**	24.5	17.5	**91.9**	63.8
	v4	8.8	76.3	13.5	**78.8**	8.1	22.6	**96.1**	82.5

Social impacts. This work aims to provide fair link prediction results even under OOD situation. However, it is also possible that LIGNN may encode bias in the training data, leading to prejudiced predictions. Also, some harmful activities such as phishing, social engineering might be augmented by link prediction models. We hope future studies will alleviate these negative impacts.

Conclusion. We propose LIGNN, a link-level deep equilibrium model for predicting link properties in graphs. We parameterize the general message passing mechanism to obtain a trainable layer, and further linearize the model to obtain a more efficient variant without loss of discriminative power. Due to the computation of stationary points, LIGNN can learn link representations that capture long-range and recursive dependencies. We demonstrate this ability by evaluating the model on various synthetic experiments. Besides, LIGNN shows great performances on real-world datasets with long-range dependencies. On general real-world datasets, LIGNN also shows competitive performances.

References

Abiteboul, S., Hull, R., and Vianu, V. Foundations of databases. 1994.

Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models. *ArXiv*, abs/1909.01377, 2019.

Bai, S., Koltun, V., and Kolter, J. Z. Multiscale deep equilibrium models. *ArXiv*, abs/2006.08656, 2020.

Bai, S., Geng, Z., Savani, Y., and Kolter, J. Z. Deep equilibrium optical flow estimation. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 610–620, 2022.

Bergen, L., O'Donnell, T. J., and Bahdanau., D. Systematic generalization with edge transformers. In *NeurIPS*, 2021.

Bordes, A., Usunier, N., García-Durán, A., Weston, J., and Yakhnenko, O. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.

Bresson, X. and Laurent, T. Residual gated graph convnets. *ArXiv*, abs/1711.07553, 2017.

Bruna, J., Zaremba, W., Szlam, A. D., and LeCun, Y. Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203, 2013.

Cai, J.-Y., Fürer, M., and Immerman, N. An optimal lower bound on the number of variables for graph identification. *Combinatorica*, 12:389–410, 1989.

Ceri, S., Gottlob, G., and Tanca, L. What you always wanted to know about datalog (and never dared to ask). *IEEE Trans. Knowl. Data Eng.*, 1:146–166, 1989.

Chen, J., Ma, T., and Xiao, C. Fastgcn: Fast learning with graph convolutional networks via importance sampling. *ArXiv*, abs/1801.10247, 2018a.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. *ArXiv*, abs/2007.02133, 2020.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Neural Information Processing Systems*, 2018b.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 2016.

Dettmers, T., Minervini, P., Stenetorp, P., and Riedel, S. Convolutional 2d knowledge graph embeddings. In *AAAI Conference on Artificial Intelligence*, 2017.

Dwivedi, V. P. and Bresson, X. A generalization of transformer networks to graphs. *ArXiv*, abs/2012.09699, 2020.

Dwivedi, V. P., Rampasek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. Long range graph benchmark. *ArXiv*, abs/2206.08164, 2022.

Galárraga, L., Teflioudi, C., Hose, K., and Suchanek, F. M. Fast rule mining in ontological knowledge bases with amie+. *The VLDB Journal*, 24:707–730, 2015.

Geng, Z., Zhang, X., Bai, S., Wang, Y., and Lin, Z. On training implicit models. In *Neural Information Processing Systems*, 2021.

González, J. C. M., Arriaga, J., and Fortin, S. Quantum mechanics and molecular structure: The case of optical isomers. *Quantum Worlds*, 2019.

Grohe, M. Descriptive complexity, canonisation, and definable graph structure theory. In *Lecture Notes in Logic*, 2017.

Gu, F., Chang, H., Zhu, W., Sojoudi, S., and Ghaoui, L. E. Implicit graph neural networks. *ArXiv*, abs/2009.06211, 2020.

Hamilton, W. L., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *NIPS*, 2017.

Hoyt, C. T., Berrendorf, M., Galkin, M., Tresp, V., and Gyori, B. M. A unified framework for rank-based evaluation metrics for link prediction in knowledge graphs. *ArXiv*, abs/2203.07544, 2022.

Ioannidis, V. N., Zheng, D., and Karypis, G. Few-shot link prediction via graph neural networks for covid-19 drug-repurposing. *ArXiv*, abs/2007.10261, 2020.

Kawaguchi, K. On the theory of implicit deep learning: Global convergence with implicit layers. *ArXiv*, abs/2102.07346, 2021.

Kazemi, S. M. and Poole, D. Simple embedding for link prediction in knowledge graphs. In *NeurIPS*, 2018.

Kipf, T. and Welling, M. Semi-supervised classification with graph convolutional networks. *ArXiv*, abs/1609.02907, 2016a.

Kipf, T. and Welling, M. Variational graph auto-encoders. *ArXiv*, abs/1611.07308, 2016b.

Koren, Y., Bell, R. M., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer*, 42, 2009.

Kreuzer, D., Beaini, D., Hamilton, W. L., L'etourneau, V., and Tossou, P. Rethinking graph transformers with spectral attention. *ArXiv*, abs/2106.03893, 2021.

Li, P., Wang, Y., Wang, H., and Leskovec, J. Distance encoding - design provably more powerful gnns for structural representation learning. 2020.

Liu, J., Hooi, B., Kawaguchi, K., and Xiao, X. Mgnni: Multiscale graph neural networks with implicit layers. *ArXiv*, abs/2210.08353, 2022a.

Liu, J., Kawaguchi, K., Hooi, B., Wang, Y., and Xiao, X. Eignn: Efficient infinite-depth graph neural networks. *ArXiv*, abs/2202.10720, 2022b.

Liu, S., Grau, B. C., Horrocks, I., and Kostylev, E. V. Indigo: Gnn-based inductive knowledge graph completion using pair-wise encoding. In *Neural Information Processing Systems*, 2021.

Maron, H., Ben-Hamu, H., Serviansky, H., and Lipman, Y. Provably powerful graph networks. *ArXiv*, abs/1905.11136, 2019.

Morris, C., Ritzert, M., Fey, M., Hamilton, W. L., Lenssen, J. E., Rattan, G., and Grohe, M. Weisfeiler and leman go neural: Higher-order graph neural networks. *ArXiv*, abs/1810.02244, 2018.

Muggleton, S. Inverse entailment and progol. *New Generation Computing*, 13:245–286, 2009.

Nguyen, T., Phung, D. Q., Adams, B., and Venkatesh, S. Towards discovery of influence and personality traits through social link prediction. *Proceedings of the International AAAI Conference on Web and Social Media*, 2021.

Nickel, M., Murphy, K. P., Tresp, V., and Gabrilovich, E. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104:11–33, 2015.

Omran, P. G., Wang, K., and Wang, Z. Scalable rule learning via learning representation. In *IJCAI*, 2018.

Park, J., Choo, J., and Park, J. Convergent graph solvers. *ArXiv*, abs/2106.01680, 2021.

Qu, M., Chen, J., Xhonneux, L.-P., Bengio, Y., and Tang, J. Rnnlogic: Learning logic rules for reasoning on knowledge graphs. In *ICLR*, 2021.

Quinlan, J. R. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 2004.

Rocktäschel, T. and Riedel, S. End-to-end differentiable proving. In *NeurIPS*, pp. 3791–3803, 2017. URL http://papers.nips.cc/paper/6969-end-to-end-differentiable-proving.

Schlichtkrull, M., Kipf, T., Bloem, P., van den Berg, R., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In *Extended Semantic Web Conference*, 2017.

Sun, Z., Deng, Z., Nie, J.-Y., and Tang, J. Rotate: Knowledge graph embedding by relational rotation in complex space. *ArXiv*, abs/1902.10197, 2018.

Teru, K. K., Denis, E., and Hamilton, W. L. Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*, 2019.

Teru, K. K., Denis, E., and Hamilton, W. L. Inductive relation prediction by subgraph reasoning. In *ICML*, 2020.

Toutanova, K. and Chen, D. Observed versus latent features for knowledge base and text inference. In *Workshop on Continuous Vector Space Models and their Compositionality*, 2015.

Toutanova, K., Lin, X. V., tau Yih, W., Poon, H., and Quirk., C. Compositional learning of embeddings for relation paths in knowledge base and text. In *ACL*, 2016.

Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., and Bouchard, G. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, 2016.

Vaswani, A., Shazeer, N. M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

Wu, F., Zhang, T., de Souza, A. H., Fifty, C., Yu, T., and Weinberger, K. Q. Simplifying graph convolutional networks. *ArXiv*, abs/1902.07153, 2019.

Xiong, W., Hoang, T.-L.-G., and Wang, W. Y. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *Conference on Empirical Methods in Natural Language Processing*, 2017.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? *ArXiv*, abs/1810.00826, 2018.

Yang, B., tau Yih, W., He, X., Gao, J., and Deng, L. Embedding entities and relations for learning and inference in knowledge bases. *CoRR*, abs/1412.6575, 2014.

Yang, B., tau Yih, W., He, X., Gao, J., and Deng, L. Embedding entities and relations for learning and inference in knowledge bases. *CoRR*, abs/1412.6575, 2015.

Yang, F., Yang, Z., and Cohen, W. W. Differentiable learning of logical rules for knowledge base completion. In *NeurIPS*, 2017.

Yang, Y., Liu, T., Wang, Y., Zhou, J., Gan, Q., Wei, Z., Zhang, Z., Huang, Z., and Wipf, D. P. Graph neural networks inspired by classical iterative algorithms. In *ICML*, 2021.

You, J., Gomes-Selman, J. M., Ying, R., and Leskovec., J. Identity-aware graph neural networks. In *AAAI*, 2021.

Zhang, M. and Chen, Y. Link prediction based on graph neural networks. *ArXiv*, abs/1802.09691, 2018.

Zhang, M. and Chen, Y. Inductive matrix completion based on graph neural networks. In *ICLR*, 2020.

Zhu, Z., Zhang, Z., Xhonneux, L.-P., and Tang, J. Neural bellman-ford networks: A general graph neural network framework for link prediction. In *Neural Information Processing Systems*, 2021.

A. Proof of Theorem 3.2

In this section we prove Theorem 3.2. We state it as follows:

Theorem A.1. *Consider the two implementations:*

$$\boldsymbol{z}_{s,t}^{(l+1)} = \boldsymbol{W}_1 \sum_{e\in\mathcal{V}} (\boldsymbol{W}_3 \boldsymbol{z}_{s,e}^{(l)}) \odot (\boldsymbol{W}_4 \boldsymbol{z}_{e,t}^{(l)}) + \boldsymbol{W}_2 \boldsymbol{z}_{s,t}^{(l)}, \tag{11}$$

$$\boldsymbol{z}_{s,t}^{\prime(l+1)} = \boldsymbol{W}_1' \sum_{e\in\mathcal{V}} (\boldsymbol{W}_3' \boldsymbol{z}_{s,e}^{(l)}) \odot (\boldsymbol{W}_4' \boldsymbol{x}_{e,t}) + \boldsymbol{W}_2' \boldsymbol{z}_{s,t}^{\prime(l)}. \tag{12}$$

Then, these variants share the same discriminative power. Given graphs $\mathcal{G}_1, \mathcal{G}_2$ and $s,t \in \mathcal{V}_1, p,q \in \mathcal{V}_2$, if at some iteration Eq. 11 maps (s,t) and (p,q) to different embeddings, then there exists W_1', W_2', W_3', W_4' such that at some iteration l, Eq. 12 also maps them to different embeddings, i.e.

$$\boldsymbol{z}_{s,t}^{\prime(l)} \neq \boldsymbol{z}_{p,q}^{\prime(l)}.$$

Vice versa.

We now show that the two parameterization Eq. 11 and Eq. 12 share equivalent discriminative power. To do so, we establish a correspondence between them and a variant of horn clauses stated as follows, which we call horn clauses with counting ($\mathcal{HCWC}$).

First, let's consider an instantiation of one $\mathcal{HCWC}$ rule:

$$\varphi \leftarrow B_1, B_2, ..., B_T : \alpha, \tag{13}$$

where φ is the head of the rule, $\alpha \in \mathbb{R}$ is the weight of the rule, and $B_1, B_2, ..., B_T$ is the rule body of length T. The semantics of the above rule is as follows. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, by applying the rule on $\mathcal{G}$, it is interpreted as follows:

$$\varphi(s,t|\mathcal{G}) := \alpha \sum_{s,t,e_1,e_2,...\in\mathcal{V}} B_1(s,e_1|\mathcal{G})\ B_2(e_1,e_2|\mathcal{G})\ ...\ B_T(e_{T-1},t|\mathcal{G}),$$

where $\varphi(\cdot,\cdot|\cdot), B_1(\cdot,\cdot|\cdot), ... \in \mathbb{R}$ are relaxations of logic predicates, and $:=$ is the assignment symbol. Thus, $\varphi(s,t|\mathcal{G})$ is able to count how many true groundings are there that satisfy the body of the rule. We call such φ that is indirectly defined by rules an intensional predicate. In contract, if some φ is directly given in the graph (e.g., edges of the graph $E(s,t|\mathcal{G})$), then it is called an extensional predicate. Note that extensional predicates cannot be rules heads.

Now, let's extend the above definitions to more complicated situations. Let Φ be the set of intensional predicates. Let $\mathcal{P}$ be the set of extensional predicates. Consider a set of $\mathcal{HCWC}$ rules $\mathcal{R}$ where there can be multiple rules that share the same head or body. We evaluate $\mathcal{R}$ iteratively as follows. At iteration 0, we set $\varphi^{(0)}(s,t|\mathcal{G}) = B(s,t|\mathcal{G})$ to be some extensional predicate value for all $s,t \in \mathcal{V}$ and $\varphi \in \Phi$. At iteration $l > 0$, we also initially set $\varphi^{(l)}(s,t|\mathcal{G}) = 0$. We then evaluate the rules in turn. For each rule of the form $\varphi \leftarrow B_1, B_2, ..., B_T$ where $\varphi \in \Phi$ and $B_1, B_2, ..., B_T \in \Phi \cup \mathcal{P}$, we evaluate the following equation in turn:

$$\varphi^{(l)}(s,t|\mathcal{G}) := \varphi^{(l)}(s,t|\mathcal{G}) + \sum_{s,t,e_1,e_2,...\in\mathcal{V}} B_1^{(l-1)}(s,e_1|\mathcal{G})\ B_2^{(l-1)}(e_1,e_2|\mathcal{G})\ ...\ B_T^{(l-1)}(e_{T-1},t|\mathcal{G}).$$

Therefore, at the end of evaluation of iteration l, each $\varphi^{(l)} \in \Phi$ is a synthesis of all the rules $r \in \mathcal{R}$ that share the same rule head φ. With the above definitions, we now prove the theorem.

Proof. We first establish the relationship between $\mathcal{HCWC}$ with the two GNN implementations.

Lemma A.2. *Given any graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ and $s,t \in \mathcal{V}_1, p,q \in \mathcal{V}_2$. Let $\boldsymbol{z}_{s,t}^{(l)}, \boldsymbol{z}_{p,q}^{(l)}$ be the embeddings at $(s,t),(p,q)$ computed by Eq. 11 and $\boldsymbol{z}_{s,t}^{\prime(l)}, \boldsymbol{z}_{p,q}^{\prime(l)}$ be the embeddings at $(s,t),(p,q)$ computed by Eq. 12, respectively. We initialize $\boldsymbol{z}_{s,t}^{(0)} = \boldsymbol{z}_{s,t}^{\prime(0)} = \boldsymbol{\phi}_{s,t}$ to be some inputs. The following statements are equivalent:*

- *$\boldsymbol{z}_{s,t}^{(l)} = \boldsymbol{z}_{p,q}^{(l)}$ holds, for any positive integer l and parameter matrices $\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3, \boldsymbol{W}_4$.*

- $\boldsymbol{z}'^{(l)}_{s,t} = \boldsymbol{z}'^{(l)}_{p,q}$ *holds, for any positive integer* l *and parameter matrices* $\boldsymbol{W}'_1, \boldsymbol{W}'_2, \boldsymbol{W}'_3, \boldsymbol{W}'_4$.
- *All* $\mathcal{HCWC}$ *predicates evaluate* (s,t) *and* (p,q) *to the same value, i.e.* $\varphi^{(l)}(s,t|\mathcal{G}_1) = \varphi^{(l)}(p,q|\mathcal{G}_2)$ *for any positive integer* l.

With the above lemma we can see that the Theorem holds. □

A.1. Proof of Lemmas

We now prove Lemma A.2.

($\neg 2 \Rightarrow \neg 1$) Suppose at iteration l and with parameter matrices $\boldsymbol{W}'_1 \in \mathbb{R}^{m\times r}, \boldsymbol{W}'_2 \in \mathbb{R}^{m\times m}, \boldsymbol{W}'_3 \in \mathbb{R}^{r\times m}, \boldsymbol{W}'_4 \in \mathbb{R}^{r\times d}$,

$$\boldsymbol{z}'^{(l)}_{s,t} \neq \boldsymbol{z}'^{(l)}_{p,q}.$$

We construct $\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3, \boldsymbol{W}_4$ as follows.

$$\begin{aligned}
\boldsymbol{W}_1 &= \begin{bmatrix} \boldsymbol{O}_{d\times r} \\ \boldsymbol{W}'_1 \end{bmatrix}, \\
\boldsymbol{W}_2 &= \begin{bmatrix} \boldsymbol{I}_d \\ \boldsymbol{W}'_2 \end{bmatrix}, \\
W_3 &= [\boldsymbol{O}_{r\times d}, \boldsymbol{W}'_3], \\
W_4 &= [\boldsymbol{W}'_4, \boldsymbol{O}_{r\times m}].
\end{aligned}$$

Obviously, the evaluation of $\boldsymbol{z}^{(l)}_{s,t}$ and $\boldsymbol{z}'^{(l)}_{s,t}$, $\boldsymbol{z}^{(l)}_{p,q}$ and $\boldsymbol{z}'^{(l)}_{p,q}$ at each iteration l are exactly equivalent.

($\neg 1 \Rightarrow \neg 3$) Assume that at iteration l and with parameter matrices $\boldsymbol{W}_1 \in \mathbb{R}^{m\times r}, \boldsymbol{W}_2 \in \mathbb{R}^{m\times m}, \boldsymbol{W}_3 \in \mathbb{R}^{r\times m}, \boldsymbol{W}_4 \in \mathbb{R}^{r\times m}$,

$$\boldsymbol{z}^{(l)}_{s,t} \neq \boldsymbol{z}^{(l)}_{p,q}.$$

We develop a set of $\mathcal{HCWC}$ rules as follows. The rules contain m intensional predicates denoted as $\varphi_1, \varphi_2, ..., \varphi_m$, and $m^2 + m^3 r$ rules. For $i \in \{0,1,2,...,m-1\}$ and $j \in \{0,1,2,...,m-1\}$, the $(im+j)$-th rule is:

$$\varphi_i \leftarrow \varphi_j : \boldsymbol{W}_2[i,j].$$

For $i \in \{0,1,...,m\}$, $j \in \{0,1,...,m-1\}$, $k \in \{0,1,...,m-1\}$ and $h \in \{0,1,...,r-1\}$, the $(m^2+i+jm+hm^2+km^2r)$-th rule is:

$$\varphi_k \leftarrow \varphi_i, \varphi_j : \boldsymbol{W}_1[k,h]\ \boldsymbol{W}_3[h,i]\ \boldsymbol{W}_4[h,j].$$

Obviously, by setting the initial values to be $\varphi^{(0)}_i(s,t|\mathcal{G}_1) = \boldsymbol{\phi}_{s,t}, \varphi^{(0)}_i(p,q|\mathcal{G}_2) = \boldsymbol{\phi}_{p,q}$. $\varphi_1, \varphi_2, ..., \varphi_m$ precisely capture the m dimensions of $\boldsymbol{z}^{(l)}_{s,t}$ at each iteration l and (s,t). Therefore, if $\boldsymbol{z}^{(l)}_{s,t}[i] \neq \boldsymbol{z}^{(l)}_{p,q}[i]$ at some dimension i, we also have

$$\varphi^{(l)}_i(s,t|\mathcal{G}_1) \neq \varphi^{(l)}_i(p,q|\mathcal{G}_2).$$

($\neg 3 \Rightarrow \neg 2$) Suppose that at iteration l we have $\varphi^{(l)}(s,t|\mathcal{G}_1) \neq \varphi^{(l)}(p,q|\mathcal{G}_2)$. By unrolling the rules and explicitly express the l-rounds of evaluation, we can express the rules in the forms of

$$\begin{aligned}
&\varphi \leftarrow B_1, B_2, ..., B_T : \alpha, \\
&..., \\
&\psi_1 \leftarrow B_2, \psi_2 : 1, \\
&... \\
&\psi_{T-2} \leftarrow \psi_{T-1}, B_{T-1} : 1, \\
&\psi_{T-1} \leftarrow B_T : 1.
\end{aligned}$$

where $B_1, B_2, ..., B_T$ are all extensional predicates, and one-round applying of the rules produces the same results with the original $\varphi^{(l)}$. Thus, there must exists some rule whose evaluation on (s,t) and (p,q) are different. Suppose the rule is $\varphi \leftarrow B_1, B_2, ..., B_T : \alpha$. We then transform the rule into a more compact form:

Similar with before, these rules align well with the evaluation of Eq. 12 and thus we can design $\boldsymbol{W}_1', \boldsymbol{W}_2', \boldsymbol{W}_3', \boldsymbol{W}_4'$ to make $\boldsymbol{z}_{s,t}'^{(l)}, \boldsymbol{z}_{p,q}'^{(l)}$ exactly capture these rules at iteration l, As a consequence, at iteration T, we have

$$\boldsymbol{z}_{s,t}'^{(T)} \neq \boldsymbol{z}_{p,q}'^{(T)}.$$

B. Proof of Prop. 4.1

In this section we prove the Proposition 4.1 stated as follows:

Proposition B.1. *The sequence computed by*

$$\boldsymbol{Z}_s^{(l+1)} = \sum_{i=1}^{D} \boldsymbol{A}_i \boldsymbol{Z}_t^{(l)} \boldsymbol{\Theta}_i, \tag{14}$$

and

$$\boldsymbol{z}'^{(l+1)}_{s,t} = \boldsymbol{W}_1 \sum_{e \in \mathcal{V}} (\boldsymbol{W}_3 \boldsymbol{z}'^{(l)}_{s,e}) \odot (\boldsymbol{W}_4 \boldsymbol{x}_{e,t}) + \boldsymbol{W}_2 \boldsymbol{z}'^{(l)}_{s,t}, \tag{15}$$

share the same discriminative power. Formally, given any $\boldsymbol{W}_1 \in \mathbb{R}^{m \times r}, \boldsymbol{W}_2 \in \mathbb{R}^{m \times d}, \boldsymbol{W}_3 \in \mathbb{R}^{r \times d}, \boldsymbol{W}_4 \in \mathbb{R}^{r \times m}$ and input $\{\boldsymbol{A}_i \in \mathbb{R}^{N \times N}\}_{i=1}^d, \boldsymbol{X}_t, \boldsymbol{Z}_t^{(0)} \in \mathbb{R}^{N \times m}$, there exists $\{\boldsymbol{\Theta}_i \in \mathbb{R}^{m \times m}\}_{i=1}^d$, such that the infinite sequence $\{\boldsymbol{Z}_t^{(l)}\}_l$ induced by Eq. 14 and Eq. 15 are the same. Vise versa.

Proof. (Eq. 15 $\Rightarrow$ Eq. 14): We first know that, according to the definitions of Z_t, Eq. 15 can be equally expressed as

$$\boldsymbol{Z}_t^{(l+1)} = \sum_{i=1}^{d} \boldsymbol{A}_i \boldsymbol{Z}_t^{(l)} \boldsymbol{W}_1 \mathrm{Diag}(\boldsymbol{\alpha}_i) \boldsymbol{W}_3 + \boldsymbol{Z}_t^{(l)} \boldsymbol{W}_2 \tag{16}$$

Then it's obvious that by absorbing the $\boldsymbol{Z}_t^{(l)} \boldsymbol{W}_2$ term into the augmented adjacency tensor, Eq. 14 precisely capture Eq. 16.

(Eq. 14 $\Rightarrow$ Eq. 15): Given any $\{\boldsymbol{\Theta}_i\}_{i=1}^d$, we first let $r = md$, and construct $\boldsymbol{W}_1 \in \mathbb{R}^{m \times r}, \boldsymbol{W}_3 \in \mathbb{R}^{r \times m}, \boldsymbol{W}_4 = [\boldsymbol{w}_1, ..., \boldsymbol{w}_d] \in \mathbb{R}^{r \times d}$ as follows. We let $\boldsymbol{W}_1$ be such a block matrix:

$$\boldsymbol{W}_1 = [\boldsymbol{I}, \boldsymbol{I}, ..., \boldsymbol{I}],$$

where $\boldsymbol{I} \in \mathbb{R}^{m \times m}$ is an identity matrix. We then let

$$\boldsymbol{W}_3 = [\boldsymbol{\Theta}_1^T, \boldsymbol{\Theta}_2^T, ..., \boldsymbol{\Theta}_d^T]^T.$$

For $\boldsymbol{w}_i$, we let

$$\boldsymbol{w}_i[j] = \begin{cases} 1, & m(i-1) \le j < mi, \\ 0, & \text{else.} \end{cases}$$

Then, it's easy to observe that

$$\begin{aligned} &\boldsymbol{W}_1 \mathrm{Diag}(\boldsymbol{w}_i) \boldsymbol{W}_3 \\ =&[\boldsymbol{I}, \boldsymbol{I}, ..., \boldsymbol{I}][O, ..., O, \boldsymbol{\Theta}_i^T, O, ..., O]^T \\ =&\boldsymbol{\Theta}_i, \end{aligned}$$

By additionally setting $\boldsymbol{W}_2 = \boldsymbol{\Theta}_D$, Eq. 16 can also capture any Eq. 14, and thus Eq. 15 can capture Eq. 14. □

C. Proof of Prop. 4.2 and Corollary 4.4

In this section we prove the proposition 4.2 and corollary 4.4.

C.1. Proof of Prop. 4.2

Proposition C.1. *The infinite sequence* $\{\boldsymbol{Z}^{(l)}\}_l$ *defined as follows:*

$$\boldsymbol{Z}^{(l+1)} = \sum_{i=1}^{D} \boldsymbol{S}_i \boldsymbol{Z}^{(l)} \boldsymbol{\Theta}_i + \boldsymbol{B} \tag{17}$$

is guaranteed to converge given the bias matrix $\boldsymbol{B} \in \mathbb{R}^{N\times m}$*. Moreover, the fixed point* $\boldsymbol{Z}^*$ *is irrelevant to the initial point* $\boldsymbol{Z}^{(0)}$*.*

Proof. We first unroll Eq. C.1 and express it in the language of vectors $\boldsymbol{z}_{s,t}^{(l)}$ where $\boldsymbol{Z}^{(l)} = [\boldsymbol{z}_{s,0}^{(l)}, ..., \boldsymbol{z}_{s,N}^{(l)}]^T$ as follows:

$$\boldsymbol{z}_{s,t}^{(l+1)} = \sum_{e\in\mathcal{V}} \frac{1}{\deg_{\mathcal{O}}(e)} \boldsymbol{C}_{e,t} \boldsymbol{z}_{s,e}^{(l)} + \boldsymbol{B}[t,:],$$
$$\text{where} \quad \boldsymbol{C}_{e,t} = \deg_{\mathcal{O}}(e) \sum_{i=1}^{D} \boldsymbol{S}_i[e,t] \boldsymbol{\Theta}_i.$$

First, with the following lemma we show that $\boldsymbol{C}_{e,t}$ is a contractive mapping.

Lemma C.2. *There exists some* γ*, such that* $\forall i, j \in \mathcal{V}$*,*

$$\|\boldsymbol{C}_{i,j}\|_2 \leq \gamma < 1.$$

Since $C_{s,e}$ is contractive, we next use Banach fixed-point theorem to prove the proposition. To do so, we define a function $\mathscr{D} : \mathbb{R}^{N\times m} \times \mathbb{R}^{N\times m}$, as follows:

$$\mathscr{D}(\boldsymbol{Z}_p, \boldsymbol{Z}_q) = \sum_{n=1}^{N} \|\boldsymbol{Z}_p[n,:] - \boldsymbol{Z}_q[n,:]\|_2 .$$

With the following lemma we can see that $\mathscr{D}$ is a metric over the space of the variables $Z_t^{(l)}$.

Lemma C.3. $\mathscr{D}$ *is a complete metric of the set* $\mathbb{R}^{N\times m}$*.*

Let $\mathscr{T}(\boldsymbol{Z}) = \boldsymbol{Z}^{(l+1)} = \sum_{i=1}^{D} \boldsymbol{S}_i \boldsymbol{Z}^{(l)} \boldsymbol{\Theta}_i + \boldsymbol{B}$. Now we show that $\mathscr{T}$ is a contractive mapping. For all $\boldsymbol{Z}_p =$

$[\boldsymbol{z}_{p,0}, ..., \boldsymbol{z}_{p,N}]^T, \boldsymbol{Z}_q = [\boldsymbol{z}_{q,0}, ..., \boldsymbol{z}_{q,N}]^T \in \mathbb{R}^{N\times m}$, we have

$$\begin{aligned}
\mathscr{D}(\mathscr{T}(\boldsymbol{Z}_p), \mathscr{T}(\boldsymbol{Z}_q)) &= \sum_{t\in\mathcal{V}} \|\mathscr{T}(\boldsymbol{Z}_p)[t,:], \mathscr{T}(\boldsymbol{Z}_p)[t,:]\|_2 \\
&= \sum_{t\in\mathcal{V}} \left\| \left(\sum_{e\in\mathcal{V}} \frac{1}{\deg_{\mathcal{O}}(e)} \boldsymbol{C}_{e,t} \boldsymbol{z}_{p,e} + \boldsymbol{B}[t,:] \right) - \left(\sum_{e\in\mathcal{V}} \frac{1}{\deg_{\mathcal{O}}(e)} \boldsymbol{C}_{e,t} \boldsymbol{z}_{q,e} + \boldsymbol{B}[t,:] \right) \right\| \\
&= \sum_{t\in\mathcal{V}} \left\| \sum_{e\in\mathcal{V}} \frac{1}{\deg_{\mathcal{O}}(e)} \boldsymbol{C}_{e,t} \left(\boldsymbol{z}_{p,e} - \boldsymbol{z}_{q,e} \right) \right\| \\
&\leq \sum_{t\in\mathcal{V}} \sum_{e\in\mathcal{V}} \frac{1}{\deg_{\mathcal{O}}(e)} \|\boldsymbol{C}_{e,t}(\boldsymbol{z}_{p,e} - \boldsymbol{z}_{q,e})\| \\
&\leq \sum_{t\in\mathcal{V}} \sum_{e\in\mathcal{N}_{\mathcal{I}}(t)} \frac{\gamma}{\deg_{\mathcal{O}}(e)} \|\boldsymbol{z}_{p,e} - \boldsymbol{z}_{q,e}\| \\
&= \sum_{e\in\mathcal{V}} \sum_{t\in\mathcal{N}_{\mathcal{O}}(e)} \frac{\gamma}{\deg_{\mathcal{O}}(e)} \|\boldsymbol{z}_{p,e} - \boldsymbol{z}_{q,e}\| \\
&= \gamma \sum_{e\in\mathcal{V}} \|\boldsymbol{z}_{p,e} - \boldsymbol{z}_{q,e}\| \\
&= \gamma \mathscr{D}(\boldsymbol{Z}_p, \boldsymbol{Z}_q).
\end{aligned}$$

Therefore, $\mathscr{T}$ is a contractive mapping. As a result, the equation $\mathscr{T}(\boldsymbol{Z}) = \boldsymbol{Z}$ has exactly one solution $\boldsymbol{Z}^*$, and the sequence $\boldsymbol{Z}^{(0)}, \boldsymbol{Z}^{(1)}, ...$ is converged at the solution $\boldsymbol{Z}^*$, for an arbitrary choice of the initial point $\boldsymbol{Z}^{(0)}$. □

C.2. Proof of Corollary 4.4

Given that Proposition C.1 holds, it is straightforward to show that Corollary 4.4 holds.

Corollary C.4. *The solution of*

$$\begin{aligned}
\boldsymbol{Y}_s &= \boldsymbol{Z}_s \boldsymbol{F}, \\
\boldsymbol{Z}_s &= \gamma \sum_{i=1}^{D} \boldsymbol{S}_i \boldsymbol{Z}_s \boldsymbol{\Theta}_i + \boldsymbol{X}_s \boldsymbol{W},
\end{aligned} \tag{18}$$

exists and is unique, given any $\gamma \in (0, 1], \epsilon > 0$ *and* $X_t \in \mathbb{R}^{N\times d}$.

Proof. First, for $i = 1, 2, ..., d$, we have

$$\|g(\boldsymbol{\Theta}_i)\|_2 \leq \|g(\boldsymbol{\Theta}_i)\|_F \leq \max_j \frac{\|\boldsymbol{\Theta}_j\|_F}{\|\boldsymbol{\Theta}_j\|_F + \epsilon} < 1.$$

Thus, Corollary C.4 is an instantiation of Proposition C.1. □

C.3. Proof of Lemmas

Proof of Lemma C.2. Recall that $\boldsymbol{S}_i = \boldsymbol{D}^{-1}\tilde{\boldsymbol{A}}_i$, i.e. $\boldsymbol{S}_i[s,t] = \frac{1}{\deg_{\mathcal{O}}(s)}\tilde{\boldsymbol{A}}_i[s,t]$, and thus we have

$$\sum_{k=1}^{D} \boldsymbol{S}_k[i,j] \leq \sum_{k=1}^{D} |\boldsymbol{S}_k[i,j]| = \frac{1}{\deg_{\mathcal{O}}(\mathrm{i})} ||\tilde{\mathbf{A}}[:,i,j]||_1 \leq \frac{1}{deg_{\mathcal{I}}(j)}.$$

Then, we have

$$\begin{aligned}||\boldsymbol{C}_{i,j}||_2 &\leq ||\boldsymbol{C}_{i,j}||_F \\ &= \left\| \sum_{k=1}^{D} \boldsymbol{S}_k[i,j]\boldsymbol{\Theta}_k \mathrm{deg}_{\mathcal{O}}(i) \right\|_F \\ &= \left\| \sum_{k=1}^{D} \tilde{\boldsymbol{A}}_k[i,j]\Theta_k \right\|_F \\ &= \frac{\|\mathbf{A}[i,j,:]\|_\Theta}{\|\mathbf{A}[i,j,:]\|_\Theta + \epsilon} \\ &\leq \max_{i,j} \frac{\|\mathbf{A}[i,j,:]\|_\Theta}{\|\mathbf{A}[i,j,:]\|_\Theta + \epsilon} < 1.\end{aligned}$$

By letting $\gamma = \max_{i,j} \frac{\|\mathbf{A}[i,j,:]\|_\Theta}{\|\mathbf{A}[i,j,:]\|_\Theta + \epsilon}$, we can see that

$$||\boldsymbol{C}_{i,j}||_2 \leq \gamma < 1.$$

□

Proof of Lemma C.3. We now show that $\mathscr{D}$ is a metric over $\mathbb{R}^{N\times m}$, i.e. the space of variables $\boldsymbol{Z}_s^{(l)}$.

(I) (Positive qualitative). $\forall \boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^{N\times m}$, we have

$$\mathscr{D}(\boldsymbol{X}, \boldsymbol{Y}) = \sum_{n=1}^{N} \|\boldsymbol{X}[n,:] - \boldsymbol{Y}[n,:]\|_2 \geq 0.$$

Suppose $\mathscr{D}(\boldsymbol{X}, \boldsymbol{Y}) = 0$. Then, we have

$$\|\boldsymbol{X}[n,:] - \boldsymbol{Y}[n,:]\|_2 = 0 \quad \Longleftrightarrow \quad \begin{aligned}&\boldsymbol{X}[n:] = \boldsymbol{Y}[n,:], \\ &\text{for} \quad n \in \{1,2,...,N\}.\end{aligned}$$

As a result, $\mathscr{D}(\boldsymbol{X}, \boldsymbol{Y}) = 0 \iff X = Y$.

(II) (Symmetry). We have

$$\begin{aligned}\mathscr{D}(\boldsymbol{X}, \boldsymbol{Y}) &= \sum_{n=1}^{N} \|\boldsymbol{X}[n,:] - \boldsymbol{Y}[n,:]\|_2 \\ &= \sum_{n=1}^{N} \|\boldsymbol{Y}[n,:] - \boldsymbol{X}[n,:]\|_2 \\ &= \mathscr{D}(\boldsymbol{Y}, \boldsymbol{X}).\end{aligned}$$

(III) (Triangle inequality). $\forall \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z} \in \mathbb{R}^{N\times m}$, we have

$$\begin{aligned}\mathscr{D}(\boldsymbol{X}, \boldsymbol{Y}) &= \sum_{n=1}^{N} \|\boldsymbol{X}[n,:] - \boldsymbol{Y}[n,:]\|_2 \\ &\leq \sum_{n=1}^{N} \|\boldsymbol{X}[n,:] - \boldsymbol{Z}[n,:]\|_2 + \|\boldsymbol{Z}[n,:] - \boldsymbol{Y}[n,:]\|_2 \\ &= \mathscr{D}(\boldsymbol{X}, \boldsymbol{Z}) + \mathscr{D}(\boldsymbol{Z}, \boldsymbol{Y}).\end{aligned}$$

(IV) (Completeness). Suppose $X_1, X_2, ... \in \mathbb{R}^{N\times m}$ is a Cauchy sequence in the metric space $(\mathbb{R}^{N\times m}, \mathscr{D})$. Then, for any $r > 0$, there is a positive integer P such that for all positive integers $p, q > P$,

$$\mathscr{D}(\boldsymbol{X}_p, \boldsymbol{X}_q) = \sum_{n=1}^{N} \|\boldsymbol{X}_p[n,:] - \boldsymbol{X}_q[n,:]\|_2 \leq r.$$

Thus, for $n = 1, 2, ..., N$,

$$\|\boldsymbol{X}_p[n,:] - \boldsymbol{X}_q[n,:]\|_2 \leq r.$$

As a result, $\boldsymbol{X}_1[n,:], \boldsymbol{X}_2[n,:], ...$ is also a Cauchy sequence in the metric space $(\mathbb{R}^m, \|\cdot\|_2)$. Since $\|\cdot\|_2$ is a complete metric in the real-valued vector space $\mathbb{R}^m$, we have

$$\lim_{i\to\infty} \boldsymbol{X}_i[n,:] \in \mathbb{R}^m$$

for all $n = 1, 2, ..., N$. As a direct corollary, we have

$$\lim_{i\to\infty} \boldsymbol{X}_i \in \mathbb{R}^{N\times m}.$$

□

D. Proof of Prop. 5.1 and 5.2

D.1. Proof of Prop. 5.1

Proposition D.1. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, LIGNN maps all disconnected nodes $s, t \in \mathcal{V}$ with equal link representations $\boldsymbol{z}_{s,t}$. Given two graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1), \mathcal{G}_2$, we can combine $\mathcal{G}_1$ and $\mathcal{G}_2$ into one graph $\mathcal{G}_3 = (\mathcal{V}_3, \mathcal{E}_3)$. Let $s, t \in \mathcal{V}_1$ and $s', t' \in \mathcal{V}_3$ be the corresponding vertices of s, t in $\mathcal{G}_1$, respectively. Then, LIGNN always maps (s, t) and (s', t') to the same link representations.*

Proof. We first show disconnected node pair (s, t) always satisfies $\boldsymbol{z}_{s,t} = \boldsymbol{0}$.

Observe: At iteration 0, disconnected $\boldsymbol{z}_{s,t}^{(0)} = \boldsymbol{x}_{s,t} = \boldsymbol{0}$.

Assume: At iteration l, disconnected $\boldsymbol{z}_{s,t}^{(l)} = \boldsymbol{0}$.

At iteration $l+1$, if

$$\boldsymbol{z}^{(l+1)} = \sum_{e \in \mathcal{V}} \boldsymbol{W}_{e,t} \boldsymbol{z}_{s,e}^{(l)} + \boldsymbol{W} \boldsymbol{x}_{s,t}$$

is nonzero, then there are 2 situations:

1. $\boldsymbol{x}_{s,t} \neq \boldsymbol{0}$: False because s and t are not connected.

2. $\boldsymbol{z}_{s,e}^{(l)} \neq \boldsymbol{0}$ and $\boldsymbol{W}_{e,t} \neq \boldsymbol{O}$: False because (s, e) and (e, t) cannot both be connected, thus with the assumption, we always have $\boldsymbol{z}_{s,e}^{(l)} = \boldsymbol{0}$ or $\boldsymbol{W}_{e,t} = \boldsymbol{O}$.

With proof by induction, disconnected (s, t) must always satisfies $\boldsymbol{z}_{s,t} = \boldsymbol{0}$.

We now show LIGNN is invariant under different graph settings. Since the update of LIGNN can be written as:

$$\boldsymbol{z}^{(l+1)} = \sum_{e \in \mathcal{N}_{\mathcal{I}}(t)} \boldsymbol{W}_{e,t} \boldsymbol{z}_{s,e}^{(l)} + \boldsymbol{W} \boldsymbol{x}_{s,t},$$

each update of a link (s, t) only incorporates its local neighbors, and thus stays invariant under the different graph settings. □

D.2. Proof of Prop. 5.2

Proposition D.2. *There are two graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1), \mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ and $s, t \in \mathcal{V}_1, p, q \in \mathcal{V}_2$, such that LIGNN maps (s, t) and (p, q) to different representations, while 1-WL test maps (s, t) and (p, q) to be equivalent.*

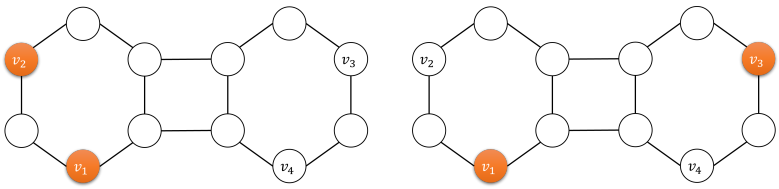

Figure 3. Accuracy of graph connectiveness task.

Proof. Figure 3 provides a counterexample. By applying 1-WL test we assign each node a color. However, by aggregating node colors together, (v_1, v_2) and (v_1, v_3) are exactly the same. For LIGNN, by starting from the node v_1, it's obvious that LIGNN can learn different embeddings for $\boldsymbol{z}_{v_1,v_2}$ and $\boldsymbol{z}_{v_1,v_3}$, thus (v_1, v_2) and (v_1, v_3) are distinguishable for LIGNN. □

E. Detailed experiment configurations

Model configurations are listed in Tabel 4. Across all experiments, model configurations are similar, and the reasons for the differences in configurations are as follows. For synthetic data, we set $m = 2$ to prevent overfitting, and set larger max iteration number to capture longer-range dependencies. KGC are with less demands for capturing long-range dependencies, so we set $\gamma = 0.8$.

In KGC task, the number of parameters is about 40k for FB15k-237 datasets, while for other datasets it is about 3k.

Table 4. Configuration

TASK	#PARAMS.	m	γ	MAX ITER.	OPT.	LR.	EPOCH
SYNTHETIC	9	2	1.0	200	ADAM	1e−1	1000
PCQM-CONTACT	2541	10	1.0	50	ADAM	1e−3	2
KGC	$\sim$ 3K	10	0.8	50	ADAM	1e−3	100

F. Discussion of normalization methods

The normalization method we use is

$$\tilde{\mathbf{A}}[s,t,:] = \frac{\mathbf{A}[s,t,:]}{\|\mathbf{A}[s,t,:]\|_\Theta + \epsilon}.$$

Since this process incorporates the parameters $\{\boldsymbol{\Theta}_i\}$, on different training steps we need to re-normalize edge features to obtain different $\{\boldsymbol{S}_i\}$. There are other normalization methods that does not requires $\{\boldsymbol{\Theta}_i\}$ as inputs. For example, consider

$$\begin{aligned}
&\boldsymbol{S}_i = \boldsymbol{D}^{-1}\hat{\boldsymbol{A}}_i, \qquad \text{where}\\
&\hat{\boldsymbol{A}}_i = \hat{\mathbf{A}}[:,:,i],\\
&\hat{\mathbf{A}}[s,t,:] = \frac{\mathbf{A}[s,t,:]}{|\mathbf{A}[s,t,:]|_1 + \epsilon}.
\end{aligned}$$

In this setting, we assume $\|\boldsymbol{\Theta}_i\|_F = 1$. This method also guarantees the convergency of the model, however, as will be shown later, it cannot efficiently capture long-range dependencies.

From Appendix C we can rewrite the LIGNN iterations as

$$\boldsymbol{z}_{s,t}^{(l+1)} = \sum_{e\in\mathcal{V}} \frac{1}{\deg_{\mathcal{O}}(e)} \boldsymbol{C}_{e,t}\boldsymbol{z}_{s,e}^{(l)} + \boldsymbol{B}[t,:].$$

In the original normalization method, if we set $\epsilon \to 0$, then for every $e \in N(v)$, $\|\boldsymbol{C}_{e,t}\|_F \to 1$. For the variant, we also have

$$
\begin{aligned}
&\|\boldsymbol{C}_{e,t}\|_F \\
&= \left\| \sum_{k=1}^{D} \boldsymbol{S}_k[e,t]\boldsymbol{\Theta}_k \right\|_F deg_{\mathcal{O}}(e) \\
&= \sqrt{\sum_{i,j} \left(\sum_{k=1}^{D} \hat{\mathbf{A}}[e,t,k]\boldsymbol{\Theta}_k[i,j] \right)^2} deg_{\mathcal{O}}(e) \\
&= \left\| \sum_{k=1}^{D} \hat{\mathbf{A}}[e,t,k]\mathrm{vec}(\boldsymbol{\Theta}_k) \right\|_2 \\
&\leq \sum_{k=1}^{D} \left\| \hat{\mathbf{A}}[e,t,k]\mathrm{vec}(\boldsymbol{\Theta}_k) \right\|_2 \\
&= \sum_{k=1}^{D} \left| \hat{\mathbf{A}}[e,t,k] \right| \ \|\mathrm{vec}(\boldsymbol{\Theta}_k)\|_2 \\
&= \sum_{k=1}^{D} \left| \hat{\mathbf{A}}[e,t,k] \right| \ \|\boldsymbol{\Theta}_k\|_F \\
&\leq \max_k ||\boldsymbol{\Theta}_k||_F.
\end{aligned}
$$

The value $\left\| \sum_{k=1}^{D} \hat{\mathbf{A}}[e,t,k]\mathrm{vec}(\boldsymbol{\Theta}_k) \right\|_2$ is the length of the composition of D vectors from $\mathbb{R}^{m^2}$ space. Since the expectation of the composition of random vectors is $\mathbf{0}$ and the total length of the D vectors is 1, the length of the composed vector is often much smaller than 1. Therefore, in this setting $\|\boldsymbol{C}_{e,t}\|_F$ is often a very small value. As a result, the model contracts $\boldsymbol{z}_{s,t}^{(l)}$ too much at each layer and cannot effectively capture long-range dependencies.

G. Implementation

In this section we provide a `Pytorch` style pseudocode that implements the forward and backward pass of LIGNN. The code is for illustration and based on dense tensors, and in realistic we can replace them with sparse operations.

```
import torch
import torch.nn as nn

class LILayer(nn.Module):
    '''
    The implicit layer
    '''
    def __init__(self, d:int, m:int, d_out:int,
                 max_iter=50, gamma=0.8, epsilon=1e-5, tol=1e-5):
        super().__init__()
        self.W_linear = nn.Linear(d, m, bias=False)
        self.Theta = nn.Parameter(torch.randn((d, m, m)))
        self.F_linear = nn.Linear(m, d_out, bias=True)

        self.max_iter = max_iter
        self.gamma = gamma
        self.epsilon = epsilon
        self.tol = tol

    def norm_Theta(self, A):
        '''Compute the Theta-norm

        Params:
          A: the input vectors, [*channels, d]
        '''
        vec_Theta = self.Theta.view(A.shape[-1], -1)
        return (A @ vec_Theta).norm(p='fro', dim=-1)

    def solve(self, f, x0):
        '''A iterative solver

        Params:
          f: the iteration function
          x0: initial value
        '''
        f0 = f(x0)
        for _ in range(self.max_iter):
            x = f0
            f0 = f(x)
            res = (f0 - x).norm() / (f0.norm() + 1e-5)
            if res < self.tol:
                break
        return f0

    def forward(self, s:int, A, inv_D):
        '''Apply forward propogation

        Params:
```

```
        s: source vertex, scalar.
        A: adjacency tensor, [N, N, d]
        inv_D: inverse degree matrix, [N, N]
      '''

      # Step 1: Linearly project X_s
      X = A[:, s, :]
      X = self.W_linear(X)

      # Step 2: Normalize A and Theta
      norm_A = self.norm_Theta(A)
      A = A.permute(2,0,1)
      A = A / (norm_A + self.epsilon)
      S = inv_D @ A

      # Step 3: Compute fixed point Z_s
      def f(Z0):
          Z1 = self.gamma * (S @ Z0 @ self.Theta).sum(dim=0) + X
          return Z1

      with torch.no_grad():
          Z = self.solve(f, X)
      Z = f(Z)

      # Step 4: Engage automatic differentiation
      Z0 = Z.clone().detach().requires_grad_()
      f0 = f(Z0)
      def backward_hook(grad):
          g = self.solve(lambda y:torch.autograd.grad(f0,
                              Z0, y, retain_graph=True)[0]
                         + grad, grad)
          return g
      if Z.requires_grad:
          Z.register_hook(backward_hook)

      # Step 5: Compute Y_s
      Y = self.F_linear(Z)
      return Y
```

H. 2-FWL tests

In this section we briefly introduce the k-FWL graph isomorphism tests (Cai et al., 1989).

H.1. 1-WL

The 1-WL test is also known as color refinement and shares similar message passing process with node-level GNNs. To begin with, each node v is assigned with a color c_v. The 1-WL test can then be summarized as follows:

Algorithm 1 1-WL test

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: the input graph; c_v for $v \in \mathcal{V}$: initial colors;
Output: the final colors;
1: $c_v^0 \leftarrow c_v$ for all $v \in \mathcal{V}$;
2: **repeat**
3: $\quad \forall v \in \mathcal{V}, c_v^{l+1} \leftarrow \text{hash}(c_v^l, \{\{c_w^l \mid w \in N(v)\}\})$;
4: **until** $\forall v \in \mathcal{V}, c_v^{l+1} = c_v^l$;
5: **return** c_v^l for every $v \in \mathcal{V}$;

Here, $N(v)$ is the set of the neighbors of v in $\mathcal{G}$. The critical part is the hash function hash. It needs to be injective in order to fully express the discriminative power of the 1-WL test.

H.2. 2-FWL

The 2-FWL test is summarized as follows.

Algorithm 2 2-FWL test

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$: the input graph; $c_{s,t}$ for $s, t \in \mathcal{V}$: initial colors;
Output: the final colors;
1: $c_{s,t}^0 \leftarrow c_{s,t}$ for all $s, t \in \mathcal{V}$;
2: **repeat**
3: $\quad \forall s, t \in \mathcal{V}, c_{s,t}^{l+1} \leftarrow \text{hash}(c_{s,t}^l, \{\{c_{\boldsymbol{e}}^l \mid \boldsymbol{e} \in N(s,t)\}\})$;
4: **until** $\forall s, t \in \mathcal{V}, c_{s,t}^{l+1} = c_{s,t}^l$;
5: **return** $c_{s,t}^l$ for every $s, t \in \mathcal{V}$;

The difference between the 2-FWL test and the 1-WL is that we now assign a color for each node pair (s, t) instead of a single node. Generally, the 2-FWL test follows the same computation procedure with the 1-WL, but we need to redefine the neighbors, i.e. $N(s, t)$, of a node pair (s, t). In the 2-FWL test, we set each node pair (s, t) to have $|\mathcal{V}|$ neighbors, with the i-th neighbor being

$$((s, i), (i, t)),$$

which is a pair of node pairs. By denoting $N_i(s, t)$ to be the i-th neighbor of (s, t), we can see that the computation steps of the 2-FWL test exactly follows Eq. 1-3, with all functions being injective.

I. Algorithm complexities

In this section we discuss the algorithm complexities of LIGNN more detailedly.

First, in Section 3 each layer is

$$z_{s,t}^{(l+1)} = W_1 \sum_{e \in \mathcal{V}} (W_3 z_{s,e}^{(l)}) \odot (W_4 z_{e,t}^{(l)}) + W_2 z_{s,t}^{(l)}.$$

Obviously, to compute one $z_{s,t}^{(l+1)}$ for a specified (s, t), we need to summarize over N terms and store $z_{s,e}^{(l)}$ and $z_{e,t}^{(l)}$ for all $e \in \mathcal{V}$ in memory. Again, to compute $z_{s,e}^{(l)}$ and $z_{e,t}^{(l)}$ for all $e \in \mathcal{V}$ we need to store $z_{i,j}^{(l-1)}$ for all $i, j \in \mathcal{V}$. Therefore, we need $O(N^2)$ memory.

Linearizing the layer, we have

$$z_{s,t}^{(l+1)} = W_1 \sum_{e \in \mathcal{V}} (W_3 z_{s,e}^{(l)}) \odot (W_4 x_{e,t}) + W_2 z_{s,t}^{(l)} = \sum_{e \in \mathcal{V}} W_1 \mathrm{Diag}(W_4 x_{e,t}) W_3 \; z_{s,e}^{(l)} + W_2 z_{s,t}^{(l)}$$

where $\mathrm{Diag}(a)$ gives the diagonal matrix of the vector a. Denote $W_4 = [w_1, w_2, ..., w_d]$, we have

$$z_{s,t}^{(l+1)} = \sum_{i=1}^{d} \sum_{e \in \mathcal{V}} x_{e,t}[i] \left(W_1 \mathrm{Diag}(w_i) W_3\right) z_{s,e}^{(l)} + W_2 z_{s,t}^{(l)}$$

Therefore, fixing s, we can see that the above evaluation is analogous to GCNs with d heads. We can stack all $z_{s,t}^{(l)}$ with the same source node s into a single matrix $Z_s^{(l)} = \left[z_{s,1}^{(l)}, z_{s,2}^{(l)}, ..., z_{s,N}^{(l)}\right]^T$, resulting in

$$Z_s^{(l+1)} = \sum_{i=1}^{D} A_i Z_s^{(l)} \hat{\Theta}_i$$

where $A_i, \hat{\Theta}_i$ follows the definitions in Section 4.1 of the paper. The $W_2 z_{s,t}^{(l)}$ term is abbreviated by adding self loops. Therefore, the computational complexity are the same with D GCN layers, indicating that we reach $O(M/N)$ time complexity and $O(N + M)$ space complexity for evaluating one $z_{s,t}^{(l+1)}$ due to the sparsity of the input features $x_{s,t}$.