Implicit Structures for Graph Neural Networks

Fangda Gu

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-185
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-185.html

November 23, 2020

Acknowledgement

I would like to express my most sincere gratitude to Professor Laurent El Ghaoui and Professor Somayeh Sojoudi for their support in my academic life. The helpful discussions with them on my research have guided me through the past two successful years at University of California, Berkeley. I want to thank Heng Chang and Professor Wenwu Zhu for their help in the implementation and verification of the work. I also want to thank Shirley Salanio for her strong logistical and psychological support.

Implicit Structures for Graph Neural Networks

by Fangda Gu

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Laurent El Ghaoui
Research Advisor

11/19/2020

(Date)

* * * * * * *

Professor Somayeh Sojoudi
Second Reader

11/19/2020

(Date)

Implicit Structures for Graph Neural Networks

Fangda Gu

Abstract

Graph Neural Networks (GNNs) are widely used deep learning models that learn meaningful representations from graph-structured data. Due to the finite nature of the underlying recurrent structure, current GNN methods may struggle to capture long-range dependencies in underlying graphs. To overcome this difficulty, we propose a graph learning framework, called Implicit Graph Neural Networks (IGNN[1]), where predictions are based on the solution of a fixed-point equilibrium equation involving implicitly defined "state" vectors. We use the Perron-Frobenius theory to derive sufficient conditions that ensure well-posedness of the framework. Leveraging implicit differentiation, we derive a tractable projected gradient descent method to train the framework. Experiments on a comprehensive range of tasks show that IGNNs consistently capture long-range dependencies and outperform the state-of-the-art GNN models.

1 Introduction

Graph neural networks (GNNs) (Zhou et al., 2018; Zhang et al., 2020) have been widely used on graph-structured data to obtain a meaningful representation of nodes in the graph. By iteratively aggregating information from neighboring nodes, GNN models encode graph-relational information into the representation, which then benefits a wide range of tasks, including biochemical structure discovery (Gilmer et al., 2017; Wan et al., 2019), computer vision (Kampffmeyer et al., 2018), and recommender systems (Ying et al., 2018). Recently, newer convolutional GNN structures (Wu et al., 2019b) have drastically improved the performance of GNNs by employing various techniques, including renormalization (Kipf and Welling, 2016), attention (Veličković et al., 2017), and simpler activation (Wu et al., 2019a).

The aforemetioned modern convolutional GNN models capture relation information up to T-hops away by performing T iterations of graph convolutional aggregation. Such information gathering procedure is similar to forward-feeding schemes in popular deep learning models, such as multi-layer perceptron and convolutional neural networks. However, despite their simplicity, these computation strategies cannot discover the dependency with a range longer than T-hops away from any given node.

One approach tackling this problem is to develop recurrent GNNs that iterate graph convolutional aggregation until convergence, without any *a priori* limitation on the number of hops. This idea arises in many traditional graph metrics, including eigenvector centrality (Newman, 2018) and PageRank (Page et al., 1999), where the metrics are implicitly defined by some fixed-point equation. Intuitively, the long-range dependency can be better captured by iterating the information passing procedure for an infinite number of times until convergence. Pioneered by (Gori et al., 2005), new recurrent GNNs leverage partial training (Gallicchio and Micheli, 2010, 2019) and approximation

[1]Fangda has published a conference version of the work (Gu et al., 2020) on NeurIPS 2020. Some content is shared.

(Dai et al., 2018) to improve performance. With shared weights, these methods avoid exploding memory issues and achieve accuracies competitive with convolutional counterparts in certain cases.

While these methods offer an alternative to the popular convolutional GNN models with added benefits for certain problems, there are still significant limitations in evaluation and training for recurrent GNN models. Conservative convergence conditions and sophisticated training procedures have limited the use of these methods in practice, and outweighed the performance benefits of capturing the long-range dependency. In addition, most of these methods cannot leverage multi-graph information or adapt to heterogeneous network settings, as prevalent in social networks as well as bio-chemical graphs (Wan et al., 2019).

Paper contributions. In this work, we present the **I**mplicit **G**raph **N**eural **N**etwork (IGNN) framework to address the problem of evaluation and training for recurrent GNNs. We first analyze graph neural networks through a rigorous mathematical framework based on the Perron-Frobenius theory (Berman and Plemmons, 1994), in order to establish general well-posedness conditions for convergence. We show that most existing analyses are special cases of our result. As for training, we propose a novel projected gradient method to efficiently train the IGNN, where we leverage implicit differentiation methods to obtain the exact gradient, and use projection on a tractable convex set to guarantee well-posedness. We show that previous gradient methods for recurrent graph neural networks can be interpreted as an approximation to IGNN. Further, we extend IGNN to heterogeneous network settings. Finally, we conduct comprehensive comparisons with existing methods, and demonstrate that our method effectively captures long-range dependencies and outperforms the state-of-the-art GNN models on a wide range of tasks.

Paper outline. In Section 2, we give an overview of related work on GNN and implicit models. In Section 3, we introduce the background and notations for this paper. Section 4 discusses the IGNN framework together with its well-posedness and training under both ordinary and heterogeneous settings. Section 5 empirically compares IGNN with modern GNN methods.

2 Related Work

GNN models. Pioneered by (Gori et al., 2005), GNN models have gained influence for graph-related tasks. Led by GCN (Kipf and Welling, 2016), convolutional GNN models (Veličković et al., 2017; Hamilton et al., 2017; Wu et al., 2019a; Jin et al., 2019; Chang et al., 2020) involve a finite number of modified aggregation steps with different weight parameters. On the other hand, recurrent GNN models (Gori et al., 2005) use the same parameters for each aggregation step and potentially enable infinite steps. (Li et al., 2015) combines recurrent GNN with recurrent neural network structures. Methods such as Fast and Deep Graph Neural Network (FDGNN) (Gallicchio and Micheli, 2010, 2019) use untrained recurrent GNN models with novel initialization to its aggregation step for graph classification. While the Stochastic Steady-State Embedding (SSE) method (Dai et al., 2018) uses an efficient approximated training and evaluation procedure for node classification. Recently, global method Geom-GCN (Pei et al., 2020) employs additional embedding approaches to capture global information. However, Geom-GCN (Pei et al., 2020) also belongs to convolutional-based GNNs, which struggle to capture very long range dependency due to the finite iterations they take.

Implicit Models. Implicit models are emerging structures in deep learning where the outputs are determined implicitly by a solution of some underlying sub-problem. Recent works (Bai et al., 2019) demonstrate the potential of implicit models in sequence modeling, physical engine (de Avila Belbute-Peres et al., 2018) and many others (Chen et al., 2018; Amos et al., 2018).(El Ghaoui et al., 2020) proposes a general implicit framework with the prediction rule based on the solution of a fixed-point equilibrium equation and discusses the well-posedness of the implicit prediction rule.

Oversmoothing. To catch the long-range dependency, another intuitive approach is to construct deeper convolutional GNNs by stacking more layers. However, (Li et al., 2018) found that the learned node embeddings become indistinguishable as the convolutional GNNs get deeper. This phenomenon is called *over-smoothing*. Since then, a line of empirical (Li et al., 2018; Chen et al., 2020; Rong et al., 2020) and theoretical (Oono and Suzuki, 2020; Zhao and Akoglu, 2020) works follows on the *over-smoothing* phenomenon. Unlike convolutional GNNs, IGNN adopts a different approach for long-range dependency based on recurrent GNNs and doesn't seem to suffer performance degradation as much even though it could be viewed as an infinite-layer GNN. See Section 5.1.5 for details.

3 Preliminaries

Graph neural networks take input data in the form of graphs. A graph is represented by $G = (V, E)$ where V is the set of $n := |V|$ nodes (or vertices) and $E \subseteq V \times V$ is the set of edges. In practice, we construct an adjacency matrix $A \in \mathbb{R}^{n\times n}$ to represent the graph G: for any two nodes $i, j \in V$, if $(i, j) \in E$, then $A_{ij} = 1$; otherwise, $A_{ij} = 0$. Some data sets provide additional information about the nodes in the form of a feature matrix $U \in \mathbb{R}^{p\times n}$, in which the feature vector for node i is given by $u_i \in \mathbb{R}^p$. When no additional feature information from nodes is provided, the data sets would require learning a feature matrix U separately in practice.

Given graph data, graph models produce a prediction $\hat{Y}$ to match the true label Y whose shape depends on the task. GNN models are effective in graph-structured data because they involve trainable aggregation steps that pass the information from each node to its neighboring nodes and then apply nonlinear activation. The aggregation step at iteration t can be written as follows:

$$X^{(t+1)} = \phi(W^{(t)}X^{(t)}A + \Omega^{(t)}U), \tag{1}$$

where $X^{(t)} \in \mathbb{R}^{m\times n}$ stacks the state vectors of nodes in time step t into a matrix, in which the state vector for node i is denoted as $x^{(t)} \in \mathbb{R}^m$; $W^{(t)}$ and $\Omega^{(t)}$ are trainable weight matrices; ϕ is an activation function. The state vectors $X^{(T)}$ at final iteration can be used as the representation for nodes that combine input features and graph spatial information. The prediction from the GNN models is given by $\hat{Y} = f_\Theta(X^{(T)})$, where f_Θ is some trainable function parameterized by Θ. In practice, a linear f_Θ is often satisfactory.

Modern GNN approaches adopt different forms of graph convolution aggregation (1). Convolutional GNNs (Wu et al., 2019b) iterate (1) with $\Omega = 0$ and set $X^{(0)} = U$. Some works temper with the adjacency matrix using renormalization (Kipf and Welling, 2016) or attention (Veličković et al., 2017)). While recurrent GNNs use explicit input from features at each step with tied weights W and Ω, some methods replace the term ΩU with $\Omega_1 UA + \Omega_2 U$, in order to account for feature information from neighboring nodes (Dai et al., 2018). Our framework adopts a similar recurrent graph convolutional aggregation idea.

A heterogeneous network is an extended type of graph that contains different types of relations between nodes instead of only one type of edge. We continue to use $G = (V, \mathcal{E})$ to represent a heterogeneous network with the node set V and the edge set $\mathcal{E} \subseteq V \times V \times R$, where R is a set of $N := |R|$ relation types. Similarly, we define the adjacency matrices A_i, where A_i is the adjacency matrix for relation type $i \in R$. Some heterogeneous networks also have relation-specific feature matrices U_i.

Notation. For a matrix $V \in \mathbb{R}^{p\times q}$, $|V|$ denotes its absolute value (*i.e.* $|V|_{ij} = |V_{ij}|$). The infinity norm, or the max-row-sum norm, writes $\|V\|_\infty$. The 1-norm, or the max-column-sum norm, is denoted as $\|V\|_1 = \|V^\top\|_\infty$. The 2-norm is shown as $\|V\|$ or $\|V\|_2$. We use $\otimes$ to represent the Kronecker product, $\langle\cdot,\cdot\rangle$ to represent inner product and use $\odot$ to represent component-wise multiplication between two matrices of the same shape. For a $p \times q$ matrix V, $\mathbf{vec}(V) \in \mathbb{R}^{pq}$ represents the vectorized form of V, obtained by stacking its columns (See Section A for details). According to the Perron-Frobenius theory (Berman and Plemmons, 1994), every squared non-negative matrix M has a real non-negative eigenvalue that gives the largest modulus among all eigenvalues of M. This non-negative eigenvalue of M is called the *Perron-Frobenius (PF) eigenvalue* and denoted by $\lambda_{\text{pf}}(M)$ throughout the paper.

4 Implicit Graph Neural Networks

We now introduce a framework for graph neural networks called **I**mplicit **G**raph **N**eural **N**etworks (IGNN), which obtains a node representation through the fixed-point solution of a non-linear "equilibrium" equation. The IGNN model is formally described by

$$\hat{Y} = f_\Theta(X), \tag{2a}$$

$$X = \phi(WXA + b_\Omega(U)). \tag{2b}$$

In equation (2), the input feature matrix $U \in \mathbb{R}^{p\times n}$ is passed through some affine transformation $b_\Omega(\cdot)$ parametrized by Ω (*i.e.* a linear transformation possibly offset by some bias). The representation, given as the "internal state" $X \in \mathbb{R}^{m\times n}$ in the rest of the paper, is obtained as the fixed-point solution of the equilibrium equation (2b), where ϕ preserves the same shape of input and output. The prediction rule (2a) computes the prediction $\hat{Y}$ by feeding the state X through the output function f_Θ. In practice, a linear map $f_\Theta(X) = \Theta X$ may be satisfactory.

Unlike most existing methods that iterate (1) for a finite number of steps, an IGNN seeks the fixed point of equation (2b) that is trained to give the desired representation for the task. Evaluation of fixed point can be regarded as iterating (1) for an infinite number of times to achieve a steady state. Thus, the final representation potentially contains information from all neighbors in the graph. In practice, this gives a better performance over the finite iterating variants by capturing the long-range dependency in the graph. Another notable benefit of the framework is that it is memory-efficient in the sense that it only maintains one current state X without other intermediate representations.

Despite its notational simplicity, the IGNN model covers a wide range of variants, including their multi-layer formulations by stacking multiple equilibrium equations similar to (2b). The SSE (Dai et al., 2018) and FDGNN (Gallicchio and Micheli, 2019) models also fit within the IGNN formulation.

4.1 Examples of IGNN

In this section we introduce some examples of the variation of IGNN.

Multi-layer Setup. It is straight forward to extend IGNN to a multi-layer setup with several sets of W and Ω parameters for each layer. For conciseness, we use the ordinary graph setting. By treating the fixed-point solution X_{l-1} of the $(l-1)$-th layer as the input U_l to the l-th layer of equilibrium equation, a multi-layer formulation of IGNN with a total of L layers is created.

$$\begin{aligned} \hat{Y} &= f_\Theta(X_L), \\ X_L &= \phi_L(W_L X_L A + b_{\Omega_L}(X_{L-1})), \\ &\vdots \\ X_l &= \phi_l(W_l X_l A + b_{\Omega_l}(X_{l-1})), \\ &\vdots \\ X_1 &= \phi_1(W_1 X_1 A + b_{\Omega_1}(U)), \end{aligned} \tag{3}$$

where $\phi_1, \ldots, \phi_L$ are activation functions. We usually assume that CONE property holds on them. And (W_l, Ω_l) is the set of weights for the l-th layer. Thus the multi-layer formulation (3) with parameters $(W_l, l = 1, \ldots, L,\ A)$ is well-posed (*i.e.* gives unique prediction $\hat{Y}$ for any input U) when (W_l, A) is well-posed for ϕ_l for any layer l. This is true since the well-posedness for a layer guarantees valid input for the next layer. Since all layers are well-posed, the formulation will give unique final output for any input of compatible shape. FDGNN (Gallicchio and Micheli, 2019) uses a similar multi-layer formulation for graph classification but is only partially trained in practive.

In terms of the affine input function, $b_\Omega(U) = \Omega U A$ is a good choice. We show that the multi-layer IGNN with such b_Ω is equivalent to a single layer IGNN (2) with higher dimensions, the same A matrix and f_Θ function. The new activation map is given by $\phi = (\phi_L, \ldots, \phi_l, \ldots, \phi_1)$. Although ϕ is written in a block-wise form, they still operate on entry level and remain non-expansive. Thus the well-posedness results still hold. The new $\tilde{W}$ and $\tilde{b_\Omega}$ write,

$$\tilde{W} = \begin{pmatrix} W_L & \Omega_L & & \\ & \ddots & \ddots & \\ & & \ddots & \Omega_2 \\ & & & W_1 \end{pmatrix}, \quad \tilde{b_\Omega}(U) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ \Omega_1 \end{pmatrix} UA. \tag{4}$$

Special Cases. Many existing GNN formulations including convolutional and recurrent GNNs can be treated as special cases of IGNN. We start by showing that GCN (Kipf and Welling, 2016), a typical example of convolutional GNNs, is indeed an IGNN. We give the matrix representation of a 2-layer GCN as follows,

$$\begin{aligned} \hat{Y} &= W_2 X_1 A, \\ X_1 &= \phi_1(W_1 U A), \end{aligned} \tag{5}$$

where A is the renormalized adjacency matrix; W_1 and W_2 are weight parameters; ϕ_1 is a CONE activation map for the first layer; and X_1 is the hidden representation of first layer. We show that

GCN (5) is in fact a special case of IGNN by constructing an equivalent single layer IGNN (2) with the same A matrix.

$$\hat{Y} = \tilde{f_\Theta}(\tilde{X}), \tag{6a}$$
$$\tilde{X} = \phi(\tilde{W}\tilde{X}A + \tilde{b_\Omega}(U)). \tag{6b}$$

The new state $\tilde{X} = (X_2, X_1)$. The new activation map is given by $\phi = (\phi_1, \mathbb{I})$, where $\mathbb{I}$ represents an identity map. And the new $\tilde{W}$, $\tilde{b_\Omega}$, and $\tilde{f_\Theta}(\tilde{X})$ are,

$$\tilde{W} = \begin{pmatrix} 0 & W_2 \\ 0 & 0 \end{pmatrix}, \quad \tilde{b_\Omega}(U) = \begin{pmatrix} 0 \\ W_1 \end{pmatrix} UA, \quad \tilde{f_\Theta}(\tilde{X}) = \begin{pmatrix} I \\ 0 \end{pmatrix} \tilde{X}. \tag{7}$$

This reformulation of single layer IGNN also extends to multi-layer GCNs with more than 2 layers as well as other convolutional GNNs. Note that the new $\tilde{W}$ for the equivalent single layer IGNN is always strictly upper triangular. Thus $|\tilde{W}|$ has only 0 eigenvalue. As a result, $\lambda_{\text{pf}}(|A^\top \otimes W|) = \lambda_{\text{pf}}(A)\lambda_{\text{pf}}(|W|) = 0$ and the sufficient condition for well-posedness is always satisfied.

Another interesting special case is SSE (Dai et al., 2018), an example of recurrent GNN, that is given by

$$\begin{aligned} \hat{Y} &= W_2 X, \\ X &= \phi(W_{1r} W_2 X A + W_{1u} U A + W'_{1u} U), \end{aligned} \tag{8}$$

which can be easily converted into a single layer IGNN with the same A matrix and CONE activation ϕ. The new $\tilde{W}$, $\tilde{b_\Omega}$, and $\tilde{f_\Theta}(X)$ are,

$$\tilde{W} = W_{1r} W_2, \quad \tilde{b_\Omega}(U) = W_{1u} U A + W'_{1u} U, \quad \tilde{f_\Theta}(X) = W_2 X. \tag{9}$$

IGNN models can generalize to heterogeneous networks with different adjacency matrices A_i and input features U_i for different relations. In that case, we have the parameters W_i and Ω_i for each relation type $i \in R$ to capture the heterogenerity of the graph. A new equilibrium equation (10) is used:

$$X = \phi\left(\sum_i (W_i X A_i + b_{\Omega_i}(U_i))\right). \tag{10}$$

In general, there may not exist a unique solution for the equilibrium equation (2b) and (10). Thus, the notion of well-posedness comes into play.

4.2 Well-posedness of IGNNs

For the IGNN model to produce a valid representation, we need to obtain some *unique* internal state $X(U)$ given any input U from equation (2b) for ordinary graph settings or equation (10) for heterogeneous network settings. However, the equilibrium equation (2b) and (10) can have no well-defined solution X given some input U. We give a simple example in the scalar setting below, where the solution to the equilibrium equation (2b) does not even exist.

4.2.1 A Scalar Example

Consider the following scalar equilibrium equation (11),

$$x = \text{ReLU}(wxa + u), \tag{11}$$

where $x, w, a, u \in \mathbb{R}$ and $\text{ReLU}(\cdot) = \max(\cdot, 0)$ is the rectified linear unit. If we set $w = a = 1$, the equation (11) will have no solutions for any $u > 0$. See Figure 1 for the example with $u = 1$.

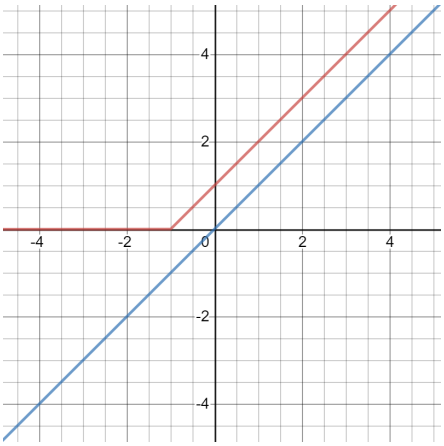


Figure 1: Plots of x (red plot) and $\text{ReLU}(wxa + u) = \text{ReLU}(x + 1)$ with $w = a = u = 1$ (blue plot). The two plots will intersect at some point whenever a solution exists. However in this case the two plots have no intersections, meaning that there is no solution to equation (11).

In order to ensure the existence and uniqueness of the solution to equation (2b) and (10), we define the notion of *well-posedness* for equilibrium equations with activation ϕ for both ordinary graphs and hetergeneous networks. This notion has been introduced in (El Ghaoui et al., 2020) for ordinary implicit models.

Definition 1 (Well-posedness on ordinary graphs)**.** *The tuple (W, A) of the weight matrix $W \in \mathbb{R}^{m\times m}$ and the adjacency matrix $A \in \mathbb{R}^{n\times n}$ is said to be well-posed for ϕ if for any $B \in \mathbb{R}^{m\times n}$, the solution $X \in \mathbb{R}^{m\times n}$ of the following equation*

$$X = \phi(WXA + B) \tag{12}$$

exists and is unique.

Definition 2 (Well-posedness on heterogeneous networks)**.** *The tuple $(W_i, A_i, i = 1, \ldots, N)$ of the weight matrices $W_i \in \mathbb{R}^{m\times m}$ and the adjacency matrices $A_i \in \mathbb{R}^{n\times n}$ is said to be well-posed for ϕ if for any $B_i \in \mathbb{R}^{m\times n}$, the solution $X \in \mathbb{R}^{m\times n}$ of the following equation*

$$X = \phi\left(\sum_{i=1}^{N}(W_i X A_i + B_i)\right) \tag{13}$$

exists and is unique.

We first develop sufficient conditions for the well-posedness property to hold on ordinary graph settings with a single edge type. The idea is to limit the structure of W and A together to ensure well-posedness for a set of activation ϕ.

In the following analysis, we assume that ϕ is component-wise non-expansive, which we refer to as the component-wise non-expansive (CONE) property. Most activation functions in deep learning satisfy the CONE property (*e.g.* Sigmoid, tanh, ReLU, Leaky ReLU, *etc.*). For simplicity, we assume that ϕ is differentiable.

We can now establish the following sufficient condition on (W, A) for our model with a CONE activation to be well-posed. Our result hinges on the notion of Perron-Frobenius (PF) eigenvalue $\lambda_{\text{pf}}(M)$ for a non-negative matrix M, as well as the notion of Kronecker product $A \otimes B \in \mathbb{R}^{pm \times qn}$ between two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$. See Appendix A for details.

Theorem 1 (PF sufficient condition for well-posedness on ordinary graphs)**.** *Assume that ϕ is a component-wise non-expansive (CONE) activation map. Then, (W, A) is well-posed for any such ϕ if $\lambda_{\text{pf}}(|A^\top \otimes W|) < 1$. Moreover, the solution X of equation (12) can be obtained by iterating equation (12).*

Proof. Recall that for any three matrices A, W, X of compatible sizes, we have $(A^\top \otimes W)\,\mathbf{vec}(X) = \mathbf{vec}(WXA)$ (Schacke, 2018). Showing equation (12) has an unique solution is equivalent to showing that the following "vectorized" equation has a unique solution:

$$\mathbf{vec}(X) = \phi(A^\top \otimes W\,\mathbf{vec}(X) + \mathbf{vec}(B))$$

It follows directly from Lemma 1 that if $\lambda_{\text{pf}}(|A^\top \otimes W|) = \lambda_{\text{pf}}(A)\lambda_{\text{pf}}(|W|) < 1$, then the above equation has unique solution that can be obtained by iterating the equation. ■

Lemma 1. *If ϕ is component-wise non-negative (CONE), M is some squared matrix and v is any real vector of compatible shape, the equation $x = \phi(Mx + v)$ has a unique solution if $\lambda_{\text{pf}}(|M|) < 1$. And the solution can be obtained by iterating the equation. Hence, $x = \lim_{t\to\infty} x_t$.*

$$x_{t+1} = \phi(Mx_t + v),\ x_0 = 0,\ t = 0, 1, \ldots \tag{14}$$

Proof. For existence, since ϕ is component-wise and non-expansive, we have that for $t \geq 1$ and the sequence $x_0, x_1, x_2, \ldots$ generated from iteration (14),

$$|x_{t+1} - x_t| = |\phi(Mx_t + v) - \phi(Mx_{t-1} + v)| \leq |M(x_t - x_{t-1})| \leq |M||x_t - x_{t-1}|.$$

For $n > m \geq 1$, the following inequality follows,

$$|x_n - x_m| \leq |M|^m \sum_{i=0}^{n-m-1} |M|^i |x_1 - x_0| \leq |M|^m \sum_{i=0}^{\infty} |M|^i |x_1 - x_0| \leq |M|^m w, \tag{15}$$

where

$$w := \sum_{i=0}^{\infty} |M|^i |x_1 - x_0| = (I - |M|)^{-1} |x_1 - x_0|.$$

Because $\lambda_{\text{pf}}(|M|) < 1$, the inverse of $I - |M|$ exists. It also follows that $\lim_{t\to\infty} |M|^t = 0$. From inequality (15), we show that the sequence $x_0, x_1, x_2, \ldots$ is a Cauchy sequence because

$0 \leq \lim_{m\to\infty} |x_n - x_m| \leq \lim_{m\to\infty} |M|^m w = 0$. And thus the sequence converges to some solution of $x = \phi(Mx + v)$.

For uniqueness, suppose both x_a and x_b satisfy $x = \phi(Mx + v)$, then the following inequality holds,

$$0 \leq |x_a - x_b| \leq |M||x_a - x_b| \leq \lim_{t\to\infty} |M|^t |x_a - x_b| = 0.$$

It follows that $x_a = x_b$ and there exists unique solution to $x = \phi(Mx + v)$. ■

We find Theorem 1 so general that many familiar and interesting results will follow from it, as discussed in the following remarks.

Remark 1. *For some non-negative adjacency matrix A, and arbitrary real parameter matrix W, $\lambda_{\text{pf}}(|A^\top \otimes W|) = \lambda_{\text{pf}}(A^\top \otimes |W|) = \lambda_{\text{pf}}(A)\lambda_{\text{pf}}(|W|)$.*

The final equality of the above remark follows from the fact that, the spectrum of the Kronecker product of matrix A and B satisfies that $\Delta(A \otimes B) = \{\mu\lambda : \mu \in \Delta(A),\ \lambda \in \Delta(B)\}$, where $\Delta(A)$ represents the spectrum of matrix A. And that, the left and right eigenvalues of a matrix are the same.

Remark 2 (Contraction sufficient condition for well-posedness (Gori et al., 2005))**.** *For any component-wise non-expansive (CONE) ϕ, if $\mathcal{A}(X) = \phi(WXA + B)$ is a contraction of X (w.r.t. vectorized norms), then (W, A) is well-posed for ϕ.*

The above remark follows from the fact that the contraction condition for any CONE activation map is equivalent to $\|A^\top \otimes W\| < 1$, which implies $\lambda_{\text{pf}}(|A^\top \otimes W|) < 1$.

Remark 3 (Well-posedness for directed acyclic graph)**.** *For a directed acyclic graph (DAG), let A be its adjacency matrix. For any real squared W, it holds that (W, A) is well-posed for every CONE activation map. Note that $\mathcal{A}(X) = \phi(WXA + B)$ need not be a contraction of X.*

Note that for DAG, A is nilpotent ($\lambda_{\text{pf}}(A) = 0$) and thus $\lambda_{\text{pf}}(|A^\top \otimes W|) = \lambda_{\text{pf}}(A)\lambda_{\text{pf}}(|W|) = 0$.

Remark 4 (Sufficient well-posedness condition for k-regular graph (Gallicchio and Micheli, 2019))**.** *For a k-regular graph, let A be its adjacency matrix. (W, A) is well-posed for every CONE activation map if $k\|W\|_2 < 1$.*

It follows from that for a k-regular graph, the PF eigenvalue of the adjacency matrix $\lambda_{\text{pf}}(A) = k$. And $\lambda_{\text{pf}}(A)\lambda_{\text{pf}}(|W|) \leq k\|W\|_2 < 1$ guarantees well-posedness.

A similar sufficient condition for well-posedness holds for heterogeneous networks.

Theorem 2 (PF sufficient condition for well-posedness on heterogeneous networks)**.** *Assume that ϕ is some component-wise non-expansive (CONE) activation map. Then, $(W_i, A_i,\ i = 1, \ldots, N)$ is well-posed for any such ϕ if $\lambda_{\text{pf}}\left(\sum_{i=1}^N |A_i^\top \otimes W_i|\right) < 1$. Moreover, the solution X of equation (13) can be obtained by iterating equation (13).*

Proof. Similarly, we can rewrite equation (13) into the following "vectorized" form.

$$\mathbf{vec}(X) = \phi\left(\sum_{i=1}^N (A_i^\top \otimes W_i)\,\mathbf{vec}(X) + \sum_{i=1}^N \mathbf{vec}(B_i)\right)$$

It follows from a similar scheme as the proof of Lemma 1 that if $\lambda_{\text{pf}}\left(\sum_{i=1}^N |A_i^\top \otimes W_i|\right) < 1$, the above equation has unique solution which can be obtained by iterating the equation. ■

Sufficient conditions in Theorems 1 and 2 guarantee convergence when iterating aggregation step to evaluate state X. Furthermore, these procedures enjoy exponential convergence in practice.

4.3 Tractable Well-posedness Condition for Training

At training time, however, it is difficult in general to ensure satisfaction of the PF sufficient condition $\lambda_{\mathrm{pf}}(|W|)\lambda_{\mathrm{pf}}(A) < 1$, because $\lambda_{\mathrm{pf}}(|W|)$ is non-convex in W. To alleviate the problem, we give a numerically tractable convex condition for well-posedness that can be enforced at training time efficiently through projection. Instead of using $\lambda_{\mathrm{pf}}(|W|) < \lambda_{\mathrm{pf}}(A)^{-1}$, we enforce the stricter condition $\|W\|_\infty < \lambda_{\mathrm{pf}}(A)^{-1}$, which guarantees the former inequality by $\lambda_{\mathrm{pf}}(|W|) \le \|W\|_\infty$. Although $\|W\|_\infty < \lambda_{\mathrm{pf}}(A)^{-1}$ is a stricter condition, we show in the following theorem that it is equivalent to the PF condition for positively homogeneous activation functions, (*i.e.* $\phi(\alpha x) = \alpha\phi(x)$ for any $\alpha \ge 0$ and x), in the sense that one can use the former condition at training without loss of generality.

Theorem 3 (Rescaled IGNN)**.** *Assume that ϕ is CONE and positively homogeneous. For an IGNN $(f_\Theta, W, A, b_\Omega, \phi)$ where (W, A) satisfies the PF sufficient condition for well-posedness, namely $\lambda_{\mathrm{pf}}(|W|) < \lambda_{\mathrm{pf}}(A)^{-1}$, there exists a linearly-rescaled equivalent IGNN $(\tilde{f_\Theta}, W', A, \tilde{b_\Omega}, \phi)$ with $\|W'\|_\infty < \lambda_{\mathrm{pf}}(A)^{-1}$ that gives the same output $\hat{Y}$ as the original IGNN for any input U.*

Proof. Similarly, we can rewrite equation (13) into the following "vectorized" form.

$$\mathbf{vec}(X) = \phi\left(\sum_{i=1}^{N}(A_i^\top \otimes W_i)\,\mathbf{vec}(X) + \sum_{i=1}^{N}\mathbf{vec}(B_i)\right)$$

It follows from a similar scheme as the proof of Lemma 1 that if $\lambda_{\mathrm{pf}}\left(\sum_{i=1}^{N}|A_i^\top \otimes W_i|\right) < 1$, the above equation has unique solution which can be obtained by iterating the equation. ■

The above-mentioned condition can be enforced by selecting a $\kappa \in [0, 1)$ and projecting the updated W onto the convex constraint set $\mathcal{C} = \{W : \|W\|_\infty \le \kappa/\lambda_{\mathrm{pf}}(A)\}$.

For heterogeneous network settings, we recall the following:

Remark 5. *For any non-negative adjacency matrix A and arbitrary real parameter matrix W, it holds that $\|A^\top \otimes W\|_\infty = \|A^\top\|_\infty\|W\|_\infty = \|A\|_1\|W\|_\infty$.*

The above remark follows from the facts that, $\|\cdot\|_\infty$ (resp. $\|\cdot\|_1$) gives maximum row (resp. column) sum of the absolute values of a given matrix. And that, for some real matrices A and B, $\|A\otimes B\|_\infty = \max_{i,j}\left(\sum_{k,l}|A_{ik}B_{jl}|\right) = \max_{i,j}\left(\sum_k|A_{ik}|\ \sum_l|B_{jl}|\right) = \max_i\left(\sum_k|A_{ik}|\right)\ \max_j\left(\sum_l|B_{jl}|\right) = \|A\|_\infty\|B\|_\infty$.

Similar to the difficulty faced in the ordinary graph settings, ensuring the PF sufficient condition on heterogeneous networks is hard in general. We propose to enforce the following tractable condition that is convex in W_i's: $\sum_{i=1}^{N}\|A_i\|_1\|W_i\|_\infty \le \kappa < 1$, $\kappa \in [0, 1)$. Note that this condition implies $\left\|\sum_{i=1}^{N}A_i^\top \otimes W_i\right\|_\infty \le \kappa$, and thus $\lambda_{\mathrm{pf}}\left(\sum_{i=1}^{N}|A_i^\top \otimes W_i|\right) \le \kappa < 1$. The PF sufficient condition for well-posedness on heterogeneous networks is then guaranteed.

4.4 Training of IGNN

We start by giving the training problem (16), where a loss $\mathcal{L}(Y, \hat{Y})$ is minimized to match $\hat{Y}$ to Y and yet the tractable condition $\|W\|_\infty \leq \kappa/\lambda_{\text{pf}}(A)$ for well-posedness is enforced with $\kappa \in [0, 1)$:

$$\min_{\Theta, W, \Omega} \ \mathcal{L}(Y, f_\Theta(X)) \ : \ \ X = \phi(WXA + b_\Omega(U)), \ \ \|W\|_\infty \leq \kappa/\lambda_{\text{pf}}(A). \tag{16}$$

The problem can be solved by projected gradient descent (involving a projection to the well-posedness condition following a gradient step), where the gradient is obtained through an implicit differentiation scheme. From the chain rule, one can easily obtain $\nabla_\Theta \mathcal{L}$ for the parameter of f_Θ and $\nabla_X \mathcal{L}$ for the internal state X. In addition, we can write the gradient with respect to scalar $q \in W \cup \Omega$ as follows:

$$\nabla_q \mathcal{L} = \left\langle \frac{\partial \left(WXA + b_\Omega(U)\right)}{\partial q}, \nabla_Z \mathcal{L} \right\rangle, \tag{17}$$

where $Z = WXA + b_\Omega(U)$ *assuming fixed* X (see Section 4.5). Here, $\nabla_Z \mathcal{L}$ is given as a solution to the equilibrium equation

$$\nabla_Z \mathcal{L} = D \odot \left(W^\top \nabla_Z \mathcal{L} \ A^\top + \nabla_X \mathcal{L} \right), \tag{18}$$

where $D = \phi'(WXA + b_\Omega(U))$ and $\phi'(z) = d\phi(z)/dz$ refers to the element-wise derivative of the CONE map ϕ. Since ϕ is non-expansive, it is 1-Lipschitz (*i.e.* the absolute value of $d\phi(z)/dz$ is not greater than 1), the equilibrium equation (18) for gradient $\nabla_Z \mathcal{L}$ admits a *unique* solution by iterating (18) to convergence, if (W, A) is well-posed for any CONE activation ϕ. (Note that $D \odot (\cdot)$ can be seen as a CONE map with each entry of D having absolute value less than or equal to 1.) Again, $\nabla_Z \mathcal{L}$ can be efficiently obtained due to exponential convergence when iterating (18) in practice.

Once $\nabla_Z \mathcal{L}$ is obtained, we can use the chain rule (via autograd software) to easily compute $\nabla_W \mathcal{L}$, $\nabla_\Omega \mathcal{L}$, and possibly $\nabla_U \mathcal{L}$ when input U requires gradients (*e.g.* in cases of features learning or multi-layer formulation). The deriviation has a deep connection to the Implicit Function Theorm. See Section 4.5 for details.

Due to the norm constraint introduced for well-posedness, each update to W requires a projection step (See Section 4.2). The new W is given by $W^+ = \pi_{\mathcal{C}}(W) := \min_{\|M\|_\infty \leq \kappa/\lambda_{\text{pf}}(A)} \|M - W\|_F^2$, where $\pi_{\mathcal{C}}$ is the projection back onto $\mathcal{C} = \{\|W\|_\infty \leq \kappa/\lambda_{\text{pf}}(A)\}$. The projection is decomposible across the rows of W. Each sub-problem will be a projection onto an $\mathcal{L}_1$-ball for which efficient methods exist (Duchi et al., 2008). A similar projected gradient descent training scheme for heterogeneous network settings is detailed in Section 4.5. Note that the gradient method in SSE (Dai et al., 2018) uses a first-order approximated solution to equation (18). FDGNN (Gallicchio and Micheli, 2019) only updates Θ at training using gradient descent.

4.5 Implicit differentiation for IGNN

To compute gradient of $\mathcal{L}$ from the training problem (16) w.r.t. a scalar $q \in W \cup \Omega$, we can use chain rule. It follows that,

$$\nabla_q \mathcal{L} = \left\langle \frac{\partial X}{\partial q}, \nabla_X \mathcal{L} \right\rangle, \tag{19}$$

where $\nabla_X \mathcal{L}$ can be easily calculated through modern autograd frameworks. But $\frac{\partial X}{\partial q}$ is non-trivial to obtain because X is only implicitly defined. Fortunately, we can still leverage chain rule in this case by carefully taking the "implicitness" into account.

To avoid taking derivatives of matrices by matrices, we again introduce the vectorized representation $\mathbf{vec}(\cdot)$ of matrices. The vectorization of a matrix $X \in \mathbb{R}^{m\times n}$, denoted $\mathbf{vec}(X)$, is obtained by stacking the columns of X into one single column vector of dimension mn. For simplicity, we use $\vec{X} := \mathbf{vec}(X)$ and $\nabla_{\vec{X}} \mathcal{L} = \mathbf{vec}(\nabla_X \mathcal{L})$ as a short hand notation of vectorization.

$$\frac{\partial \vec{X}}{\partial q} = \frac{\partial \vec{X}}{\partial \vec{Z}} \cdot \frac{\partial \vec{Z}}{\partial q}, \tag{20}$$

where $Z = WXA + b_\Omega(U)$ ($\vec{Z} = (A^\top \otimes W)\vec{X} + \overrightarrow{b_\Omega(U)}$) *assuming fixed* X. Unlike X in equation (2b), Z is not implicitly defined and should only be considered as a closed evaluation of $Z = WXA + b_\Omega(U)$ assuming X doesn't change depending on Z. In some sense, the Z in equation (20) doesn't equal to $WXA + b_\Omega(U)$. However, the closeness property will greatly simplify the evaluation of $\frac{\partial \vec{Z}}{\partial q}$. It turns out that we can still employ chain rule in this case to calculate $\frac{\partial \vec{X}}{\partial \vec{Z}}$ for such Z by taking the change of X before hand into account as follows,

$$\frac{\partial \vec{X}}{\partial \vec{Z}} = \frac{\partial \phi(\vec{Z})}{\partial \vec{Z}} + \frac{\partial \phi\left((A^\top \otimes W)\vec{X} + \overrightarrow{b_\Omega(U)}\right)}{\partial \vec{X}} \cdot \frac{\partial \vec{X}}{\partial \vec{Z}}, \tag{21}$$

where the second term accounts for the change in X that was ignored in Z. Another way to view this calculation is to right multiply $\frac{\partial \vec{Z}}{\partial q}$ on both sides of equation (21), which gives the chain rule evaluation of $\frac{\partial \vec{X}}{\partial q}$ that takes the gradient flowing back to X into account:

$$\frac{\partial \vec{X}}{\partial q} = \frac{\partial \phi\left((A^\top \otimes W)\vec{X} + \overrightarrow{b_\Omega(U)}\right)}{\partial q} + \frac{\partial \phi\left((A^\top \otimes W)\vec{X} + \overrightarrow{b_\Omega(U)}\right)}{\partial \vec{X}} \cdot \frac{\partial \vec{X}}{\partial q}.$$

The equation (21) can be simplified as follows,

$$\begin{aligned} \frac{\partial \vec{X}}{\partial \vec{Z}} &= (I - J)^{-1}\tilde{D}, \\ J &= \frac{\partial \phi\left((A^\top \otimes W)\vec{X} + \overrightarrow{b_\Omega(U)}\right)}{\partial \vec{X}} = \tilde{D}(A^\top \otimes W), \end{aligned} \tag{22}$$

where $\tilde{D} = \frac{\partial \phi(\vec{Z})}{\partial \vec{Z}} = \mathbf{diag}\left(\phi'\left((A^\top \otimes W)\vec{X} + \overrightarrow{b_\Omega(U)}\right)\right)$. Now we can rewrite equation (19) as

$$\nabla_q \mathcal{L} = \left\langle \frac{\partial \vec{Z}}{\partial q}, \nabla_{\vec{Z}} \mathcal{L} \right\rangle, \tag{23}$$

$$\nabla_{\vec{Z}} \mathcal{L} = \left(\frac{\partial \vec{X}}{\partial \vec{Z}}\right)^\top \nabla_{\vec{X}} \mathcal{L}, \tag{24}$$

which is equivalent to equation (17). $\nabla_{\vec{Z}}\mathcal{L}$ should be interpreted as the direction of steepest change of $\mathcal{L}$ for $Z = WXA + b_\Omega(U)$ *assuming fixed* X. Plugging equation (21) to (24), we arrive at the following equilibrium equation (equivalent to equation (18))

$$\begin{aligned}\nabla_{\vec{Z}}\mathcal{L} &= \tilde{D}(A \otimes W^\top)\nabla_{\vec{Z}}\mathcal{L} + \tilde{D}\,\nabla_{\vec{X}}\mathcal{L},\\ \nabla_Z\mathcal{L} &= D \odot \left(W^\top \nabla_Z\mathcal{L} A^\top + \nabla_X\mathcal{L}\right),\end{aligned} \tag{25}$$

where $D = \phi'(WXA + b_\Omega(U))$. Interestingly, $\nabla_Z\mathcal{L}$ turns out to be given as a solution of an equilibrium equation particularly similar to equation (2b) in the IGNN "forward" pass. In fact, we can see element-wise multiplication with D as a CONE "activation map" $\tilde{\phi}(\cdot) = D \odot (\cdot)$. And it follows from Section 4.2 that if $\lambda_{\text{pf}}(W)\lambda_{\text{pf}}(A) < 1$, then $\lambda_{\text{pf}}(W^\top)\lambda_{\text{pf}}(A^\top) < 1$ and $\nabla_Z\mathcal{L}$ can be uniquely determined by iterating the above equation (25). Although the proof will be more involved, if (W, A) is well-posed for any CONE activation map, we can conclude that equilibrium equation (25) is also well-posed for $\tilde{\phi}$ where ϕ can be any CONE activation map.

Finally, by plugging the evaluated $\nabla_Z\mathcal{L}$ into equation (23), we get the desired gradients. Note that it is also possible to obtain gradient $\nabla_U\mathcal{L}$ by setting the q in the above calculation to be $q \in U$. This is valid because we have no restrictions on selection of q other than that it is not X, which is assumed fixed. Following the chain rule, we can give the closed form formula for $\nabla_W\mathcal{L}$, $\nabla_\omega\mathcal{L}, \omega \in \Omega$, and $\nabla_u\mathcal{L}, u \in U$.

$$\nabla_W\mathcal{L} = \nabla_Z\mathcal{L}\; A^\top X^\top, \quad \nabla_\omega\mathcal{L} = \left\langle \frac{\partial b_\Omega(U)}{\partial \omega}, \nabla_Z\mathcal{L} \right\rangle, \quad \nabla_u\mathcal{L} = \left\langle \frac{\partial b_\Omega(U)}{\partial u}, \nabla_Z\mathcal{L} \right\rangle.$$

Heterogeneous Network Setting We start by giving the training problem for heterogeneous networks similar to training problem (16) for ordinary graphs,

$$\begin{aligned}\min_{\Theta, W, \Omega}\;\; & \mathcal{L}(Y, f_\Theta(X))\\ \text{s.t.}\;\; & X = \phi\left(\sum_{i=1}^N (W_i X A_i + b_{\Omega_i}(U_i))\right),\\ & \sum_{i=1}^N \|A_i\|_1 \|W_i\|_\infty \le \kappa.\end{aligned} \tag{26}$$

The training problem can be solved again using projected gradient descent method where the gradient of W_i and Ω_i for $i \in R$ can be obtained with implicit differentiation. Using chain rule, we write the gradient of a scalar $q \in \bigcup_i (W_i \cup \Omega_i)$,

$$\nabla_q\mathcal{L} = \left\langle \frac{\partial\left(\sum_{i=1}^N (W_i X A_i + b_{\Omega_i}(U_i))\right)}{\partial q}, \nabla_Z\mathcal{L} \right\rangle, \tag{27}$$

where $Z = \sum_{i=1}^N (W_i X A_i + b_{\Omega_i}(U_i))$ and $\nabla_Z\mathcal{L}$ in equation (27) should be interpreted as "direction of fastest change of $\mathcal{L}$ for Z *assuming fixed* X". Similar to the derivation in ordinary graphs setting, such notion of $\nabla_Z\mathcal{L}$ enables convenient calculation of $\nabla_q\mathcal{L}$. And the vectorized gradient w.r.t. Z

can be expressed as a function of the vectorized gradient w.r.t. X:

$$\nabla_{\vec{Z}}\mathcal{L} = \left(\frac{\partial \vec{X}}{\partial \vec{Z}}\right)^{\top} \nabla_{\vec{X}}\mathcal{L} \tag{28}$$

$$\begin{aligned}\frac{\partial \vec{X}}{\partial \vec{Z}} &= \frac{\partial \phi(\vec{Z})}{\partial \vec{Z}} + \frac{\partial \phi\left(\sum_{i=1}^{N}\left((A_i^{\top}\otimes W_i)\vec{X} + \overrightarrow{b_{\Omega_i}(U_i)}\right)\right)}{\partial \vec{X}} \cdot \frac{\partial \vec{X}}{\partial \vec{Z}} \\ &= (I-J)^{-1}\tilde{D} \qquad (29) \\ J &= \frac{\partial \phi\left(\sum_{i=1}^{N}\left((A_i^{\top}\otimes W_i)\vec{X} + \overrightarrow{b_{\Omega_i}(U_i)}\right)\right)}{\partial \vec{X}} = \tilde{D}\sum_{i=1}^{N}(A_i^{\top}\otimes W_i),\end{aligned}$$

where $\tilde{D} = \frac{\partial \phi(\vec{Z})}{\partial \vec{Z}} = \mathbf{diag}\left(\phi'\left(\sum_{i=1}^{N}\left((A_i^{\top}\otimes W_i)\vec{X} + \overrightarrow{b_{\Omega_i}(U_i)}\right)\right)\right)$. Plugging the expression (29) into (28), we arrive at the following equilibrium equation for $\nabla_{\vec{Z}}\mathcal{L}$ and $\nabla_Z\mathcal{L}$,

$$\begin{aligned}\nabla_{\vec{Z}}\mathcal{L} &= \tilde{D}\sum_{i=1}^{N}(A_i \otimes W_i^{\top})\nabla_{\vec{Z}}\mathcal{L} + \tilde{D}\,\nabla_{\vec{X}}\mathcal{L} \\ \nabla_Z\mathcal{L} &= D \odot \left(\sum_{i=1}^{N}(W_i^{\top}\nabla_Z\mathcal{L}A_i^{\top}) + \nabla_X\mathcal{L}\right), \qquad (30)\end{aligned}$$

where $D = \phi'\left(\sum_{i=1}^{N}(W_iXA_i + b_{\Omega_i}(U_i))\right)$. Not surprisingly, the equilibrium equation (30) again appears to be similar to the equation (10) in the IGNN "forward" pass. We can also view element-wise multiplication with D as a CONE "activation map" $\tilde{\phi}(\cdot) = D \odot (\cdot)$. And it follows from Section 4.2 that if $\lambda_{\text{pf}}(|A^{\top}\otimes W|) < 1$, then $\lambda_{\text{pf}}(|A \otimes W^{\top}|) < 1$ and $\nabla_Z\mathcal{L}$ can be uniquely determined by iterating the above equation (25). It also holds that if $(W_i, A_i, i \in \{1,\ldots,N\})$ is well-posed for any CONE activation ϕ, then we can conclude that equilibrium equation (30) is also well-posed for $\tilde{\phi}$ where ϕ can be any CONE activation map.

Finally, by plugging the evaluated $\nabla_Z\mathcal{L}$ into equation (27), we get the desired gradients. It is also possible to obtain gradient $\nabla_{U_i}\mathcal{L}$ by setting the q in the above calculation to be $q \in \bigcup_i U_i$. This is valid because we have no restrictions on selection of q other than that it is not X, which is assumed fixed.

After the gradient step, the projection to the tractable condition mentioned in Section 4.3 can be done approximately by assigning κ_i for each relation $i \in R$ and projecting W_i onto $\mathcal{C}_i = \{\|W_i\|_\infty \leq \kappa_i/\|A\|_1\}$. Ensuring $\sum_i \kappa_i = \kappa < 1$ will guarantee that the PF condition for heterogeneous network is satisfied. However, empirically, setting $\kappa_i < 1$ with $\sum_i \kappa_i > 1$ in some cases is enough for the convergence property to hold for the equilibrium equations.

5 Numerical Experiment

In this section, we demonstrate the capability of IGNN on effectively learning a representation that captures the long-range dependency and therefore offers the state-of-the-art performance on both synthetic and real-world data sets. More specifically, we test IGNN against a selected set of baselines on 6 node classification data sets (Chains, PPI, AMAZON, ACM, IMDB, DBLP) and 5 graph

classification data sets (MUTAG, PTC, COX2, PROTEINS, NC11), where Chains is a synthetic data set; PPI and AMAZON are multi-label classification data sets; ACM, IMDB and DBLP are based on heterogeneous networks. We inherit the same experimental settings and reuse the results of baselines from literatures in some of the data sets. The test set performance is reported. Detailed description of the data sets, our preprocessing procedure, hyper-parameters, and other information of experiments can be found in Section 5.1.

Synthetic Chains Data Set. To evaluate GNN's capability for capturing the underlying long-range dependency in graphs, we create the Chains data set where the goal is to classify nodes in a chain of length l. The information of the class is only sparsely provided as the feature in an end node. We use a small training set, validation set, and test set with only 20, 100, and 200 nodes, respectively. For simplicity, we only consider the binary classification task. Four representative baselines are implemented and compared. We show in Figure 2 that IGNN and SSE (Dai et al., 2018) both capture the long-range dependency with IGNN offering a better performance for longer chains, while finite-iterating convolutional GNNs with $T = 2$, including GCN (Kipf and Welling, 2016), SGC (Wu et al., 2019a) and GAT (Veličković et al., 2017), fail to give meaningful predictions when the chains become longer. However, selecting a larger T for convolutional GNNs does not seem to help in this case of limited training data. We further discuss this aspect in Section 5.1.

Figure 2: Micro-F_1 (%) performance with respect to the length of the chains.

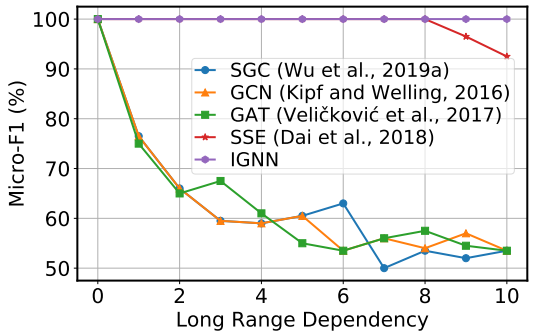


Table 1: Multi-label node classification Micro-F_1 (%) performance on PPI data set.

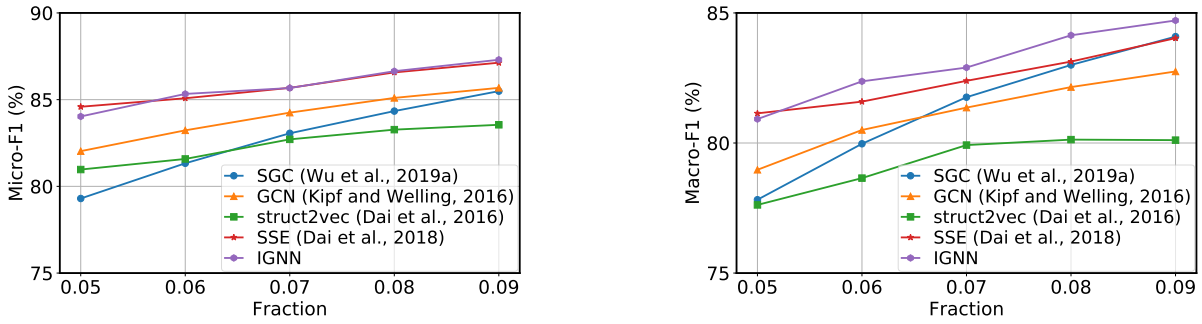
Method	Micro-F_1 /%
Multi-Layer Perceptron	46.2
GCN (Kipf and Welling, 2016)	59.2
GraphSAGE (Hamilton et al., 2017)	78.6
SSE (Dai et al., 2018)	83.6
GAT (Veličković et al., 2017)	97.3
IGNN	**97.6**

Node Classification. The popular benchmark data set Protein-Protein Interaction (PPI) models the interactions between proteins using a graph, with nodes being proteins and edges being interactions. Each protein can have at most 121 labels and be associated with additional 50-dimensional features. The train/valid/test split is consistent with GraphSage (Hamilton et al., 2017). We report the micro-averaged F_1 score of a multi-layer IGNN against other popular baseline models. The results can be found in Table 1. By capturing the underlying long-range dependency between proteins, the IGNN achieves the best performance compared to other baselines.

To further manifest the scalability of IGNN towards larger graphs, we conduct experiments on a large multi-label node classification data set, namely the Amazon product co-purchasing network data set (Yang and Leskovec, 2015) [2]. The data set renders products as nodes and co-purchases as edges but provides no input features. 58 product types with more than 5,000 products are selected

[2] http://snap.stanford.edu/data/#amazon

Figure 3: Micro/Macro-F_1 (%) performance on the multi-label node classification task with Amazon product co-purchasing network data set.



from a total of 75,149 product types. While holding out 10% of the total nodes as test set, we vary the training set fraction from 5% to 9% to be consistent with (Dai et al., 2018). The data set come with no input feature vectors and thus require feature learning at training. Both Micro-F_1 and Macro-F_1 are reported on the held-out test set, where we compare IGNN with a set of baselines consistent with those in the synthetic data set. However, we use struct2vec (Dai et al., 2016) as an alternative to GAT since GAT faces a severe out-of-memory issue in this task.

As shown in Figure 3, IGNN again outperforms the baselines in most cases, especially when the amount of supervision grows. When more labels are available, more high-quality feature vectors of the nodes are learned and this enables the discovery of more long-range dependency. This phenomenon is aligned with our observation that IGNN achieves a better performance when there is more long-range dependency in the underlying graph.

Graph Classification. Aside from node classification, we test IGNN on graph classification tasks. A total of 5 bioinformatics benchmarks are chosen: MUTAG, PTC, COX2, NCI1 and PROTEINS (Yanardag and Vishwanathan, 2015). See details of data sets in Section 5.1. Under the graph classification setting, we compare a multi-layer IGNN with a comprehensive set of baselines, including a variety of GNNs and a number of graph kernels. Following identical settings as (Yanardag and Vishwanathan, 2015; Xu et al., 2018), 10-fold cross-validation with LIB-SVM (Chang and Lin, 2011) is conducted. The average prediction accuracy and standard deviations are reported in Table 2. In this experiment, IGNN achieves the best performance in 4 out of 5 experiments given the competitive baselines. Such performance further validates IGNN's success in learning converging aggregation steps that capture long-range dependencies when generalized to unseen testing graphs.

Heterogeneous Networks. Following our theoretical analysis on heterogeneous networks, we investigate how IGNN takes advantage of heterogeneity on node classification tasks. Three benchmarks based on heterogeneous network are chosen, *i.e.*, ACM, IMDB and DBLP (Wang et al., 2019; Park et al., 2019). More information regarding the heterogeneous network data sets can be found in Section 5.1. Table 3 compares IGNN against a set of state-of-the-art GNN baselines for heterogeneous networks. The heterogeneous variant of IGNN continues to offer a competitive performance on all 3 data sets where IGNN gives the best performance in ACM and IMDB data sets. While on DBLP, IGNN underperforms DMGI but still outperforms other baselines by large margin. Good performance on heterogeneous networks demonstrates the flexibility of IGNN on

Table 2: Graph classification accuracy (%). Results are averaged (and std are computed) on the outer 10 folds.

Data sets	MUTAG	PTC	COX2	PROTEINS	NCI1
# graphs	188	344	467	1113	4110
Avg # nodes	17.9	25.5	41.2	39.1	29.8
DGCNN (Zhang et al., 2018)	85.8	58.6	–	75.5	74.4
DCNN (Atwood and Towsley, 2016)	67.0	56.6	–	61.3	62.6
GK (Shervashidze et al., 2009)	81.4 ± 1.7	55.7 ± 0.5	–	71.4 ± 0.3	62.5 ± 0.3
RW (Gärtner et al., 2003)	79.2 ± 2.1	55.9 ± 0.3	–	59.6 ± 0.1	–
PK (Neumann et al., 2016)	76.0 ± 2.7	59.5 ± 2.4	81.0 ± 0.2	73.7 ± 0.7	82.5 ± 0.5
WL (Shervashidze et al., 2011)	84.1 ± 1.9	58.0 ± 2.5	83.2 ± 0.2	74.7 ± 0.5	$\mathbf{84.5 \pm 0.5}$
FDGNN (Gallicchio and Micheli, 2019)	88.5 ± 3.8	63.4 ± 5.4	83.3 ± 2.9	76.8 ± 2.9	77.8 ± 1.6
GCN (Kipf and Welling, 2016)	85.6 ± 5.8	64.2 ± 4.3	–	76.0 ± 3.2	80.2 ± 2.0
GIN (Xu et al., 2018)	89.0 ± 6.0	63.7 ± 8.2	–	75.9 ± 3.8	82.7 ± 1.6
IGNN	$\mathbf{89.3 \pm 6.7}$	$\mathbf{70.1 \pm 5.6}$	$\mathbf{86.9 \pm 4.0}$	$\mathbf{77.7 \pm 3.4}$	80.5 ± 1.9

handling heterogeneous relationships.

Table 3: Node classification Micro/Macro-F_1 (%) performance on heterogeneous network data sets.

Data sets	ACM		IMDB		DBLP	
Metric	Micro-F_1	Macro-F_1	Micro-F_1	Macro-F_1	Micro-F_1	Macro-F_1
DGI (Veličković et al., 2018)	88.1	88.1	60.6	59.8	72.0	72.3
GCN/GAT	87.0	86.9	61.1	60.3	71.7	73.4
DeepWalk (Perozzi et al., 2014)	74.8	73.9	55.0	53.2	53.7	53.3
mGCN (Ma et al., 2019)	86.0	85.8	63.0	62.3	71.3	72.5
HAN (Wang et al., 2019)	87.9	87.8	60.7	59.9	70.8	71.6
DMGI (Park et al., 2019)	89.8	89.8	64.8	64.8	**76.6**	**77.1**
IGNN	**90.5**	**90.6**	**65.5**	**65.5**	73.8	75.1

5.1 More on Experiments

In this section, we give detailed information about the experiments we conduct.

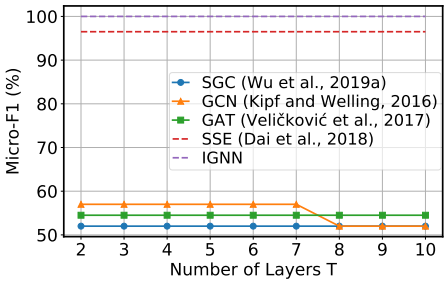
For preprocessing, we apply the *renormalization trick* consistent with GCN (Kipf and Welling, 2016) on the adjacent matrix of all data sets.

In terms of hyperparameters, unless otherwise specified, for IGNN, we use affine transformation $b_\Omega(U) = \Omega UA$; linear output function $f_\Theta(X) = \Theta X$; ReLU activation $\phi(\cdot) = \max(\cdot, 0)$; learning rate 0.01; dropout with parameter 0.5 before the output function; and $\kappa = 0.95$. We tune layers, hidden nodes, and κ through grid search. The hyperparameters for other baselines are consistent with that reported in their papers. Results with identical experimental settings are reused from previous works.

5.1.1 Synthetic Chains Data Set

We construct a synthetic node classification task to test the capability of models of learning to gather information from distant nodes. We consider the chains directed from one end to the other

Figure 4: Chains with $l = 9$. Traditional methods fail even with more iterations.



end with length l (*i.e.* $l+1$ nodes in the chain). For simplicity, we consider binary classification task with 2 types of chains. Information about the type is only encoded as 1/0 in first dimension of the feature (100d) on the starting end of the chain. The labels are provided as one-hot vectors (2d). In the data set we choose chain length $l = 9$ and 20 chains for each class with a total of 400 nodes. The training set consists of 20 data points randomly picked from these nodes in the total 40 chains. Respectively, the validation set and test set have 100 and 200 nodes.

A single-layer IGNN is implemented with 16 hidden unites and weight decay of parameter 5×10^{-4} for all chains data sets with different l. Four representative baselines are chosen: Stochastic Steady-state Embedding (**SSE**) (Dai et al., 2018), Graph Convolutional Network (**GCN**) (Kipf and Welling, 2016), Simple Graph Convolution (**SGC**) (Wu et al., 2019a) and Graph Attention Network (**GAT**) (Veličković et al., 2017). They all use the same hidden units and weight decay as IGNN. For (**GAT**), 8 head attention is used. For (**SSE**), we use the embedding directly as output and fix-point iteration $n_h = 8$, as suggested (Dai et al., 2018).

As mentioned in Section 5, convolutional GNNs with $T = 2$ cannot capture the dependency with a range larger than 2-hops. To see how convolutional GNNs capture the long-range dependency as T grows, we give an illustration of Micro-F_1 verses T for the selected baselines in Figure 4. From the experiment, we find that convolutional GNNs cannot capture the long-range dependency given larger T. This might be a result of the limited number of training nodes in this chain task. As T grows, convolutional GNNs experience an explosion of number of parameters to train. Thus the training data becomes insufficient for these models as the number of parameters increases.

5.1.2 Node Classification

For node classification task, we consider the applications under both transductive (Amazon) (Yang and Leskovec, 2015) and inductive (PPI) (Hamilton et al., 2017) settings. Transducive setting is where the model has access to the feature vectors of all nodes during training, while inductive setting is where the graphs for testing remain completely unobserved during training. The statistics of the data sets can be found in Table 4.

For experiments on Amazon, we construct a one-layer IGNN with 128 hidden units. No weight decay is utilized. The hyper parameters of baselines are consistent with (Yang and Leskovec, 2015; Dai et al., 2018).

For experiments on PPI, a five-layer IGNN model is applied for this multi-label classification tasks with hidden units as [1024, 512, 512, 256, 121] and $\kappa = 0.98$ for each layer. In addition, four MLPs are applied between the first four consecutive IGNN layers. We use the identity output

Table 4: The overview of data set statistics in node classification tasks.

Data set	# Nodes	# Edges	# Labels	Label type	Graph type
Amazon (transductive)	334,863	925,872	58	Product type	Co-purchasing
PPI (inductive)	56,944	818,716	121	Bio-states	Protein

function. Neither weight decay nor dropout is employed. We keep the experimental settings of baselines consistent with (Veličković et al., 2017; Dai et al., 2018; Kipf and Welling, 2016; Hamilton et al., 2017).

5.1.3 Graph Classification

For graph classification, 5 bioinformatics data sets are employed with information given in Table 2. We compare IGNN with a comprehensive set of baselines, including a variety of GNNs: Deep Graph Convolutional Neural Network (**DGCNN**) (Zhang et al., 2018), Diffusion-Convolutional Neural Networks (**DCNN**) (Atwood and Towsley, 2016), Fast and Deep Graph Neural Network (**FDGNN**) (Gallicchio and Micheli, 2019), **GCN** (Kipf and Welling, 2016) and Graph Isomorphism Network (**GIN**) (Xu et al., 2018), and a number of state-of-the-art graph kernels: Graphlet Kernel (**GK**) (Shervashidze et al., 2009), Random-walk Kernel (**RW**) (Gärtner et al., 2003), Propagation Kernel (**PK**) (Neumann et al., 2016) and Weisfeiler-Lehman Kernel (**WL**) (Shervashidze et al., 2011). We reuse the results from literatures (Xu et al., 2018; Gallicchio and Micheli, 2019) since the same experimental settings are maintained.

As of IGNN, a three-layer IGNN is constructed for comparison with the hidden units of each layer as 32 and $\kappa = 0.98$ for all layers. We use an MLP as the output function. Besides, batch normalization is applied on each hidden layer. Neither weight decay nor dropout is utilized.

5.1.4 Heterogeneous Networks

For heterogeneous networks, three data sets are chosen (ACM, IMDB, and DBLP). Consistent with previous works (Park et al., 2019), we use the the publicly available ACM data set (Wang et al., 2019), preprocessed DBLP and IMDB data sets (Park et al., 2019). For ACM and DBLP data sets, the nodes are papers and the aim is to classify the papers into three classes (Database, Wireless Communication, Data Mining), and four classes (DM, AI, CV, NLP)[3], respectively. For IMDB data set, the nodes are movies and we aim to classify these movies into three classes (Action, Comedy, Drama). The detailed information of data sets can be referred to Table 5. The preprocessing procedure and splitting method on three data sets keep consistent with (Park et al., 2019).

State-of-the-art baselines are selected for comparison with IGNN, including no-attribute network embedding: **DeepWalk** (Perozzi et al., 2014), attributed network embedding: **GCN**, **GAT** and **DGI** (Veličković et al., 2018), and attributed multiplex network embedding: **mGCN** (Ma et al., 2019), **HAN** (Wang et al., 2019) and **DMGI** (Park et al., 2019). Given the same experimental settings, we reuse the results of baselines from (Park et al., 2019).

A one-layer IGNN with hidden units as 64 is implemented on all data sets. Similar to **DMGI**, a weight decay of parameter 0.001 is used. For ACM, $\kappa = (0.55, 0.55)$ is used for Paper-Author and

[3]**DM**: KDD,WSDM,ICDM, **AI**: ICML,AAAI,IJCAI, **CV**: CVPR, **NLP**: ACL,NAACL,EMNLP

Table 5: Statistics of the data sets for heterogeneous graphs (Park et al., 2019). The node attributes are bag-of-words of text. Num. labeled data denotes the number of nodes involved during training.

	Relations (A-B)	Num. A	Num. B	Num. A-B	Relation type	Num. relations	Num. node attributes	Num. labeled data	Num. classes
ACM	Paper-Author	3,025	5,835	9,744	P-A-P	29,281	1,830 (Paper abstract)	600	3
	Paper-Subject	3,025	56	3,025	P-S-P	2,210,761			
IMDB	Movie-Actor	3,550	4,441	10,650	M-A-M	66,428	1,007 (Movie plot)	300	3
	Movie-Director	3,550	1,726	3,550	M-D-M	13,788			
DBLP	Paper-Author	7,907	1,960	14,238	P-A-P	144,783	2,000 (Paper abstract)	80	4
	Paper-Paper	7,907	7,907	10,522	P-P-P	90,145			
	Author-Term	1,960	1,975	57,269	P-A-T-A-P	57,137,515			

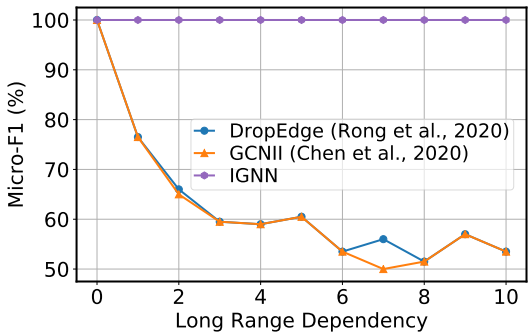
Paper-Subject relations. For IMDB, we select $\kappa = (0.5, 0.5)$ for Movie-Actor and Movie-Director relations. For DBLP, $\kappa = (0.7, 0.4)$ is employed for Paper-Author and Paper-Paper relations. As mentioned in Section 4.5, in practice, the convergence property can still hold when $\sum_i \kappa_i > 1$.

5.1.5 Over-smoothness

Convolutional GNNs has suffered from over-smoothness when the model gets deep. An interesting question to ask is whether IGNN suffers from the same issue and experience performance degradation in capturing long-range dependency with its "infinitively deep" GNN design.

In an effort to answer this question, we compared IGNN against two latest convolutional GNN models that solve the over-smoothness issue, GCNII Chen et al. (2020) and DropEdge Rong et al. (2020). We use the same experimental setting as the Chains experiment in section 5. Both GCNII and DropEdge are implemented with 10-layer and is compared with IGNN in capturing long-range dependency. The result is reported in Figure 5. We observe that IGNN consistently outperforms both GCNII and DropEdge as the chains gets longer. The empirical result suggest little suffering from over-smoothness for recurrent GNNs.

Figure 5: Micro-F_1 (%) performance with respect to the length of the chains.



6 Conclusion

In this paper, we present the implicit graph neural network model, a framework of recurrent graph neural networks. We describe a sufficient condition for well-posedness based on the Perron-Frobenius theory and a projected gradient decent method for training. Similar to some other recurrent graph neural network models, implicit graph neural network captures the long-range dependency, but it carries the advantage further with a superior performance in a variety of tasks, through rigorous conditions for convergence and exact efficient gradient steps. More notably, the flexible framework extends to heterogeneous networks where it maintains its competitive performance.

Broader Impact

GNN models are widely used on applications involving graph-structured data, including computer vision, recommender systems, and biochemical strucature discovery. Backed by more rigorous mathematical arguments, our research improves the capability GNNs of capturing the long-range dependency and therefore boosts the performance on these applications.

The improvements of performance in the applications will give rise to a better user experience of products and new discoveries in other research fields. But like any other deep learning models, GNNs runs into the problem of interpretability. The trade-off between performance and interpretability has been a topic of discussion. On one hand, the performance from GNNs benefits the tasks. On the other hand, the lack of interpretability might make it hard to recognize underlying bias when applying such algorithm to a new data set. Recent works (Hardt et al., 2016) propose to address the fairness issue by enforcing the fairness constraints.

While our research focuses on performance by capturing the long-range dependency, like many other GNNs, it does not directly tackle the fairness and interpretability aspect. We would encourage further work on fairness and interpretability on GNNs. Another contribution of our research is on the analysis of heterogeneous networks, where the fairness on treatment of different relationships remains unexplored. The risk of discrimination in particular real-world context might require cautious handling when researchers develop models.

References

Amos, B., Jimenez, I., Sacks, J., Boots, B., and Kolter, J. Z. (2018). Differentiable mpc for end-to-end planning and control. In *Advances in Neural Information Processing Systems*, pages 8289–8300.

Atwood, J. and Towsley, D. (2016). Diffusion-convolutional neural networks. In *Advances in neural information processing systems*, pages 1993–2001.

Bai, S., Kolter, J. Z., and Koltun, V. (2019). Deep equilibrium models. In *Advances in Neural Information Processing Systems*, pages 688–699.

Berman, A. and Plemmons, R. J. (1994). *Nonnegative matrices in the mathematical sciences*. SIAM.

Chang, C.-C. and Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27.

Chang, H., Rong, Y., Xu, T., Huang, W., Sojoudi, S., Huang, J., and Zhu, W. (2020). Spectral graph attention network. *arXiv preprint arXiv:2003.07450*.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. (2020). Simple and deep graph convolutional networks. *arXiv preprint arXiv:2007.02133*.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583.

Dai, H., Dai, B., and Song, L. (2016). Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, pages 2702–2711.

Dai, H., Kozareva, Z., Dai, B., Smola, A., and Song, L. (2018). Learning steady-states of iterative algorithms over graphs. In *International conference on machine learning*, pages 1106–1114.

de Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J., and Kolter, J. Z. (2018). End-to-end differentiable physics for learning and control. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 7178–7189. Curran Associates, Inc.

Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. (2008). Efficient projections onto the l 1-ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279.

El Ghaoui, L., Travacca, B., Gu, F., Tsai, A. Y.-T., and Askari, A. (2020). Implicit deep learning. *arXiv preprint arXiv:1908.06315*.

Gallicchio, C. and Micheli, A. (2010). Graph echo state networks. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

Gallicchio, C. and Micheli, A. (2019). Fast and deep graph neural networks. *arXiv preprint arXiv:1911.08941*.

Gärtner, T., Flach, P., and Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*, pages 129–143. Springer.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org.

Gori, M., Monfardini, G., and Scarselli, F. (2005). A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE.

Gu, F., Chang, H., Zhu, W., Sojoudi, S., and El Ghaoui, L. (2020). Implicit graph neural networks. *Advances in Neural Information Processing Systems*, 33.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034.

Hardt, M., Price, E., and Srebro, N. (2016). Equality of opportunity in supervised learning. In *Advances in neural information processing systems*, pages 3315–3323.

Jin, M., Chang, H., Zhu, W., and Sojoudi, S. (2019). Power up! robust graph convolutional network against evasion attacks based on graph powering. *arXiv preprint arXiv:1905.10029*.

Kampffmeyer, M., Chen, Y., Liang, X., Wang, H., Zhang, Y., and Xing, E. P. (2018). Rethinking knowledge graph propagation for zero-shot learning. *arXiv preprint arXiv:1805.11724*.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Li, Q., Han, Z., and Wu, X. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI-18 AAAI Conference on Artificial Intelligence*, pages 3538–3545.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2015). Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

Ma, Y., Wang, S., Aggarwal, C. C., Yin, D., and Tang, J. (2019). Multi-dimensional graph convolutional networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 657–665. SIAM.

Neumann, M., Garnett, R., Bauckhage, C., and Kersting, K. (2016). Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102(2):209–245.

Newman, M. (2018). *Networks*. Oxford university press.

Oono, K. and Suzuki, T. (2020). Graph neural networks exponentially lose expressive power for node classification. In *ICLR 2020 : Eighth International Conference on Learning Representations*.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Park, C., Kim, D., Han, J., and Yu, H. (2019). Unsupervised attributed multiplex network embedding. *arXiv preprint arXiv:1911.06750*.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. (2020). Geom-gcn: Geometric graph convolutional networks. In *ICLR 2020 : Eighth International Conference on Learning Representations*.

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710.

Rong, Y., Huang, W., Xu, T., and Huang, J. (2020). Dropedge: Towards deep graph convolutional networks on node classification. In *ICLR 2020 : Eighth International Conference on Learning Representations*.

Schacke, K. (2018). On the kronecker product.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561.

Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. (2009). Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Veličković, P., Fedus, W., Hamilton, W. L., Liò, P., Bengio, Y., and Hjelm, R. D. (2018). Deep graph infomax. *arXiv preprint arXiv:1809.10341*.

Wan, F., Hong, L., Xiao, A., Jiang, T., and Zeng, J. (2019). Neodti: neural integration of neighbor information from a heterogeneous network for discovering new drug–target interactions. *Bioinformatics*, 35(1):104–111.

Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., and Yu, P. S. (2019). Heterogeneous graph attention network. In *The World Wide Web Conference*, pages 2022–2032.

Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. (2019a). Simplifying graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6861–6871. PMLR.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2019b). A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

Yanardag, P. and Vishwanathan, S. (2015). Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374.

Yang, J. and Leskovec, J. (2015). Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems*, 42(1):181–213.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 974–983.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Zhang, Z., Cui, P., and Zhu, W. (2020). Deep learning on graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*.

Zhao, L. and Akoglu, L. (2020). Pairnorm: Tackling oversmoothing in gnns. In *ICLR 2020 : Eighth International Conference on Learning Representations*.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2018). Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*.

A Kronecker Product

For two matrices A and B, the Kronecker product of $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$ is denoted as $A \otimes B \in \mathbb{R}^{pm \times qn}$:

$$A \otimes B = \begin{pmatrix} A_{11}B & \cdots & A_{1n}B \\ \vdots & \ddots & \vdots \\ A_{m1}B & \cdots & A_{mn}B \end{pmatrix}.$$

By definition of the Kronecker product, $(A \otimes B)^\top = A^\top \otimes B^\top$. Additionally, the following equality holds assuming compatible shapes, $(A^\top \otimes W)\,\mathbf{vec}(X) = \mathbf{vec}(WXA)$ (Schacke, 2018), where $\mathbf{vec}(X) \in \mathbb{R}^{mn}$ denotes the vectorization of matrix $X \in \mathbb{R}^{m \times n}$ by stacking the columns of X into a single column vector of dimension mn. Suppose $x_i \in \mathbb{R}^m$ is the i-th column of X, $\mathbf{vec}(X) = [x_1^\top, \ldots, x_n^\top]^\top$.

Leveraging the definition of Kronecker product and vectorization, the following equality holds, $(A^\top \otimes W)\,\mathbf{vec}(X) = \mathbf{vec}(WXA)$ (Schacke, 2018). Intuitively, this equality reshapes WXA which is linear in X into a more explicit form $(A^\top \otimes W)\,\mathbf{vec}(X)$ which is linear in $\mathbf{vec}(X)$, a flattened form of X. Through the transformation, we place WXA into the form of Mx. Thus, we can employ Lemma 1 to obtain the well-posedness conditions.