Implicit Graph Neural Networks: A Monotone Operator Viewpoint

Justin Baker* [1] **Qingsong Wang*** [1] **Cory Hauck** [2] **Bao Wang** [1]

Abstract

Implicit graph neural networks (IGNNs) – that solve a fixed-point equilibrium equation using Picard iteration for representation learning – have shown remarkable performance in learning long-range dependencies (LRD) in the underlying graphs. However, IGNNs suffer from several issues, including 1) their expressivity is limited by their parameterizations for the well-posedness guarantee, 2) IGNNs are unstable in learning LRD, and 3) IGNNs become computationally inefficient when learning LRD. In this paper, we provide a new well-posedness characterization for IGNNs leveraging monotone operator theory, resulting in a much more expressive parameterization than the existing one. We also propose an orthogonal parameterization for IGNN based on Cayley transform to stabilize learning LRD. Furthermore, we leverage Anderson-accelerated operator splitting schemes to efficiently solve for the fixed point of the equilibrium equation of IGNN with monotone or orthogonal parameterization. We verify the computational efficiency and accuracy of the new models over existing IGNNs on various graph learning tasks at both graph and node levels. Code is available at `https://github.com/Utah-Math-Data-Science/MIGNN`

1. Introduction

Implicit graph neural networks (IGNNs) – that solve a fixed-point equilibrium equation using Picard iteration for graph representation learning – can learn long-range dependencies (LRD) in the underlying graphs, showing remarkable performance for various tasks [49; 24; 39; 43; 15]. Let $G = (V, E)$ represent a graph, where V is the set of nodes, and $E \subseteq V \times V$ is the set of edges. The connectivity of G can be represented by the adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ with $A_{ij} = 1$ if there is an edge connecting nodes $i, j \in V$; otherwise $A_{ij} = 0$. Let $\boldsymbol{X} \in \mathbb{R}^{d\times n}$ be the initial node features whose i-th column $\boldsymbol{x}_i \in \mathbb{R}^d$ is the initial feature of the i-th node. IGNN [24] learns the node representation by finding the fixed point, denoted as $\boldsymbol{Z}^*$, of the Picard iteration below

$$\boldsymbol{Z}^{(k+1)} = \sigma\big(\boldsymbol{W}\boldsymbol{Z}^{(k)}\boldsymbol{G} + g_{\boldsymbol{B}}(\boldsymbol{X})\big), \ \text{ for } k = 0, 1, \cdots, \quad (1)$$

where σ is the nonlinearity (e.g. ReLU), $g_{\boldsymbol{B}}$ is a function parameterized by $\boldsymbol{B}$ (e.g. $g_{\boldsymbol{B}}(\boldsymbol{X}) = \boldsymbol{B}\boldsymbol{X}\boldsymbol{G}$), matrices $\boldsymbol{W}$ and $\boldsymbol{B} \in \mathbb{R}^{d\times d}$ are learnable weights, and $\boldsymbol{G}$ is a graph-related matrix. In IGNN, $\boldsymbol{G}$ is chosen as $\hat{\boldsymbol{A}} := \hat{\boldsymbol{D}}^{-1/2}(\boldsymbol{I} + \boldsymbol{A})\hat{\boldsymbol{D}}^{-1/2}$ with $\boldsymbol{I}$ being the identity matrix and $\hat{\boldsymbol{D}}$ is the degree matrix with $\hat{D}_{ii} = 1 + \sum_{j=1}^n A_{ij}$. The prediction of IGNN is given by $f_\Theta(\boldsymbol{Z}^*)$, a function parameterized by Θ. IGNNs have several merits: 1) The depth of IGNN is adaptive to particular data and tasks rather than fixed. 2) Training IGNNs requires constant memory independent of their depth – leveraging implicit differentiation [46; 1; 35; 12]. 3) IGNNs have better potential to capture LRD of the underlying graph compared to existing GNNs, including GCN [55], GAT [52], SSE [16], and SGC [59]. Moreover, the bias term used by IGNN helps to overcome the over-smoothing issue of deep GNNs, which has also been studied in [51]. Nevertheless, we notice that IGNN has limited expressivity for graph learning and is unstable and inefficient in learning LRD, and we provide details about these issues below.

Well-posedness of IGNN limits its expressivity. IGNN constrains the weight matrix $\boldsymbol{W}$ using a tractable projected gradient descent method to ensure the well-posedness of Picard iteration [24], *constraining the magnitude of $\boldsymbol{W}$'s eigenvalues to be less than one*; see Sec. 2 for details. This limits the selection of $\boldsymbol{W}$ and thereby limits the expressivity of IGNNs.

IGNN is unstable and inefficient in learning LRD. To understand when IGNN can learn LRD, we run IGNN using the settings in [24] to classify directed chains – a synthetic dataset designed to test the effectiveness of GNNs in learning LRD [24] and we discuss details of this dataset and task in Sec. 5.1. Fig. 1 plots epoch vs. accuracy of IGNN for the chain classification. Here, each epoch means iterating Equation (1) until convergence and then updating $\boldsymbol{W}$ and $\boldsymbol{B}$.

*Equal contribution [1]Department of Mathematics and Scientific Computing and Imaging Insitute, University of Utah. [2]Oak Ridge National Laboratory. Correspondence to: Bao Wang <wangbaonj@gmail.com>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

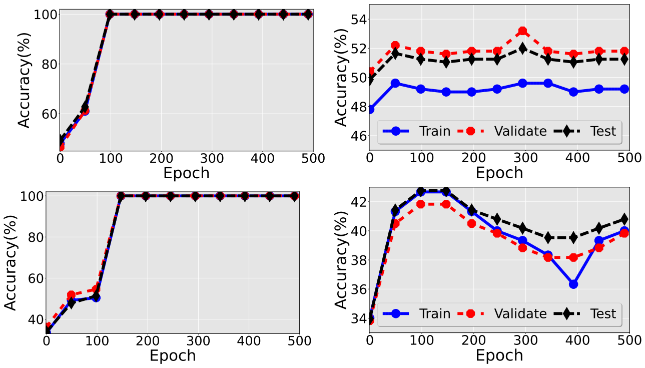


Figure 1. Epoch vs. training, validation, and test accuracy of IGNN for classifying directed chains. First row: binary chains of length 100 (left) and 250 (right). Second row: three-class chains of length 80 (left) and 100 (right). Notice that the training, validation, and test accuracy curves in the left panel are close to each other. The figures in the left panel use the same legend as the right ones.

IGNN can classify binary chains perfectly at length 100 but performs near-random guesses when the length is 250; see the first row of Fig. 1. For the three-class chains, IGNN's performance is very poor at chain length 100 but performs well at length 80; see the second row of Fig. 1.

We investigate the results in Fig. 1 by studying the dynamics of eigenvalues of the matrix $|\boldsymbol{W}|$[1]. For illustrative purpose, we consider $\lambda_1(|\boldsymbol{W}|)$ and $\lambda_2(|\boldsymbol{W}|)$, the largest and second largest eigenvalues of $|\boldsymbol{W}|$ in magnitude. Fig. 2 (left) contrasts the evolution of the magnitude of $\lambda_1(|\boldsymbol{W}|)$ and $\lambda_2(|\boldsymbol{W}|)$ of IGNN when classifying nodes on chains with different lengths. We see that the magnitude of both eigenvalues goes to 1 when IGNN becomes accurate. However, Fig. 2 (right) shows that IGNN takes many more iterations in each epoch when the magnitude of eigenvalues gets close to 1. Indeed, when $\lambda_1(|\boldsymbol{W}|) \to 1$, the Lipschitz constant of the linear map $\boldsymbol{W}\boldsymbol{Z}\boldsymbol{G} + g_{\boldsymbol{B}}(\boldsymbol{X})$ is close to 1, slowing down the convergence of the Picard iteration. The results in Fig. 2 echo our intuition: the representation of a given node aggregates one more hop of neighboring nodes' information after each Picard iteration; when the magnitude of eigenvalues gets close to 1, Equation (1) converges slowly so that IGNN can capture LRD before Picard iteration converges.

We report the classification results of different lengths in Appendix H; these results show prevalently that IGNNs suffer from two bottlenecks: 1) An inherent *tradeoff between computational efficiency and capability for learning LRD*. 2) The performance of IGNNs, based on Picard iteration, is *unstable* across tasks. In particular, starting from random Gaussian initialization of $\boldsymbol{W}$ – the default initialization – IGNN cannot learn LRD if none of the eigenvalues of $\boldsymbol{W}$ gets close to 1 in magnitude.

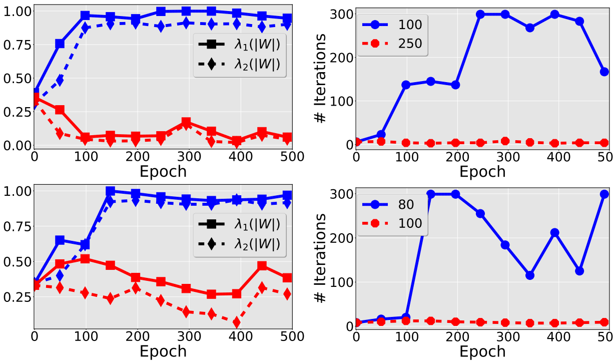


Figure 2. Epoch vs. the magnitude of $\lambda_1(|\boldsymbol{W}|)$ and $\lambda_2(|\boldsymbol{W}|)$ and the iterations required for each epoch. First row: binary chains, second row: three-class chains. In left and right plots, red and blue lines represent different chain lengths. On the left, we use solid and dashed lines to plot $\lambda_1(|\boldsymbol{W}|)$ and $\lambda_2(|\boldsymbol{W}|)$, respectively.

1.1. Our contribution

We develop expressive, stable, and computationally efficient monotone operator IGNN (MIGNN) [2]. To boost the expressivity of IGNN, we derive a new well-posedness condition for MIGNN based on an extension of the well-posedness condition of monotone operator equilibrium networks [57] to IGNNs; see Sec 2. The new well-posedness condition informs us to design a ***monotone parameterization*** of $\boldsymbol{W}$, whose eigenvalues can take a much wider range than that of IGNNs. To stabilize IGNN for learning LRD, we propose a Cayley transform-based ***orthogonal parameterization*** of $\boldsymbol{W}$ for MIGNN; see Sec. 3. Picard iteration is ***inefficient or impossible*** to find the fixed point of MIGNN with monotone or orthogonal parameterization. As such, we implement MIGNNs leveraging operator splitting schemes; see Sec. 4. We verify the efficacy of MIGNN on various benchmark tasks; see Sec. 5.

1.2. Additional related work

We briefly review some representative related works in three directions: 1) deep equilibrium models (DEQs), a.k.a. fixed-point networks, 2) implicit GNNs based on fixed-point networks, and 3) orthogonal parameterizations for recurrent neural networks (RNNs).

DEQ. IGNNs are related to DEQs [5; 17; 6], but the equilibrium equation of IGNN differs from DEQ's in that IGNN encodes graph structure. DEQs are a class of infinite depth weight-tied feedforward neural networks with forward propagation using root-finding and backpropagation using implicit differentiation. As a result, training DEQs only requires constant memory independent of the network's depth. Monotone operator theory has been used to guarantee the convergence of DEQs [57] and to improve the robustness of implicit networks [28]. The convergence of DEQs has also been considered by constraining the network's weights

[1] $|\boldsymbol{W}|$ is obtained by taking the entry-wise absolute value of $\boldsymbol{W}$.

[2] Starting from here, we use MIGNN to stress that the model is based on monotone operator theory.

[33]. Linearized DEQs are studied in [30]. Jacobian regularization has been used to stabilize the training of DEQs [7]. Anderson-accelerated DEQs with learned acceleration-related hyperparameters are also proposed [8].

Implicit GNNs. Several efforts devoted to advancing IGNNs are based on fixed-point networks: EIGNN removes the nonlinearity in each intermediate iteration and derives a closed form of the infinite iterations [39], convergent graph solver (CGS) is an IGNN model with convergence guarantees by constructing the input-dependent linear contracting iterative maps [43], GIND leverages implicit nonlinear diffusion to access infinite hops of neighbors [15].

Orthogonal parameterization for deep learning. The Picard iteration Equation (1) is related to the hidden state updates of RNNs [46; 18; 1; 34; 41]. Learning LRD is challenging for RNNs due to exploding and vanishing gradient during backpropagation through time [56; 10; 44; 41; 54; 42]. Enforcing orthogonal parameterization for RNNs is an effective approach to overcome exploding and vanishing gradients, benefiting RNNs for learning LRD [3; 58; 29; 53; 40; 25].

1.3. Notation

We denote scalars by lower- or upper-case letters and vectors/matrices with lower- or upper-case boldface letters. For a vector $\boldsymbol{a}$, we use $\|\boldsymbol{a}\|$ and $\|\boldsymbol{a}\|_\infty$ to denote its ℓ_2 and ℓ_∞ norms, respectively. We use $\boldsymbol{I}$ to denote the identity matrix whose dimension can be inferred from the context. For a matrix $\boldsymbol{A}$, we denote its transpose by $\boldsymbol{A}^\top$, its inverse by $\boldsymbol{A}^{-1}$, its Frobenius, induced ℓ_2, and induced ℓ_∞ norms by $\|\boldsymbol{A}\|_F$, $\|\boldsymbol{A}\|$, and $\|\boldsymbol{A}\|_\infty$, respectively, and its i-th largest eigenvalue in magnitude by $\lambda_i(\boldsymbol{W})$. Given matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, we denote their Kronecker/entry-wise product as $\boldsymbol{A} \otimes \boldsymbol{B}$/$\boldsymbol{A} \odot \boldsymbol{B}$, and denote $\boldsymbol{A} \succ \boldsymbol{B}$ ($\boldsymbol{A} \succeq \boldsymbol{B}$) if $\boldsymbol{A} - \boldsymbol{B}$ is positive definite (semi-positive definite). We denote the vectorized $\boldsymbol{A}$ in column-major order as $\mathrm{vec}(\boldsymbol{A})$.

2. Well-posedness of MIGNN

We characterize the well-posedness of MIGNN leveraging monotone operator theory; see Appendix B for a brief review. Using the Kronecker product[3] and vectorization of a matrix, we can rewrite Equation (1) into the following equivalent vectorized form

$$\mathrm{vec}(\boldsymbol{Z}^{(k+1)}) = \sigma\big(\boldsymbol{G}^\top \otimes \boldsymbol{W}\mathrm{vec}(\boldsymbol{Z}^{(k)}) + \mathrm{vec}(g_{\boldsymbol{B}}(\boldsymbol{X}))\big). \quad (2)$$

Gu et al. propose the well-posedness condition of IGNN as $\lambda_1(|\boldsymbol{G}^\top \otimes \boldsymbol{W}|) < 1$, guaranteeing that the unique fixed point of Equation (2) can be found by Picard iteration. Let $\boldsymbol{G} = \hat{\boldsymbol{A}}$ introduced before, then all eigenvalues of $\boldsymbol{G}$ are in $[-1, 1]$ with $\lambda_1(\boldsymbol{G}) = 1$. Therefore, the well-posedness of IGNN is equivalent to requiring $\lambda_1(|\boldsymbol{W}|) < 1$ as $\lambda_1(|\boldsymbol{G}^\top \otimes \boldsymbol{W}|) = \lambda_1(\boldsymbol{G})\lambda_1(|\boldsymbol{W}|) = \lambda_1(|\boldsymbol{W}|)$. Then, IGNN parameterizes $\boldsymbol{W}$ by relaxing the well-posedness condition $\lambda_1(|\boldsymbol{W}|) < 1$ to $\|\boldsymbol{W}\|_\infty < 1$, which constrains the magnitudes of eigenvalues of $\boldsymbol{W}$ to be less than 1.

To allow a wider range of $\boldsymbol{W}$ and enhance the expressivity, we utilize the monotone operator theory. Following the discussion in [57], assuming σ to be a proximal operator[4] of a convex closed proper function f, finding the fixed point of Equation (2) is equivalent to solving the monotone inclusion problem: find $\mathbf{0} \in (\mathcal{F} + \mathcal{G})(\mathrm{vec}(\boldsymbol{Z}))$ with $\mathcal{F}$ and $\mathcal{G}$ being two set-valued functions, given below

$$\begin{aligned}\mathcal{F}(\mathrm{vec}(\boldsymbol{Z})) &= (\boldsymbol{I} - \boldsymbol{G}^\top \otimes \boldsymbol{W})\mathrm{vec}(\boldsymbol{Z}) - \mathrm{vec}(g_{\boldsymbol{B}}(\boldsymbol{X})) \\ \mathcal{G} &= \partial f,\end{aligned} \quad (3)$$

where ∂f denotes the subgradient f. While it may seem that the above discussion is placing a strong restriction on σ, we want to note that most activation functions commonly used in machine learning satisfy this requirement. For example, when σ is ReLU, then $\sigma = \mathrm{prox}_f^\alpha$ for $\forall\alpha > 0$ with f being the indicator of the positive octant, i.e. $f(x) = I\{x \geq 0\}$. The above monotone inclusion problem admits a unique solution if the operator $\mathcal{F}$ is strongly monotone, i.e. $\boldsymbol{I} - \boldsymbol{G}^\top \otimes \boldsymbol{W} \succeq m\boldsymbol{I}$ or

$$\frac{1}{2}\big(\boldsymbol{G}^\top \otimes \boldsymbol{W} + \boldsymbol{G} \otimes \boldsymbol{W}^\top\big) \preceq (1-m)\boldsymbol{I}.$$

Therefore, we obtain the following well-posedness condition for MIGNN:

Proposition 2.1 (Well-posedness condition for MIGNN)**.** *Let the nonlinearity σ be ReLU and $\boldsymbol{K} = \frac{1}{2}(\boldsymbol{G}^\top \otimes \boldsymbol{W} + \boldsymbol{G} \otimes \boldsymbol{W}^\top)$. Then the MIGNN model Equation (2) is well-posed as long as $\boldsymbol{K} \preceq (1-m)\boldsymbol{I}$ for some $m > 0$. As $\boldsymbol{K}$ is symmetric, $\boldsymbol{K} \preceq (1-m)\boldsymbol{I}$ is equivalent to requiring that each eigenvalue of $\boldsymbol{K}$ is no more than $1-m$.*

We prove Proposition 2.1 in the appendix; similarly, the proofs of all the other theoretical results are provided in the appendix. *The well-posedness condition in Proposition 2.1 allows for more flexible parameterizations than [24]* by enabling the real part of eigenvalues of $\boldsymbol{W}$ to be in the interval $(-\infty, 1)$ and the imaginary part to be arbitrary.

3. Parameterizations of MIGNN

This section presents the monotone and orthogonal parameterizations of $\boldsymbol{W}$ for MIGNN in Equation (2); the two parameterizations can enhance IGNN's expressivity and stabilize learning LRD, respectively.

3.1. Monotone parameterization

Proposition 2.1 informs us to design a more expressive parameterization of $\boldsymbol{W}$ for MIGNN than that used for IGNN.

Proposition 3.1 (Monotone parameterization)**.** *Let $G = (V, E)$ be a graph and let $\mathbf{G}$ be $\mathbf{L}/2$ with $\mathbf{L} := \mathbf{D}^{-1/2}(\mathbf{D}-$*

[3] See Appendix D for a review of some properties about the Kronecker product.

[4] The proximal operator of a function f is defined as $\mathrm{prox}_f^\alpha(x) \equiv \arg\min_z \left\{\frac{1}{2}\|x-z\|^2 + \alpha f(z)\right\}$ for $\alpha > 0$.

$\boldsymbol{A})\boldsymbol{D}^{-1/2}$ *being the (symmetric) normalized Laplacian, where* $\boldsymbol{A}$ *is the adjacency matrix and* $\boldsymbol{D}$ *is the degree matrix with* $D_{ii} = \sum_{j=1}^{n} A_{ij}$. *Then the following MIGNN model:*

$$\boldsymbol{Z}^{(k+1)} = \sigma(\boldsymbol{W}\boldsymbol{Z}^{(k)}\boldsymbol{G} + g_{\boldsymbol{B}}(\boldsymbol{X}))$$

is well-posed when $\boldsymbol{W}$ *is parameterized as follows:*

$$\boldsymbol{W} = (1-m)\boldsymbol{I} - \boldsymbol{C}\boldsymbol{C}^\top + \boldsymbol{F} - \boldsymbol{F}^\top, \tag{4}$$

where $\boldsymbol{C}, \boldsymbol{F} \in \mathbb{R}^{d\times d}$ *are arbitrary matrices, and* $m > 0$.

Remark 3.2. In monotone parameterization, we set $\boldsymbol{G}$ to be $\boldsymbol{L}/2$, whose eigenvalues are in $[0, 1]$. In contrast, the eigenvalues of $\hat{\boldsymbol{A}}$ used in IGNN, see Sec. 1, are in $[-1, 1]$. Next, we parameterize $\boldsymbol{W}$ as in Equation (4), whose eigenvalues have real part in $(-\infty, 1-m]$. Thus, $\frac{1}{2}(\boldsymbol{G}^\top \otimes \boldsymbol{W} + \boldsymbol{G} \otimes \boldsymbol{W}^\top) \preceq (1-m)\boldsymbol{I}$, guaranteeing the well-posedness of MIGNN. Moreover, $\boldsymbol{W} = (1-m)\boldsymbol{I} - \boldsymbol{C}\boldsymbol{C}^\top + \boldsymbol{F} - \boldsymbol{F}^\top$ describes all possible $\boldsymbol{W}$ that satisfy $\boldsymbol{W} \preceq (1-m)\boldsymbol{I}$.

3.2. Orthogonal parameterization

As discussed in Sec. 1, IGNN learns LRD when $\lambda_1(|\boldsymbol{W}|)$ approaches 1 in magnitude. This is often not the case when starting from Gaussian random initialization – making IGNN unstable for learning LRD. Inspired by the unitary RNN [3], we propose to use the orthogonal parameterization [25; 38; 37] with a learnable scaling factor to stabilize MIGNN in learning LRD. In particular, we parameterize $\boldsymbol{W}$ by the following scaled Cayley map:

$$\boldsymbol{W} = \phi(\gamma)(\boldsymbol{I} - \boldsymbol{S})(\boldsymbol{I} + \boldsymbol{S})^{-1}, \tag{5}$$

where $\phi(\cdot)$ is the sigmoid function and $\gamma \in \mathbb{R}$ is a learnable parameter ensuring $\phi(\gamma) \in (0, 1)$. $\boldsymbol{S} = \boldsymbol{C} - \boldsymbol{C}^\top$ is a skew-symmetric matrix with $\boldsymbol{C} \in \mathbb{R}^{d\times d}$ being an arbitrary parameterized matrix. It is evident that MIGNN with the parameterization in Equation (5) is well-posed with $\boldsymbol{G}$ being $\hat{\boldsymbol{A}}$ defined in Sec. 1. Also, all eigenvalues of $(\boldsymbol{I} - \boldsymbol{S})(\boldsymbol{I} + \boldsymbol{S})^{-1}$ have magnitude 1; see a derivation in Appendix E.3. To effectively learn LRD, MIGNN only requires the scalar $\phi(\gamma)$ to converge to 1.

4. Implementation of MIGNN

Notice that *Picard iteration does not converge when monotone parameterization is applied and converges slowly when orthogonal parameterization is used if* $\phi(\gamma) \to 1$. Thus, we leverage the operator splitting schemes to find the fixed point of MIGNN with monotone or orthogonal parameterization. There are three widely used operator splitting schemes, forward-backward, Peaceman-Rachford, and Douglas-Rachford splitting [48], which will be considered in this paper. Operator splitting schemes often converge faster than Picard iteration and guarantee convergence even when Picard iteration fails [48]. In particular, for small graphs and tasks where learning LRD is not crucial, we use Anderson-accelerated forward-backward splitting (FB) to implement MIGNN with monotone parameterization. For tasks that require learning LRD, we employ Anderson-accelerated Peaceman-Rachford splitting (PR)[5], with Neumann series approximation, to implement MIGNN with orthogonal parameterization. We provide the rationale for these choices in Secs. 4.1.1 and 4.1.2. We structure this section as follows: In Sec. 4.1, we present FB (Sec. 4.1.1)/PR (Sec. 4.1.2) for finding the fixed point of MIGNNs using monotone/orthogonal parameterization. In Sec. 4.2, we present algorithms for updating the parameters of MIGNN.

4.1. Forward propagation for finding the fixed point

4.1.1. Forward-backward splitting

We can find the fixed point of MIGNN in Equation (2) via FB using the following iterative scheme:

$$\begin{aligned}\boldsymbol{Z}^{(k+1)} &:= F_\alpha^{\mathrm{FB}}(\boldsymbol{Z}^{(k)}) \\ &:= \mathrm{prox}_f^\alpha\left(\boldsymbol{Z}^{(k)} - \alpha \cdot \left(\boldsymbol{Z}^{(k)} - \boldsymbol{W}\boldsymbol{Z}^{(k)}\boldsymbol{G} - g_{\boldsymbol{B}}(\boldsymbol{X})\right)\right),\end{aligned} \tag{6}$$

where constant $\alpha > 0$. We provide a detailed implementation of FB in Appendix F.1. The Lipschitz constant of the FB iteration is $L^{\mathrm{FB}} := \sqrt{1 - 2\alpha m + \alpha^2\|\boldsymbol{I} - \boldsymbol{G}^\top \otimes \boldsymbol{W}\|^2}$, see Sec. 5 in [48]. Therefore, FB converges to the fixed point if $\alpha < 2m/\|\boldsymbol{I} - \boldsymbol{G}^\top \otimes \boldsymbol{W}\|^2$. *By choosing a proper* α, *FB can converge in the regime that Picard iteration does not.* However, when the monotone parameterization is used $\|\boldsymbol{W}\|$ can be arbitrarily large. Thus α needs to be small to guarantee the convergence of FB, in which case the Lipschitz constant is close to 1, and the convergence of FB will be significantly slowed. FB is appealing for learning with small graphs and tasks where learning LRD is not crucial. In this case, we use monotone parameterization to improve the expressivity of the model, and ***we denote MIGNN with monotone parameterization using FB as*** **`MIGNN-Mon`**. For large graphs and tasks that require learning LRD, FB suffers from slow convergence. Next, we will present PR, which is better for learning large-scale graphs and LRD. Furthermore, we argue that PR is unsuitable for implementing MIGNN with monotone parameterization.

4.1.2. Peaceman-Rachford splitting

PR used in [57] is guaranteed to converge for a much broader choice of α and requires fewer iterations than FB. However, *each iteration of PR requires inverting large matrices, which is computationally much more expensive and less scalable than FB.* More precisely, PR finds the solution $\boldsymbol{Z}^*$ of the MIGNN by letting $\boldsymbol{Z}^* = \mathrm{prox}_f^\alpha(\boldsymbol{U}^*)$ where $\boldsymbol{U}^* \in \mathbb{R}^{d\times n}$ is obtained from the fixed-point iteration $\mathrm{vec}(\boldsymbol{U}^{(k+1)}) = F_\alpha^{\mathrm{PR}}(\mathrm{vec}(\boldsymbol{U}^{(k)})) := \mathcal{C}_{\mathcal{F}}\mathcal{C}_{\mathcal{G}}(\mathrm{vec}(\boldsymbol{U}^{(k)}))$ with $\mathcal{C}_{\mathcal{F}}$ and $\mathcal{C}_{\mathcal{G}}$ being the Cayley operators (see Appendix B for details) of $\mathcal{F}$ and $\mathcal{G}$, respectively. Let $\boldsymbol{u}^{(k)}$ be the shorthand notation

[5] We denote Anderson-accelerated FB and PR as FB and PR.

of $\mathrm{vec}(\boldsymbol{U}^{(k)})$. Then we can formulate PR as follows:

$$\begin{aligned}\boldsymbol{u}^{(k+1)} := F_{\alpha}^{\mathrm{PR}}(\boldsymbol{u}^{(k)}) = 2\boldsymbol{V}\big(2\,\mathrm{prox}_f^{\alpha}(\boldsymbol{u}^{(k)}) - \boldsymbol{u}^{(k)} + \\ \alpha\,\mathrm{vec}(g_{\boldsymbol{B}}(\boldsymbol{X}))\big) - 2\,\mathrm{prox}_f^{\alpha}(\boldsymbol{u}^{(k)}) + \boldsymbol{u}^{(k)},\end{aligned} \tag{7}$$

where $\boldsymbol{V} := (\boldsymbol{I} + \alpha(\boldsymbol{I} - \boldsymbol{G}^\top \otimes \boldsymbol{W}))^{-1}$ and $\boldsymbol{u}^{(0)}$ is the zero vector. With the parameterizations discussed in Sec. 3, the linear operator $\mathcal{F}$ in Equation (3) is strongly monotone and L-Lipschitz where $L = \|\boldsymbol{I} - \boldsymbol{G}^\top \otimes \boldsymbol{W}\|$. Therefore, its Cayley operator $\mathcal{C}_{\mathcal{F}}$ and hence F_{α}^{PR} is contractive with the optimal choice of α being $1/L$; see Section 6 in [48]. In particular, it is suggested to choose $\alpha = 1/(1+\phi(\gamma))$ when using orthogonal parameterization $\boldsymbol{W} = \phi(\gamma)(\boldsymbol{I} - \boldsymbol{S})(1 + \boldsymbol{S})^{-1}$. The pseudocode for the detailed implementation of PR in Equation (7) can be found in Appendix F.1.

Remark 4.1. Douglas-Rachford splitting (DR) is another option for finding the fixed point of the equilibrium equation, which is often faster than PR. However, in our case PR is contractive, making it faster than DR for the same α.

PR also benefits MIGNNs in learning LRD when an orthogonal parameterization is used. To see this, we have the following Neumann series expansion of $\boldsymbol{V}(\boldsymbol{u}^{(k)})$:

$$\begin{aligned}\boldsymbol{V}(\boldsymbol{u}^{(k)}) &= (\boldsymbol{I} + \alpha(\boldsymbol{I} - \boldsymbol{G}^\top \otimes \boldsymbol{W}))^{-1}(\boldsymbol{u}^{(k)}) \\ &= \frac{1}{1+\alpha}\left(\boldsymbol{I} - \frac{\boldsymbol{G}^\top \otimes \boldsymbol{W}}{1+1/\alpha}\right)^{-1}(\boldsymbol{u}^{(k)}) \\ &= \frac{1}{1+\alpha}\sum_{i=0}^{\infty}\frac{\mathrm{vec}\left(\boldsymbol{W}^i\boldsymbol{U}^{(k)}\boldsymbol{G}^i\right)}{(1+1/\alpha)^i}\end{aligned} \tag{8}$$

where the last equality follows from $(\boldsymbol{A}\otimes\boldsymbol{B})^k = \boldsymbol{A}^k \otimes \boldsymbol{B}^k$, and $(\boldsymbol{A}\otimes\boldsymbol{B})\mathrm{vec}(\boldsymbol{C}) = \mathrm{vec}(\boldsymbol{B}\boldsymbol{C}\boldsymbol{A}^\top)$ for $\forall \boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{C}$ that satisfy dimensional consistency. Equation (8) indicates that each node can access information from its ∞-hop neighbors in a single PR iteration for MIGNN with orthogonal parameterization. This cannot be said of monotone parameterization with large $\|\boldsymbol{W}\|$, as the Neumann series expansion in the last equality of Equation (8) no longer applies. Evaluating $\frac{1}{1+\alpha}\big(\boldsymbol{I} - \frac{\boldsymbol{G}^\top\otimes\boldsymbol{W}}{1+1/\alpha}\big)^{-1}(\boldsymbol{u}^{(k)})$ can be carried out by using Bartels–Stewart algorithm [9], which converts computing $\boldsymbol{V}$ into diagonalizing the matrix $\boldsymbol{G}^\top$ and $\boldsymbol{W}$, respectively. From Equation (8), we have

$$\begin{aligned}&\boldsymbol{V}(\mathrm{vec}(\boldsymbol{U}^{(k)})) \\ &= \frac{1}{1+\alpha}\mathrm{vec}\Big(\boldsymbol{Q}_{\boldsymbol{W}}\Big[\boldsymbol{H} \odot \Big(\boldsymbol{Q}_{\boldsymbol{W}}^{-1}\boldsymbol{U}^{(k)}\boldsymbol{Q}_{\boldsymbol{G}^\top}\Big)\Big]\boldsymbol{Q}_{\boldsymbol{G}^\top}^\top\Big)\end{aligned} \tag{9}$$

where $\boldsymbol{Q}_{\boldsymbol{G}^\top}\boldsymbol{\Lambda}_{\boldsymbol{G}^\top}\boldsymbol{Q}_{\boldsymbol{G}^\top}^\top$ and $\boldsymbol{Q}_{\boldsymbol{W}}\boldsymbol{\Lambda}_{\boldsymbol{W}}\boldsymbol{Q}_{\boldsymbol{W}}^{-1}$ are the eigendecomposition of $\boldsymbol{G}^\top$ and of $\boldsymbol{W}$, respectively, and $\boldsymbol{H} \in \mathbb{R}^{d\times n}$ whose (i,j)-th entry is $H_{ij} = 1/(1 - \frac{1}{1+1/\alpha}(\boldsymbol{\Lambda}_{\boldsymbol{W}})_{ii}(\boldsymbol{\Lambda}_{\boldsymbol{G}^\top})_{jj})$. We provide a proof of Equation (9) in Appendix E.4. According to Equation (9), one only needs to calculate the eigen-decomposition of $\boldsymbol{G}$ once prior to training and the eigen-decomposition of $\boldsymbol{W}$ once per epoch. The above matrix inversion procedure echos the idea of EIGNN [39]. MIGNN has multiple layers, with each fixed point iteration representing one layer. In contrast, EIGNN is reduced to a one-layer model; see Appendix A.2 for details on EIGNN.

Although PR can capture LRD in a single iteration, computing $\boldsymbol{V}$ in Equation (7) requires computationally prohibitive matrix inversion. To overcome this computational issue, we use Neumann series expansion to approximate the matrix inversion when orthogonal parameterization is used for MIGNN. Notice that the Neumann series approximation does not work for MIGNN using monotone parameterization since we can no longer use the Neumann series approximation. Therefore, MIGNN with monotone parameterization using PR splitting is not scalable to learning large graphs.

Neumann series approximation. In the orthogonal parameterization of $\boldsymbol{W}$ we have $\|\frac{\boldsymbol{G}^\top\otimes\boldsymbol{W}}{1+1/\alpha}\| < 1$, ensuring efficient approximation of $\boldsymbol{V}$ in Equation (7) using only a few terms of its Neumann series expansion. The K-th order Neumann series expansion of $\boldsymbol{V}(\mathrm{vec}(\boldsymbol{U}^{(k)}))$ is given by

$$\boldsymbol{N}_K(\mathrm{vec}(\boldsymbol{U}^{(k)})) := \frac{1}{1+\alpha}\sum_{i=0}^{K}\frac{\mathrm{vec}\left(\boldsymbol{W}^i\boldsymbol{U}^{(k)}\boldsymbol{G}^i\right)}{(1+1/\alpha)^i}. \tag{10}$$

According to Equation (7), the K-th order Neumann series approximated PR iteration function, denoted as $\tilde{F}_{\alpha}^{\mathrm{PR,K}}$, can be written as follows:

$$\begin{aligned}\boldsymbol{u}^{(k+1)} := \tilde{F}_{\alpha}^{\mathrm{PR,K}}(\boldsymbol{u}^{(k)}) = 2\boldsymbol{N}_K\Big(2\,\mathrm{prox}_f^{\alpha}(\boldsymbol{u}^{(k)}) - \boldsymbol{u}^{(k)} \\ + \alpha\,\mathrm{vec}(g_{\boldsymbol{B}}(\boldsymbol{X}))\Big) - 2\,\mathrm{prox}_f^{\alpha}(\boldsymbol{u}^{(k)}) + \boldsymbol{u}^{(k)}.\end{aligned} \tag{11}$$

Each node can access information from its K-hop neighbors using the K-th order Neumann series approximated PR iteration, which is more efficient than the existing IGNN. Also, such a treatment can significantly accelerate forward propagation. We can intuitively understand this as follows: Each iteration of MIGNN, with K-th order Neumann series approximated PR iteration, aggregates information from K-hop neighbors, enabling the use of much fewer iterations than that of IGNN, which aggregates one hop per iteration. MIGNN can use a much smaller $\lambda_1(|\boldsymbol{W}|)$ than IGNN to reach the same number of hops, meaning MIGNN converges much faster than IGNN.

Regarding the computational complexity: In each epoch, the parameter K in the K-th order Neumann series affects the training time complexity linearly as $\mathcal{O}(KMd|E_P|)$, where $|E_P|$ denotes the number of non-zero entries in the graph-related matrix $\boldsymbol{G}$, M denotes the maximal number of iterations, and d is the feature dimension which is much smaller than the number of nodes. ***We denote the model as `MIGNN-N`K when W is parameterized with orthogonal parameterization, and the fixed point is obtained using K-th order Neumann series approximated PR iteration.***

4.1.3. Anderson acceleration

Note that the main steps in both FB and PR involve solving iterative equations, e.g. Equations (6) and (7), and we can utilize Anderson acceleration [2] to accelerate the convergence of these iterative equations. We provide the detailed formulation and pseudocode for Anderson-accelerated operator splitting-based MIGNNs in Appendix F.3.

4.2. Backward propagation for updating MIGNNs

We derive backpropagation for MIGNN based on implicit differentiation [23; 5; 17]. Recall that the vectorized MIGNN $\mathrm{vec}(\boldsymbol{Z}) = \sigma\left(\boldsymbol{G}^\top \otimes \boldsymbol{W}\mathrm{vec}(\boldsymbol{Z}) + \mathrm{vec}(g_{\boldsymbol{B}}(\boldsymbol{X}))\right)$, has equilibrium point $\mathrm{vec}(\boldsymbol{Z}^*)$. For any loss function ℓ and any parameter θ ($\boldsymbol{W}$ or $\boldsymbol{B}$), we have

$$\begin{aligned}\frac{\partial \ell}{\partial \theta} =& \frac{\partial \ell}{\partial \mathrm{vec}(\boldsymbol{Z}^*)}\left(\boldsymbol{I} - \boldsymbol{J}\left(\boldsymbol{G}^\top \otimes \boldsymbol{W}\right)\right)^{-1} \\ &\times \frac{\partial \sigma\left(\boldsymbol{G}^\top \otimes \boldsymbol{W}\mathrm{vec}(\boldsymbol{Z}^*) + \mathrm{vec}(g_{\boldsymbol{B}}(\boldsymbol{X}))\right)}{\partial \theta}\end{aligned} \tag{12}$$

where J is the Jacobian of the nonlinearity σ evaluated at $\boldsymbol{G}^\top \otimes \boldsymbol{W}\mathrm{vec}(\boldsymbol{Z}^*) + \mathrm{vec}(g_{\boldsymbol{B}}(\boldsymbol{X}))$. The values of the first and last term in Equation (12) can be found through automatic differentiation by running one more iteration in the forward pass. Note that the product of the first two terms remains the same for any θ. Hence one only needs to compute it once in each backward pass. However, it can still be expensive to find $(\partial \ell)/(\partial \mathrm{vec}(\boldsymbol{Z}^*))(\boldsymbol{I} - \boldsymbol{J}\left(\boldsymbol{G}^\top \otimes \boldsymbol{W}\right))^{-1}$. Following Theorem 2 in [57], the operator splitting methods can be used in the backward pass so that computing $(\boldsymbol{I} - \boldsymbol{J}(\boldsymbol{G}^\top \otimes \boldsymbol{W}))^{-1}$ can be converted into computing $\boldsymbol{V} = (\boldsymbol{I} - (\boldsymbol{G}^\top \otimes \boldsymbol{W}))^{-1}$, which is already calculated in the forward pass; see Appendix F.2. Similar to the forward propagation, the backpropagation can also benefit from Anderson acceleration using an iterative formulation, and we provide more details in Appendix F.2.

5. Experimental Results

In this section, we compare the performance of MIGNN-Mon and MIGNN-NK with IGNN and several other popular GNNs on various graph classification tasks at both node and graph levels. We aim to show that 1) MIGNN-Mon is more expressive than IGNN for both node and graph classifications, and 2) MIGNN-NK can learn LRD stably. The training procedure details and hyperparameters used in each task are provided in Appendix J.

5.1. Directed chain classification

We first test MIGNNs on the synthetic chain task using the experimental setup from [39], which is *designed to test the efficacy of GNNs in learning LRD*. The chain dataset comprises c classes and n_c single-linked directed chains, each containing l nodes. For each chain, only the feature on the first node has the label information, and the goal is to classify all other nodes in the chain with the same label. Note that to classify nodes accurately, the information of the first node needs to be propagated to all the remaining nodes, requiring GNNs to learn LRD. The data is partitioned into training, validation, and test sets of 5%, 10%, and 85%, respectively. We consider binary ($c = 2$) and three-class classification ($c = 3$) problems over several different chain lengths. For IGNN, we use the experimental settings used in [24]. We consider MIGNN-NK for this task. Fig. 3 shows the averaged test accuracy over 5 random seeds of different models for classifying directed chains of length ranging from 50 to 300 in an increment of 50 for the binary case and from 40 to 200 in an increment of 20 for the three-class case. *Both MIGNN-N3 and MIGNN-N5 classify the chains almost perfectly for all random initializations of the considered chain lengths for both binary and three-class classification tasks.* We study the impact of the order of Neumann series approximation on the chain classification accuracy and computational time in Appendix G.

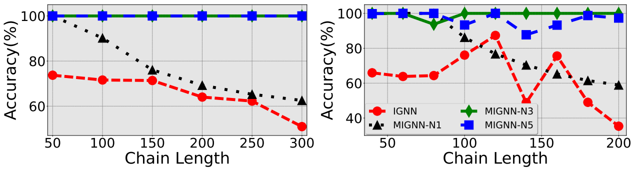


Figure 3. The accuracy of IGNN and MIGNN of different configurations for classifying chains of different lengths. Left: binary classification ($c = 2$). Right: three-class classification ($c = 3$). The legend for the left figure is the same as that in the right one.

Based on the operator splitting theory, we expect MIGNNs to be more computationally efficient than IGNNs when both models can accurately classify the nodes. Fig. 4 compares the accuracy and computational efficiency of MIGNN-N2 over IGNN for a three-class chain classification. We see that MIGNN-N2 achieves and maintains perfect accuracy (left panel) after around 200 epochs and takes approximately constant time (middle panel) and iterations (right panel) for each epoch. In contrast, IGNN abruptly changes around epoch 350, resulting in a significantly increased iteration count and time elapsed for each epoch.

5.2. A graph node classification task

In this section, we compare MIGNN-N1 against IGNN and other GNNs for a larger-scale graph node classification task – Amazon co-purchasing dataset, which contains 334863 nodes, 925872 edges, and the diameter of the graph is 44 [63]; we provide details of the Amazon co-purchasing dataset in Appendix I. *Learning LRD is crucial for accurately classifying the graph node in this task [24].* As in [16], we train on portions of the graph ranging from 5% to 9%, and test on sets representing 10% of the total graph. We report both Macro-F1 and Micro-F1, which are consistent with [24]. Fig. 5 contrasts the computational cost of MIGNN-N1 with IGNN using 5% of the graph for training. We observe that $\lambda_1(|\boldsymbol{W}|)$ of MIGNN-N1 is close to one

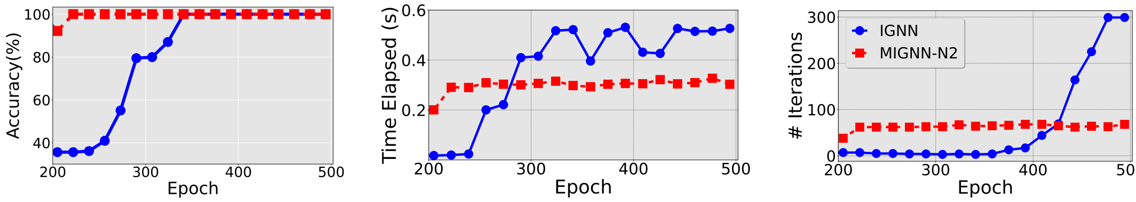


Figure 4. The accuracy and efficiency of MIGNN-N2 over IGNN for three class chains, of length 140, classification. The time elapsed is computed for both forward and backward propagation. The number of iterations in the forward pass is plotted on the right. The left two panels share the same legend as the right one.

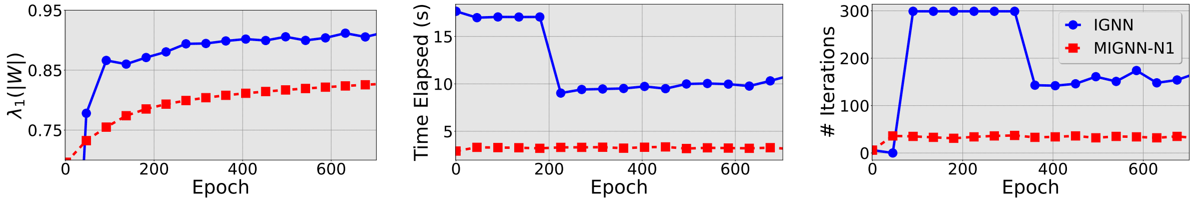


Figure 5. Epoch vs. $\lambda_1(|\boldsymbol{W}|)$, the time required for each epoch, and iterations required for each epoch of IGNN and MIGNN-N1 for the Amazon dataset with 5% training portion. The time elapsed is computed for both forward and backward propagation. The number of iterations in the forward pass is plotted on the right. The left two panels share the same legend as the right one.

Portion	0.05	0.06	0.07	0.08	0.09
GCN [32]	82.01/78.96	83.15/80.43	84.15/81.31	85.05/82.12	85.57/82.68
SGC [59]	79.26/77.80	81.26/79.94	83.05/81.73	84.31/83.02	85.47/84.05
SSE [16]	84.52/81.14	85.05/81.56	85.57/82.33	86.42/83.07	86.98/83.98
IGNN [24]	84.01/80.85	85.31/**82.33**	85.57/82.85	86.43/84.12	**87.26/84.68**
MIGNN-N1	**84.59/81.40**	**85.37**/82.13	**85.89/83.44**	**86.65/84.22**	87.09/84.59

Table 1. Amazon Micro-F1/Macro-F1 accuracy (%). We take the results of the baseline models from [24].

but much smaller than that of IGNN (Fig. 5 left), implying MIGNN-N1 can learn LRD while exhibiting faster convergence than IGNN as confirmed by the fact that MIGNN-N1 saves significantly in computational time (Fig. 5 middle) and the number of iterations (Fig. 5 right) over IGNN.

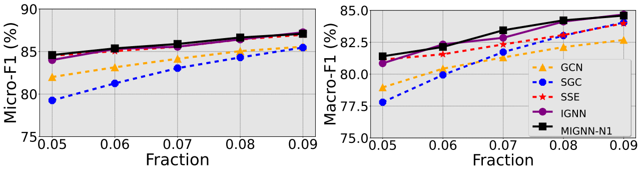


Figure 6. Fraction vs. Micro-F1 (left) and Macro-F1 (right) training accuracy on the Amazon dataset.

Fig. 6 contrasts MIGNN-N1 with baseline models (the same as those on [24] except we exclude the worst performed one) when trained on portions of the graph ranging from 5% to 9%. We see that MIGNN-N1 outperforms almost all baseline models over all different portions of the graph for the training. Though MIGNN-N1 does not outperform IGNN significantly, *MIGNN-N1 enjoys significant computational advantages over IGNN*. In this task, implicit models, including IGNN and MIGNN-N1 outperform all explicit models by a remarkable margin. Table 1 lists the accuracy numbers corresponding to Fig. 6.

5.3. Some other small-scale node classification tasks

We further test MIGNNs with both parameterizations for a few small-scale graph node classification tasks, including Cora, Citeseer, and Pubmed; each dataset's statistics of node/edge/average shortest path between nodes are 2485/5069/5.27, 2120/3679/9.31, 19717/44324/6.34, respectively. *Learning LRD is not crucial for these tasks since the graph's diameter is quite small*. However, even for these small-scale graph node classification tasks MIGNN is still able to outperform IGNN and can even outperform many explicit GNNS and other improved implicit models. We use the training procedure outlined in [15] and report the mean accuracy of 10-fold cross-validation in Table 2, where the baseline explicit models are those that achieve strong results for these three datasets [15]. These results confirm the expressivity of MIGNN using monotone parameterization. Both MIGNN models outperform IGNN and even outperform many explicit and improved implicit GNNs. We provide experimental details in Appendix J.

Datasets	Cora	Citeseer	Pubmed
Geom-GCN [45]	85.35 ± 1.57	$\mathbf{78.02 \pm 1.15}$	89.95 ± 0.47
GCNII [14]	$\mathbf{88.37 \pm 1.25}$	77.33 ± 1.48	$\mathbf{90.15 \pm 0.43}$
APPNP [20]	85.09 ± 0.25	75.73 ± 0.30	79.73 ± 0.31
GCN+GDC [22]	83.58 ± 0.23	73.35 ± 0.27	78.72 ± 0.37
GIND [15]	88.25	76.81	89.22
IGNN [24]	85.80	75.24	87.66
EIGNN [39]	85.89	75.31	87.92
MIGNN-Mon	87.02 ± 1.2	76.15 ± 1.8	$89.18 \pm .40$
MIGNN-NK	88.31 ± 1.2	76.91 ± 1.6	89.28 ± 0.35

Table 2. Classification mean accuracy (%) ± standard deviation for three small-scale node classification tasks (10-fold cross-validation). We boldface the best accuracy for each task and color the optimal accuracy of implicit models in blue. Here we take K in MIGNN-NK as a hyper parameter and report the best performance (Cora:8, Citeseer:10, PubMed:10).

Datasets # graphs/Avg # nodes	MUTAG 188/17.9	PTC 344/25.5	COX2 467/41.2	PROTEINS 1113/39.1	NCI1 4110/29.8
WL [50]	84.1 ± 1.9	58.0 ± 2.5	83.2 ± 0.2	74.7 ± 0.5	$\mathbf{84.5 \pm 0.5}$
DCNN [4]	67.0	56.6	—	61.3	62.6
DGCNN [64]	85.8	58.6	—	75.5	74.4
GIN [61]	89.4 ± 5.6	64.6 ± 7.0	—	76.2 ± 3.4	82.7 ± 1.7
FDGNN [19]	88.5 ± 3.8	63.4 ± 5.4	83.3 ± 2.9	76.8 ± 2.9	77.8 ± 1.6
IGNN [24]	76.0 ± 13.4	60.5 ± 6.4	79.7 ± 3.4	76.5 ± 3.4	73.5 ± 1.9
GIND [15]	89.3 ± 7.4	66.9 ± 6.6	84.8 ± 4.2	77.2 ± 2.9	78.8 ± 2.9
MIGNN-Mon	81.8 ± 9.1	$\mathbf{72.6 \pm 6.7}$	85.0 ± 5.3	77.9 ± 3.4	73.6 ± 2.0
MIGNN-N1	86.1 ± 9.1	70.9 ± 6.5	86.5 ± 2.8	79.0 ± 3.3	78.4 ± 1.2
MIGNN-N3	91.4 ± 7.5	71.2 ± 3.2	$\mathbf{88.2 \pm 4.1}$	$\mathbf{80.1 \pm 3.8}$	80.8 ± 1.81

Table 3. Graph classification mean accuracy (%) ± standard deviation for 10-fold cross-validation.

5.4. Graph classification

We further verify that *MIGNN-Mon can be more expressive than IGNN for graph classification* since the eigenvalues of monotone parameterization are more flexible than IGNN. We consider five bioinformatics-related graph classification benchmarks: MUTAG, PTC, COX2, PROTEINS, and NCI1 [62], and some details of these datasets are provided in Appendix I. We perform training with 10-fold cross-validation using the experimental setup of [24]. The averaged test accuracy and standard deviation across the 10 folds are shown in Table 3, where the baseline explicit GNNs are adopted from [15]. For both IGNN and MIGNN-Mon, we use the hyperparameters outlined in [24]. Clearly, *MIGNN-Mon outperforms IGNN on all five graph classification tasks*. Indeed, MIGNN outperforms all other implicit models (IGNN, GIND, MGNII). To confirm that the enhanced expressivity is due to the expanded range of $\boldsymbol{W}$, we report on the evolution of $\lambda_1(|\boldsymbol{W}|)$ for three of the ten folds of MUTAG in Fig. 7, showing that $\lambda_1(|\boldsymbol{W}|)$ exceeds one. Table 3 also reports the accuracy of MIGNN-N1 and MIGNN-N3 and compares with a few existing implicit GNNs, including IGNN and GIND. Overall, MIGNN performs well for graph classification; in particular, MIGNN-N3 achieves the best accuracy on COX2 and PROTEINS tasks among all studied models. We provide an ablation study of the impact of the order of the Neumann series on classification accuracy and computational time in Appendix G.

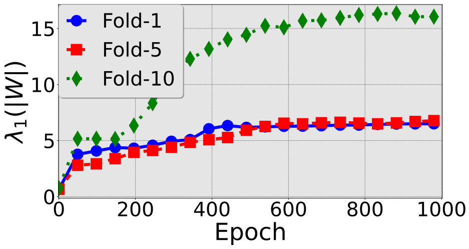


Figure 7. $\lambda_1(|\boldsymbol{W}|)$ of MIGNN-Mon vs. Epoch on MUTAG.

5.5. Physical diffusion in networks

We further consider a physical problem of fluid flow in porous media, following [43]. The model is a 3D graph whose nodes and edges correspond to pore chambers and throats. We sample training graphs of different sizes between 100 and 500, which are generated to fit into 0.1 m^3 cubes. We aim to predict the equilibrium pressures $\boldsymbol{Z}^*$ inside pore networks G. We train MIGNN to minimize the mean-squared error (MSE) between the prediction and $\boldsymbol{Z}^*$. We utilize the experimental setup of [43] and include their reported results for IGNN. Both IGNN and MIGNN use the same encoder and decoder architecture. Graphs of $50 - 200$ nodes are sampled in training and 1000 test graphs are generated for pore counts from 200 to 500. Fig. 8 shows the MSE for the test graphs as the number of nodes (pores) varies from 200 to 500. *MIGNN with both monotone and orthogonal parameterizations outperform IGNN by a significant margin*. For this task of learning physical diffusion in networks, CGS [43] performs better than MIGNN and IGNN in accuracy. As future work, we plan to integrate the the learnable graph-related matrix $\boldsymbol{G}$ used in CGS with our proposed MIGNN to further improve the performance of MIGNN for learning physical diffusion in networks.

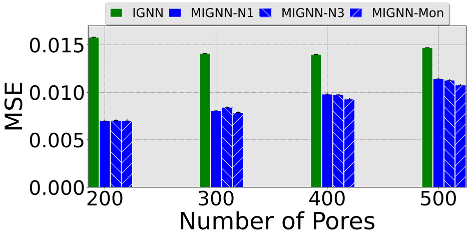


Figure 8. The average MSE of 500 sampled test iterations vs. the number of pores. The error bars represent the standard error of the prediction. MIGNN with different parameterizations outperforms IGNN by a significant amount.

6. Concluding Remarks

We propose MIGNN based on a monotone operator viewpoint of IGNN. In particular, MIGNN can be parameterized more flexibly than the baseline IGNN. We provide efficient implementations of MIGNN leveraging different Anderson-accelerated operator splitting schemes. Numerically, MIGNN remarkably outperforms the baseline IGNN in accuracy, stability, computational efficiency, and learning long-range dependencies. As IGNNs are closely related to RNNs, an interesting future direction is to explore if the ideas from other RNN-related techniques that benefit learning long-range dependencies [60; 26] can be adapted to the improvement of IGNNs.

Acknowledgements

This material is based on research sponsored by NSF grants DMS-1952339, DMS-2152762, and DMS-2208361, DOE grant DE-SC0021142 and DE-SC0023490. Moreover, this material is based, in part, upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, as part of their Applied Mathematics Research Program. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for the United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

References

[1] Almeida, L. B. A learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In *Artificial neural networks: concept learning*, pp. 102–111, 1990.

[2] Anderson, D. G. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4): 547–560, 1965.

[3] Arjovsky, M., Shah, A., and Bengio, Y. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pp. 1120–1128, 2016.

[4] Atwood, J. and Towsley, D. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[5] Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32, 2019.

[6] Bai, S., Koltun, V., and Kolter, J. Z. Multiscale deep equilibrium models. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

[7] Bai, S., Koltun, V., and Kolter, J. Z. Stabilizing equilibrium models by Jacobian regularization. In *International Conference on Machine Learning*, pp. 554–565. PMLR, 2021.

[8] Bai, S., Koltun, V., and Kolter, J. Z. Neural deep equilibrium solvers. In *International Conference on Learning Representations*, 2022.

[9] Bartels, R. H. and Stewart, G. W. Solution of the matrix equation ax+ xb= c [f4]. *Communications of the ACM*, 15(9):820–826, 1972.

[10] Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[11] Biewald, L. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[12] Blondel, M., Berthet, Q., Cuturi, M., Frostig, R., Hoyer, S., Llinares-López, F., Pedregosa, F., and Vert, J.-P. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*, 2021.

[13] Bolte, J., Le, T., Pauwels, E., and Silveti-Falls, T. Nonsmooth implicit differentiation for machine-learning and optimization. *Advances in neural information processing systems*, 34:13537–13549, 2021.

[14] Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pp. 1725–1735. PMLR, 2020.

[15] Chen, Q., Wang, Y., Wang, Y., Yang, J., and Lin, Z. Optimization-induced graph implicit nonlinear diffusion. In *International Conference on Machine Learning*, pp. 3648–3661. PMLR, 2022.

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

[17] El Ghaoui, L., Gu, F., Travacca, B., Askari, A., and Tsai, A. Implicit deep learning. *SIAM Journal on Mathematics of Data Science*, 3(3):930–958, 2021.

[18] Elman, J. L. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[19] Gallicchio, C. and Micheli, A. Fast and deep graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34 (04), pp. 3898–3905, 2020.

[20] Gasteiger, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*, 2018.

[21] Gasteiger, J., Bojchevski, A., and Günnemann, S. Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations*, 2019.

[22] Gasteiger, J., Weißenberger, S., and Günnemann, S. Diffusion improves graph learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[23] Gilbert, J. C. Automatic differentiation and iterative processes. *Optimization methods and software*, 1(1): 13–21, 1992.

[24] Gu, F., Chang, H., Zhu, W., Sojoudi, S., and El Ghaoui, L. Implicit graph neural networks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020.

[25] Helfrich, K., Willmott, D., and Ye, Q. Orthogonal recurrent neural networks with scaled cayley transform. In *International Conference on Machine Learning*, pp. 1969–1978. PMLR, 2018.

[26] Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[27] Horn, R. A. and Johnson, C. R. Topics in matrix analysis, 1991. *Cambridge University Presss, Cambridge*, 37:39, 1991.

[28] Jafarpour, S., Davydov, A., Proskurnikov, A., and Bullo, F. Robust Implicit Networks via Non-Euclidean Contractions. In *Advances in Neural Information Processing Systems*, volume 34, pp. 9857–9868, 2021.

[29] Jing, L., Shen, Y., Dubcek, T., Peurifoy, J., Skirlo, S., LeCun, Y., Tegmark, M., and Soljačić, M. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1733–1741. JMLR. org, 2017.

[30] Kawaguchi, K. On the theory of implicit deep learning: Global convergence with implicit layers. In *International Conference on Learning Representations*, 2021.

[31] Kincaid, D. and Cheney, W. Numerical analysis, brooks. *Cole Publishing Company*, 20:10–13, 1991.

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

[33] Kolter, J. Z. and Manek, G. Learning stable deep dynamics models. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[34] Kolter, J. Z., Duvenaud, D., and Johnson, M. Deep implicit layers - neural ODEs, deep equilibrium models, and beyond, 2020.

[35] Krantz, S. G. and Parks, H. R. *The implicit function theorem: history, theory, and applications*. Springer Science & Business Media, 2002.

[36] Leskovec, J., Adamic, L. A., and Huberman, B. A. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, 1(1):5–es, 2007.

[37] Lezcano Casado, M. Trivializations for gradient-based optimization on manifolds. *Advances in Neural Information Processing Systems*, 32, 2019.

[38] Lezcano-Casado, M. and Martınez-Rubio, D. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *International Conference on Machine Learning*, pp. 3794–3803. PMLR, 2019.

[39] Liu, J. and et al. Efficient graph neural networks. In *Advances in Neural Information Processing Systems*, 2021.

[40] Mhammedi, Z., Hellicar, A., Rahman, A., and Bailey, J. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2401–2409. JMLR. org, 2017.

[41] Nguyen, T., Baraniuk, R., Bertozzi, A., Osher, S., and Wang, B. MomentumRNN: Integrating momentum into recurrent neural networks. *Advances in Neural Information Processing Systems*, 33:1924–1936, 2020.

[42] Nguyen, T. M., Baraniuk, R., Kirby, R., Osher, S., and Wang, B. Momentum transformer: Closing the performance gap between self-attention and its linearization. In *Mathematical and Scientific Machine Learning*, pp. 189–204. PMLR, 2022.

[43] Park, J., Choo, J., and Park, J. Convergent graph solvers. In *International Conference on Learning Representations*, 2022.

[44] Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. 1310–1318, 2013.

[45] Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. Geom-GCN: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020.

[46] Pineda, F. Generalization of back propagation to recurrent and higher order neural networks. In *Neural information processing systems*, 1987.

[47] Robinson, S. M. An implicit-function theorem for a class of nonsmooth functions. *Mathematics of operations research*, 16(2):292–309, 1991.

[48] Ryu, E. K. and Boyd, S. Primer on monotone operator methods. *Appl. Comput. Math*, 15(1):3–43, 2016.

[49] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[50] Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., and Borgwardt, K. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*, pp. 488–495. PMLR, 2009.

[51] Thorpe, M., Nguyen, T. M., Xia, H., Strohmer, T., Bertozzi, A., Osher, S., and Wang, B. GRAND++: Graph neural diffusion with a source term. In *International Conference on Learning Representations*, 2022.

[52] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[53] Vorontsov, E., Trabelsi, C., Kadoury, S., and Pal, C. On orthogonality and learning recurrent networks with long term dependencies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3570–3578. JMLR. org, 2017.

[54] Wang, B., Xia, H., Nguyen, T., and Osher, S. How does momentum benefit deep neural networks architecture design? a few case studies. *Research in the Mathematical Sciences*, 9(3):57, 2022.

[55] Welling, M. and Kipf, T. N. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2016.

[56] Werbos, P. J. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988.

[57] Winston, E. and Kolter, J. Z. Monotone operator equilibrium networks. In *Advances in neural information processing systems*, volume 33, pp. 10718–10728, 2020.

[58] Wisdom, S., Powers, T., Hershey, J., Le Roux, J., and Atlas, L. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 4880–4888, 2016.

[59] Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Simplifying graph convolutional networks. In *International conference on machine learning*, pp. 6861–6871. PMLR, 2019.

[60] Xia, H., Suliafu, V., Ji, H., Nguyen, T., Bertozzi, A., Osher, S., and Wang, B. Heavy ball neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 34:18646–18659, 2021.

[61] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

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

[63] Yang, J. and Leskovec, J. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, pp. 1–8, 2012.

[64] Zhang, M., Cui, Z., Neumann, M., and Chen, Y. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32 (1), 2018.

Supplementary materials for

Implicit Graph Neural Networks: A Monotone Operator Viewpoint

A. A Brief Review of IGNN and Related Models

A.1. IGNN: Forward and backward propagation

IGNN employs a projected gradient descent method in the training phase to ensure their proposed well-posedness condition is satisfied. In forward propagation, IGNN finds the equilibrium through the Picard iteration. During backward propagation, IGNN uses the implicit function theorem at the equilibrium to compute the gradient. The computationally expensive terms related to $\frac{\partial \ell}{\partial \mathrm{vec}(\boldsymbol{Z}^*)}\left(\boldsymbol{I}-\boldsymbol{J}\left(\boldsymbol{G}^\top \otimes \boldsymbol{W}\right)\right)^{-1}$, see Section 4.2 for notations, is also computed implicitly through Picard iteration.

A.2. EIGNN, CGS, and GIND

EIGNN. Efficient infinite-depth graph neural networks (EIGNN) is an implicit graph neural network model proposed by Liu et al. [39] whose counterpart in explicit GNN is simple graph convolution (SGC) [59]. The main update step in EIGNN is given by

$$\boldsymbol{Z}^{(k+1)} = \gamma g(\boldsymbol{F})\boldsymbol{Z}^{(k)}\boldsymbol{G} + \boldsymbol{X} \tag{13}$$

where $\boldsymbol{Z}^{(\cdot)}$ denotes the hidden feature, $\boldsymbol{G}$ is the normalized augmented adjacency matrix $\hat{\boldsymbol{A}}$ (see Section 1), $\boldsymbol{X}$ is the input feature, $g(\boldsymbol{F})$ is the weight matrix which is parameterized to guarantee convergence, and γ is a constant scalar in $(0,1)$. Note that, there is no nonlinearity in the fixed-point Equation (13) and this allows EIGNN to find the equilibrium by the following closed formula:

$$\lim_{k\to\infty} \mathrm{vec}\left(\boldsymbol{Z}^{(k+1)}\right) = \left(\boldsymbol{I} - \gamma(\boldsymbol{G}^\top \otimes g(\boldsymbol{F}))\right)^{-1}\mathrm{vec}(\boldsymbol{X}). \tag{14}$$

For computation efficiency consideration, the matrix inverse operation is reduced to eigen-decomposition of $\boldsymbol{G}^\top$ and $g(\boldsymbol{F})$ where the eigenvalue decomposition $\boldsymbol{G}^\top$ is pre-calculated before training.

CGS. Convergent graph solver (CGS) is an implicit graph neural network proposed by Park et al. [43] where the fixed point equation in use can be described as follows:

$$\boldsymbol{Z}^{(k+1)} = \gamma\boldsymbol{Z}^{(k)}\boldsymbol{G}_\theta + g_{\boldsymbol{B}}(\boldsymbol{X}) \tag{15}$$

where $\boldsymbol{Z}^{(\cdot)}$ is the hidden feature, γ is the contraction factor, $\boldsymbol{G}_\theta \in \mathbb{R}^{n\times n}$ is the graph-related matrix that is learnable and $g_{\boldsymbol{B}}(\boldsymbol{X})$ is the input-dependent bias term. Similar to the EIGNN case, the linearity in Equation (15) allows the fixed point to be found by a closed formula.

GIND. The optimization-induced graph implicit nonlinear diffusion (GIND) is an implicit graph neural network proposed by Chen et al. [15]. GIND involves a fixed point iteration equation of the following form:

$$\boldsymbol{Z}^{(k+1)} = -\boldsymbol{W}^\top\sigma(\boldsymbol{W}(\boldsymbol{Z}^{(k)} + g_{\boldsymbol{B}}(\boldsymbol{X}))\boldsymbol{G})\boldsymbol{G}^\top, \tag{16}$$

where $\boldsymbol{Z}^{(\cdot)}$ is the hidden feature, $\boldsymbol{W}$ is the weight matrix, $g_{\boldsymbol{B}}(\boldsymbol{X})$ is some input-dependent bias term, and $\boldsymbol{G}$ is a normalization of the adjacency matrix $\boldsymbol{A}$. The precise definition of $\boldsymbol{G}$ is given as $\boldsymbol{G} := \hat{\boldsymbol{D}}^{-1/2}\boldsymbol{A}/\sqrt{2}$ where $\hat{\boldsymbol{D}}$ is the degree matrix of the augmented adjacency matrix $\boldsymbol{A}+\boldsymbol{I}$ given as $\hat{D}_{ii} := 1 + \sum_j A_{ij}$. The weight matrix $\boldsymbol{W}$ is parameterized so that $\|\boldsymbol{W}\|\|\boldsymbol{G}\| < 1$. Similar to IGNN, the Picard iteration is employed to find the fixed point. The authors have claimed that the new fixed-point equation (Equation (16)) represents a nonlinear diffusion process with anisotropic properties while IGNN only represents a linear isotropic diffusion. However, we observe that GIND is closely related to the following simple variant of IGNN where the main change is to

$$\boldsymbol{Z}^{(k+1)} = \sigma\left(\boldsymbol{W}(-\boldsymbol{W}^\top)\boldsymbol{Z}^{(k)}\boldsymbol{G}^\top\boldsymbol{G} + \boldsymbol{W}g_{\boldsymbol{B}}(\boldsymbol{X})\boldsymbol{G}\right) \tag{17}$$

where the notations are the same as in Equation (16). In fact, once $\|\boldsymbol{W}\|\|\boldsymbol{G}\| < 1$ and assume σ is a non-expansive activation function (for example, tanh, ReLU, ELU), then Equation (17) is contractive and hence its fixed point exists. Let $\boldsymbol{Z}^*$ be the fixed-point of Equation (17), then we claim that $\tilde{\boldsymbol{Z}} = -\boldsymbol{W}^\top \boldsymbol{Z}^* \boldsymbol{G}^\top$ is the fixed point of Equation (16) with the same $\boldsymbol{W}$, $\boldsymbol{G}$, and $g_{\boldsymbol{B}}(\boldsymbol{X})$ used in both Equation (17) and Equation (16). This can be seen from the following direct calculation:

$$\begin{aligned}
\tilde{\boldsymbol{Z}} &= -\boldsymbol{W}^\top \boldsymbol{Z}^* \boldsymbol{G}^\top \\
&= -\boldsymbol{W}^\top \sigma\left(\boldsymbol{W}(-\boldsymbol{W}^\top)\boldsymbol{Z}^*\boldsymbol{G}^\top\boldsymbol{G} + \boldsymbol{W} g_{\boldsymbol{B}}(\boldsymbol{X})\boldsymbol{G}\right)\boldsymbol{G}^\top \\
&= -\boldsymbol{W}^\top \sigma\left(\boldsymbol{W}\tilde{\boldsymbol{Z}}\boldsymbol{G} + \boldsymbol{W} g_{\boldsymbol{B}}(\boldsymbol{X})\boldsymbol{G}\right)\boldsymbol{G}^\top \\
&= -\boldsymbol{W}^\top \sigma\left(\boldsymbol{W}(\tilde{\boldsymbol{Z}} + g_{\boldsymbol{B}}(\boldsymbol{X}))\boldsymbol{G}\right)\boldsymbol{G}^\top.
\end{aligned}$$

B. A Brief Review of Monotone Operator Theory

B.1. Operators

In this section, we briefly review the definition and basic theory of monotone operators; more details can be found in [48]. We say $\mathcal{T}$ is a *(set-valued) operator* if $\mathcal{T}$ maps a point in $\mathbb{R}^d$ to a subset of $\mathbb{R}^d$. and we denote this as $\mathcal{T}: \mathbb{R}^d \rightrightarrows \mathbb{R}^d$. We define the graph of an operator as follows:

$$\operatorname{Gra}\mathcal{T} = \{(\boldsymbol{x}, \boldsymbol{u}) | \boldsymbol{u} \in \mathcal{T}(\boldsymbol{x})\}.$$

Mathematically, an operator and its graph are equivalent. In other words, we can view $\mathcal{T}: \mathbb{R}^d \rightrightarrows \mathbb{R}^d$ as a point-to-set mapping and as a subset of $\mathbb{R}^d \times \mathbb{R}^d$.

Many notions for functions can be extended to operators. For example, the domain and range of an operator $\mathcal{T}$ are defined as

$$\operatorname{dom}\mathcal{T} = \{\boldsymbol{x} \mid \mathcal{T}(\boldsymbol{x}) \neq \emptyset\}, \quad \operatorname{range}\mathcal{T} = \{\boldsymbol{y} \mid \boldsymbol{y} = \mathcal{T}(\boldsymbol{x}), \boldsymbol{x} \in \mathbb{R}^d\}.$$

If $\mathcal{T}$ and $\mathcal{S}$ are two operators, we define their composition as

$$\mathcal{T} \circ \mathcal{S}(\boldsymbol{x}) = \mathcal{T}\mathcal{S}(\boldsymbol{x}) = \mathcal{T}(\mathcal{S}(\boldsymbol{x})),$$

and their sum as

$$(\mathcal{T} + \mathcal{S})(\boldsymbol{x}) = \mathcal{T}(\boldsymbol{x}) + \mathcal{S}(\boldsymbol{x}).$$

Alternatively, we can define the operator composition and sum using their graphs,

$$\mathcal{T}\mathcal{S} = \big\{(\boldsymbol{x}, \boldsymbol{z}) \mid \exists\, \boldsymbol{y}\ (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}, (\boldsymbol{y}, \boldsymbol{z}) \in \mathcal{T}\big\},$$

$$\mathcal{T} + \mathcal{S} = \big\{(\boldsymbol{x}, \boldsymbol{y} + \boldsymbol{z}) \mid (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{T}, (\boldsymbol{x}, \boldsymbol{z}) \in \mathcal{S}\big\}.$$

The identity ($\mathcal{I}$) and zero ($\mathbf{0}$) operators are defined as follows

$$\mathcal{I} = \{(\boldsymbol{x}, \boldsymbol{x}) \mid \boldsymbol{x} \in \mathbb{R}^d\}, \quad \mathbf{0} = \{(\boldsymbol{x}, \mathbf{0}) \mid \boldsymbol{x} \in \mathbb{R}^d\}.$$

We say an operator $\mathcal{T}$ is L-Lipschitz ($L > 0$) if

$$\|\mathcal{T}(\boldsymbol{x}) - \mathcal{T}(\boldsymbol{y})\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|,\ \forall \boldsymbol{x}, \boldsymbol{y} \in \operatorname{dom}\mathcal{T},$$

i.e.

$$\|\boldsymbol{u} - \boldsymbol{v}\| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|, \quad \forall (\boldsymbol{x}, \boldsymbol{u}), (\boldsymbol{y}, \boldsymbol{v}) \in \mathcal{T}.$$

The *inverse operator* of $\mathcal{T}$ is defined as

$$\mathcal{T}^{-1} = \{(\boldsymbol{y}, \boldsymbol{x}) \mid (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{T}\}.$$

When $\mathbf{0} \in \mathcal{T}(\boldsymbol{x})$, we say that $\boldsymbol{x}$ is a *zero* of $\mathcal{T}$. We write the zero set of an operator $\mathcal{T}$ as

$$\operatorname{Zer}\mathcal{T} = \{\boldsymbol{x} \mid \mathbf{0} \in \mathcal{T}(\boldsymbol{x})\} = \mathcal{T}^{-1}(\mathbf{0}).$$

B.2. Monotone operators

An operator $\mathcal{T}$ on $\mathbb{R}^d$ is said to be *monotone* if

$$\langle \boldsymbol{u}-\boldsymbol{v}, \boldsymbol{x}-\boldsymbol{y}\rangle \geq 0, \quad \forall(\boldsymbol{x},\boldsymbol{u}),(\boldsymbol{y},\boldsymbol{v}) \in \mathcal{T},$$

where $\langle\cdot,\cdot\rangle$ denotes the inner product between two vectors. Equivalently, we can express monotonicity as

$$\langle \mathcal{T}(\boldsymbol{x})-\mathcal{T}(\boldsymbol{y}), \boldsymbol{x}-\boldsymbol{y}\rangle \geq 0, \quad \forall \boldsymbol{x},\boldsymbol{y} \in \mathbb{R}^d.$$

Furthermore, we say the operator $\mathcal{T}$ is *maximal monotone* if there is no other monotone operator $\mathcal{S}$ s.t. $\operatorname{Gra}\mathcal{T} \subset \operatorname{Gra}\mathcal{S}$ properly. In other words, if the monotone operator $\mathcal{T}$ is not maximal, then there exists $(\boldsymbol{x},\boldsymbol{u}) \notin \mathcal{T}$ s.t. $\mathcal{T}\cup\{(\boldsymbol{x},\boldsymbol{u})\}$ is still monotone. A continuous monotone function $\mathcal{F}:\mathbb{R}^d \to \mathbb{R}^d$ is maximal monotone.

An operator $\mathcal{T}:\mathbb{R}^d \rightrightarrows \mathbb{R}^d$ is B-*strongly monotone* or B-coercive if $B>0$ and

$$\langle \boldsymbol{u}-\boldsymbol{v}, \boldsymbol{x}-\boldsymbol{y}\rangle \geq B\|\boldsymbol{x}-\boldsymbol{y}\|^2, \quad \forall(\boldsymbol{x},\boldsymbol{u}),(\boldsymbol{y},\boldsymbol{v}) \in \mathcal{T}.$$

We say $\mathcal{T}$ is strongly monotone if it is B-strongly monotone for some unspecified constant $B \in (0,\infty)$. In particular, a linear operator $\mathcal{F}(\boldsymbol{x})=\boldsymbol{G}\boldsymbol{x}+\boldsymbol{h}$ for $\boldsymbol{G}\in\mathbb{R}^{d\times d}$ and $\boldsymbol{h}\in\mathbb{R}^d$ is maximal monotone if and only if $\boldsymbol{G}+\boldsymbol{G}^\top \succeq \boldsymbol{0}$ ($\boldsymbol{0}$ stands for the matrix whose entries are all zero) and B-strongly monotone if $\frac{1}{2}(\boldsymbol{G}+\boldsymbol{G}^\top)\succeq B\boldsymbol{I}$. Similarly, a subdifferentiable operator ∂f is maximal monotone if and only if f is a convex closed proper (CCP) function.

An operator $\mathcal{T}$ is β-*cocoercive* or β-*inverse strongly monotone* if $\beta>0$ and

$$\langle \boldsymbol{u}-\boldsymbol{v}, \boldsymbol{x}-\boldsymbol{y}\rangle \geq \beta\|\boldsymbol{u}-\boldsymbol{v}\|^2, \quad \forall(\boldsymbol{x},\boldsymbol{u}),(\boldsymbol{y},\boldsymbol{v}) \in \mathcal{T}.$$

We say $\mathcal{T}$ is cocoercive if it is β-cocoercive for some unspecified constant $\beta\in(0,\infty)$. In particular, if the linear operator $\mathcal{F}(\boldsymbol{x})=\boldsymbol{G}\boldsymbol{x}+\boldsymbol{h}$ is B-strongly monotone and L-Lipschitz, then $\mathcal{F}$ is $\frac{B}{L^2}$-cocoercive.

C. A Brief Review of Operator Splitting Schemes

In this section, we provide a brief review of a few celebrated operator-splitting schemes for solving fixed-point equilibrium equations.

C.1. Resolvent and Cayley operators

The *resolvent* and *Cayley* operators of an operator $\mathcal{T}$ is defined, respectively, as follows

$$\mathcal{R}_{\mathcal{T}}=(\mathcal{I}+\alpha\mathcal{T})^{-1},$$

and

$$\mathcal{C}_{\mathcal{T}}=2\mathcal{R}_{\mathcal{T}}-\mathcal{I},$$

where $\alpha>0$ is a constant. The resolvent and Cayley operators are both non-expansive, i.e. they both have Lipschitz constant $L\leq 1$ for any maximal monotone operator $\mathcal{T}$, and the resolvent operator $\mathcal{R}_{\mathcal{T}}$ is contractive (i.e. $L<1$) for strongly monotone $\mathcal{T}$, the Cayley operator $\mathcal{C}_{\mathcal{T}}$ is contractive for strongly monotone and Lipschitz $\mathcal{T}$.

There are two well-known properties associated with the resolvent operators:

- First, when $\mathcal{F}(\boldsymbol{x})=\boldsymbol{G}\boldsymbol{x}+\boldsymbol{h}$ is a linear operator, then

$$\mathcal{R}_{\mathcal{F}}(\boldsymbol{x})=\big(\boldsymbol{I}+\alpha\boldsymbol{G}\big)^{-1}(\boldsymbol{x}-\alpha\boldsymbol{h}).$$

- Second, when $\mathcal{F}=\partial f$ for some CCP function f, then the resolvent is given by the following proximal operator

$$\mathcal{R}_{\mathcal{F}}(\boldsymbol{x})=\operatorname{prox}_f^\alpha(\boldsymbol{x}):=\arg\min_{\boldsymbol{z}}\Big\{\frac{1}{2}\|\boldsymbol{x}-\boldsymbol{z}\|^2+\alpha f(\boldsymbol{z})\Big\}.$$

C.2. Operator splitting schemes

Operator splitting schemes refer to methods to find a zero in a sum of operators (assumed here to be maximal monotone), i.e. find $\boldsymbol{x}$ s.t.

$$\mathbf{0} \in (\mathcal{F}+\mathcal{G})(\boldsymbol{x}).$$

We present a few popular operator splitting schemes for solving the above monotone inclusion problem.

- *Forward-backward splitting (FB)*: Consider the monotone inclusion problem

  $$\operatorname{find}_{\boldsymbol{x}\in\mathbb{R}^d}\ \mathbf{0} \in (\mathcal{F}+\mathcal{G})(\boldsymbol{x}),$$

  where $\mathcal{F}$ and $\mathcal{G}$ are maximal monotone and $\mathcal{F}$ is single-valued. Then for any $\alpha > 0$, we have

  $$\begin{aligned}\mathbf{0} \in (\mathcal{F}+\mathcal{G})(\boldsymbol{x}) &\Leftrightarrow \mathbf{0} \in (\mathcal{I}+\alpha\mathcal{G})(\boldsymbol{x}) - (\mathcal{I}-\alpha\mathcal{F})(\boldsymbol{x})\\ &\Leftrightarrow (\mathcal{I}+\alpha\mathcal{G})(\boldsymbol{x}) \ni (\mathcal{I}-\alpha\mathcal{F})(\boldsymbol{x})\\ &\Leftrightarrow \boldsymbol{x} = \mathcal{R}_{\mathcal{G}}(\mathcal{I}-\alpha\mathcal{F})(\boldsymbol{x}).\end{aligned}$$

  Therefore, $\boldsymbol{x}$ is a solution if and only if it is a fixed point of $\mathcal{R}_{\mathcal{G}}(\mathcal{I}-\alpha\mathcal{F})$. Moreover, assume $\mathcal{F}$ is β-cocoercive, then the Picard iteration using forward-backward splitting can be written as

  $$\boldsymbol{x}^{(k+1)} = \mathcal{R}_{\mathcal{G}}(\boldsymbol{x}^{(k)} - \alpha\mathcal{F}\boldsymbol{x}^{(k)}),$$

  which converges if $\alpha \in (0, 2\beta)$ and $\operatorname{Zer}(\mathcal{F}+\mathcal{G}) \neq \emptyset$.

- *Peaceman-Rachford splitting (PR)*: Consider the following monotone inclusion problem

  $$\operatorname{find}_{\boldsymbol{x}\in\mathbb{R}^d}\ \mathbf{0} \in (\mathcal{F}+\mathcal{G})(\boldsymbol{x}),$$

  where $\mathcal{F}$ and $\mathcal{G}$ are maximal monotone. For any $\alpha > 0$, we have

  $$\begin{aligned}\mathbf{0} \in (\mathcal{F}+\mathcal{G})(\boldsymbol{x}) &\Leftrightarrow \mathbf{0} \in (\mathcal{I}+\alpha\mathcal{F})(\boldsymbol{x}) - (\mathcal{I}-\alpha\mathcal{G})(\boldsymbol{x})\\ &\Leftrightarrow \mathbf{0} \in (\mathcal{I}+\alpha\mathcal{F})(\boldsymbol{x}) - \mathcal{C}_{\mathcal{G}}(\mathcal{I}+\alpha\mathcal{G})(\boldsymbol{x})\\ &\Leftrightarrow \mathbf{0} \in (\mathcal{I}+\alpha\mathcal{F})(\boldsymbol{x}) - \mathcal{C}_{\mathcal{G}}(\boldsymbol{z}),\ \boldsymbol{z} \in (\mathcal{I}+\alpha\mathcal{G})(\boldsymbol{x})\\ &\Leftrightarrow \mathcal{C}_{\mathcal{G}}(\boldsymbol{z}) \in (\mathcal{I}+\alpha\mathcal{F})\mathcal{R}_{\mathcal{G}}(\boldsymbol{z}),\ \boldsymbol{x} = \mathcal{R}_{\mathcal{G}}(\boldsymbol{z})\\ &\Leftrightarrow \mathcal{R}_{\mathcal{F}}\mathcal{C}_{\mathcal{G}}(\boldsymbol{z}) = \mathcal{R}_{\mathcal{G}}(\boldsymbol{z}),\ \boldsymbol{x} = \mathcal{R}_{\mathcal{G}}(\boldsymbol{z})\\ &\Leftrightarrow \mathcal{C}_{\mathcal{F}}\mathcal{C}_{\mathcal{G}}(\boldsymbol{z}) = \boldsymbol{z},\ \boldsymbol{x} = \mathcal{R}_{\mathcal{G}}(\boldsymbol{z}).\end{aligned}$$

  Therefore, $\boldsymbol{x}$ is a solution if and only if there is a solution of the fixed-point equilibrium equation $\boldsymbol{z} = \mathcal{C}_{\mathcal{F}}\mathcal{C}_{\mathcal{G}}(\boldsymbol{z})$ and $\boldsymbol{x} = \mathcal{R}_{\mathcal{G}}(\boldsymbol{z})$, which is called *Peaceman-Rachford splitting*.

- *Douglas-Rachford splitting (DR)*: Sometimes the operator $\mathcal{C}_{\mathcal{F}}\mathcal{C}_{\mathcal{G}}$ is merely nonexpansive, the Picard iteration with PR given below

  $$\boldsymbol{z}^{(k+1)} = \mathcal{C}_{\mathcal{F}}\mathcal{C}_{\mathcal{G}}(\boldsymbol{z}^{(k)})$$

  is not guaranteed to converge. To guarantee convergence, we note that for any $\forall\alpha > 0$, we have

  $$\mathbf{0} \in (\mathcal{F}+\mathcal{G})(\boldsymbol{x}) \ \Leftrightarrow\ \Big(\frac{1}{2}\mathcal{I} + \frac{1}{2}\mathcal{C}_{\mathcal{F}}\mathcal{C}_{\mathcal{G}}\Big)(\boldsymbol{z}) = \boldsymbol{z},\ \boldsymbol{x} = \mathcal{J}_{\mathcal{G}}(\boldsymbol{z}).$$

  And the above splitting is called *Douglas-Rachford splitting*. The Picard iteration with DR can be written as follows:

  $$\begin{aligned}\boldsymbol{x}^{(k+1/2)} &= \mathcal{R}_{\mathcal{G}}(\boldsymbol{z}^{(k)})\\ \boldsymbol{x}^{(k+1)} &= \mathcal{R}_{\mathcal{F}}(2\boldsymbol{x}^{(k+1/2)} - \boldsymbol{z}^{(k)})\\ \boldsymbol{z}^{(k+1)} &= \boldsymbol{z}^{(k)} + \boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k+1/2)}\end{aligned}$$

  which converges for any $\alpha > 0$ if $\operatorname{Zer}(\mathcal{F}+\mathcal{G}) \neq \emptyset$.

D. Some Properties of Kronecker product

In this section, we collect some Kronecker product results that are used in this paper.

Definition D.1. Let $\boldsymbol{A} \in \mathbb{R}^{p\times q}$, $\boldsymbol{B} \in \mathbb{R}^{r\times s}$ be two matrices. Their Kronecker product $\boldsymbol{A} \otimes \boldsymbol{B} \in \mathbb{R}^{pr\times qs}$ is defined as follows:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} A_{11}\boldsymbol{B} & \dots & A_{1q}\boldsymbol{B} \\ \vdots & & \vdots \\ A_{p1}\boldsymbol{B} & \dots & A_{pq}\boldsymbol{B} \end{bmatrix}$$

The following identities about Kronecker product hold:

- $(\boldsymbol{A} \otimes \boldsymbol{B})^\top = \boldsymbol{A}^\top \otimes \boldsymbol{B}^\top \quad \forall \boldsymbol{A} \in \mathbb{R}^{p\times q}, B \in \mathbb{R}^{r\times s}$
- $\|\boldsymbol{A} \otimes \boldsymbol{B}\| = \|\boldsymbol{A}\|\|\boldsymbol{B}\| \quad \forall \boldsymbol{A} \in \mathbb{R}^{p\times q}, B \in \mathbb{R}^{r\times s}$
- $\|\boldsymbol{A} \otimes \boldsymbol{B}\|_\infty = \|\boldsymbol{A}\|_\infty\|\boldsymbol{B}\|_\infty \quad \forall \boldsymbol{A} \in \mathbb{R}^{p\times q}, B \in \mathbb{R}^{r\times s}$
- $(\boldsymbol{A} \otimes \boldsymbol{B})\mathrm{vec}(\boldsymbol{C}) = \mathrm{vec}(\boldsymbol{B}\boldsymbol{C}\boldsymbol{A}^\top) \quad \forall \boldsymbol{A} \in \mathbb{R}^{s,r}, \boldsymbol{B} \in \mathbb{R}^{p\times q}, \boldsymbol{C} \in \mathbb{R}^{q\times r}$
- $(\boldsymbol{A} \otimes \boldsymbol{B}) \otimes \boldsymbol{C} = \boldsymbol{A} \otimes (\boldsymbol{B} \otimes \boldsymbol{C}) \quad \forall \boldsymbol{A} \in \mathbb{R}^{m,n}, \boldsymbol{B} \in \mathbb{R}^{p\times q}, \boldsymbol{C} \in \mathbb{R}^{r\times s}$
- $\boldsymbol{A} \otimes (\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{A} \otimes \boldsymbol{B} + \boldsymbol{A} \otimes \boldsymbol{C} \quad \forall \boldsymbol{A} \in \mathbb{R}^{p\times q}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{r\times s}$
- $(\boldsymbol{A} + \boldsymbol{B}) \otimes \boldsymbol{C} = \boldsymbol{A} \otimes \boldsymbol{C} + \boldsymbol{B} \otimes \boldsymbol{C} \quad \forall \boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{p\times q}, \boldsymbol{C} \in \mathbb{R}^{r\times s}$
- $(\boldsymbol{A} \otimes \boldsymbol{B})(\boldsymbol{C} \otimes \boldsymbol{D}) = \boldsymbol{A}\boldsymbol{C} \otimes \boldsymbol{B}\boldsymbol{D} \quad \forall \boldsymbol{A} \in \mathbb{R}^{p\times q}, \boldsymbol{B} \in \mathbb{R}^{r\times s}, \boldsymbol{C} \in \mathbb{R}^{q\times k}, \boldsymbol{D} \in \mathbb{R}^{s\times l}$

Proposition D.2 (Theorem 4.2.12 in [27])**.** *Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{B} \in \mathbb{R}^{m\times m}$. If we denote the eigenvalue sets of $\boldsymbol{A}$ and $\boldsymbol{B}$ as $\Lambda(\boldsymbol{A}) = \{\lambda_1(\boldsymbol{A}), \dots, \lambda_n(\boldsymbol{A})\}$ and $\Lambda(\boldsymbol{B}) = \{\lambda_1(\boldsymbol{B}), \dots, \lambda_m(\boldsymbol{B})\}$, then the eigenvalue set of $\boldsymbol{A} \otimes \boldsymbol{B}$ is $\Lambda(\boldsymbol{A} \otimes \boldsymbol{B}) = \{\lambda_i(\boldsymbol{A}) \cdot \lambda_j(\boldsymbol{B}), i = 1, \dots, n, j = 1, \dots, m\}$.*

E. Technical Proofs

E.1. Lipschitz constant vs. Largest magnitude of eigenvalue

Let $f(\boldsymbol{Z}) = \boldsymbol{W}\boldsymbol{Z}\boldsymbol{G} + \boldsymbol{B}$ be a linear map. With slightly abuse of notation, we still denote the vectorized version of f as f which reads $f(\mathrm{vec}(\boldsymbol{Z})) = (\boldsymbol{G}^\top \otimes \boldsymbol{W})\mathrm{vec}(\boldsymbol{Z}) + \mathrm{vec}(\boldsymbol{B})$ (See Appendix D for properties of the Kronecker product). The Lipschitz constant $\mathrm{Lip}_\infty(f)$ of the linear map f with respect to the ℓ_∞ vector norm is exactly the ∞-norm $\|\boldsymbol{G} \otimes \boldsymbol{W}\|_\infty = \|\boldsymbol{G}^\top\|_\infty\|\boldsymbol{W}\|_\infty$. Recall the following general result about the matrix norm and the largest magnitude of eigenvalue.

Theorem E.1 (Theorem 4 in Section 4.6 in [31])**.** *The largest magnitude of eigenvalue $\lambda_1(\boldsymbol{A})$ of a matrix $\boldsymbol{A}$ satisfies*

$$\lambda_1(\boldsymbol{A}) = \inf_{\|\cdot\|_{\mathrm{M}}} \|\boldsymbol{A}\|_{\mathrm{M}}$$

in which the infimum is taken over all subordinate matrix norms $\|\cdot\|_{\mathrm{M}}$ including 2-norm and ∞-norm.

Meanwhile, note that one has $\|\boldsymbol{W}\|_\infty = \||\boldsymbol{W}|\|_\infty$ by definition. Hence one has $\mathrm{Lip}_\infty(f) = \|\boldsymbol{G}^\top\|_\infty\|\boldsymbol{W}\|_\infty \geq \lambda_1(\boldsymbol{G}^\top)\lambda_1(|\boldsymbol{W}|)$. Note that, when $\boldsymbol{G}$ is the normalized adjacency matrix of undirected graph $\hat{\boldsymbol{A}}$, we have $\lambda_1(\boldsymbol{G}^\top) = \lambda_1(\boldsymbol{G}) = 1$ and hence we have $\mathrm{Lip}_\infty(f) \geq \lambda_1(|\boldsymbol{W}|)$.

E.2. Proofs for Section 2

Proof of Proposition 2.1. First recall the operator splitting problem 3 in Section 1:

$$\text{find } \mathbf{0} \in (\mathcal{F} + \mathcal{G})(\mathrm{vec}(\boldsymbol{Z})),$$

where

$$\mathcal{F}(\mathrm{vec}(\boldsymbol{Z})) = (\boldsymbol{I} - \boldsymbol{G}^\top \otimes \boldsymbol{W})\mathrm{vec}(\boldsymbol{Z}) - \mathrm{vec}(g_{\boldsymbol{B}}(\boldsymbol{X})) \text{ and } \mathcal{G} = \partial f,$$

here f is the indicator of the positive octant, i.e. $f(x) = I\{x \geq 0\}$ for which we have prox_f^α equals σ, the ReLU activation function, for all $\alpha > 0$. Note that, from the condition $\boldsymbol{K} = \frac{1}{2}\big(\boldsymbol{G}^\top \otimes \boldsymbol{W} + \boldsymbol{G} \otimes \boldsymbol{W}^\top\big) \preceq (1-m)\boldsymbol{I}$, one has $\boldsymbol{G}^\top \otimes \boldsymbol{W} \preceq (1-m)\boldsymbol{I}$ and hence

$$\boldsymbol{I} - \boldsymbol{G}^\top \otimes \boldsymbol{W} \succeq m\boldsymbol{I}$$

which says $\mathcal{F}$ is m-strongly monotone for some $m > 0$. As the function $\mathcal{F}$ is a linear and hence continuous function defined on the entire $\mathbb{R}^{d \times n}$, it is then automatically maximal monotone once it is monotone. Since f is a CCP function, its subdifferential operator $\mathcal{G} = \partial f$ is maximal monotone. In particular, as the linear map $\mathcal{F}$ is single-valued, we can apply the FB splitting scheme in Appendix C.2 as the following: for any $\alpha > 0$, we have

$$\begin{aligned}
\boldsymbol{0} \in (\mathcal{F} + \mathcal{G})(\mathrm{vec}(\boldsymbol{Z})) &\Leftrightarrow \mathrm{vec}(\boldsymbol{Z}) = \mathcal{R}_{\mathcal{G}}(\mathcal{I} - \alpha\mathcal{F})(\mathrm{vec}(\boldsymbol{Z})). \\
&\Leftrightarrow \mathrm{vec}(\boldsymbol{Z}) = \mathrm{prox}_f^\alpha\left(\mathrm{vec}(\boldsymbol{Z}) - \alpha \cdot \left(\mathrm{vec}(\boldsymbol{Z}) - \boldsymbol{G}^\top \otimes \boldsymbol{W}\mathrm{vec}(\boldsymbol{Z}) - \mathrm{vec}(g_{\boldsymbol{B}}(\boldsymbol{X}))\right)\right), \\
&\Leftrightarrow \mathrm{vec}(\boldsymbol{Z}) = \sigma\left(\mathrm{vec}(\boldsymbol{Z}) - \alpha \cdot \left(\mathrm{vec}(\boldsymbol{Z}) - \boldsymbol{G}^\top \otimes \boldsymbol{W}\mathrm{vec}(\boldsymbol{Z}) - \mathrm{vec}(g_{\boldsymbol{B}}(\boldsymbol{X}))\right)\right).
\end{aligned}$$

When $\alpha = 1$ in the last above, we recover the MIGNN model Equation (2):

$$\mathrm{vec}(\boldsymbol{Z}) = \sigma(\boldsymbol{G}^\top \otimes \boldsymbol{W}\mathrm{vec}(\boldsymbol{Z}) + \mathrm{vec}(g_{\boldsymbol{B}}(\boldsymbol{X}))$$

This shows the equivalence between finding a fixed point of MIGNN model Equation (2) and finding a zero of the operator splitting problem Equation (3). Therefore, when $\boldsymbol{K} \preceq (1-m)\boldsymbol{I}$, the linear map $\mathcal{F}$ is strongly monotone and Lipschitz, the monotone splitting problem and hence the MIGNN model is well-sposed, see Appendix C.2. □

E.3. Proofs for Section 3

Proof of Proposition 3.1. Since the normalized Laplacian $\boldsymbol{L}$ is symmetric, we have

$$\boldsymbol{K} = \frac{1}{2}\left(\frac{1}{2}\boldsymbol{L}^\top \otimes \boldsymbol{W} + \frac{1}{2}\boldsymbol{L} \otimes \boldsymbol{W}^\top\right) = \frac{1}{2}\boldsymbol{L} \otimes \left(\frac{1}{2}\left(\boldsymbol{W} + \boldsymbol{W}^\top\right)\right).$$

The property of Kronecker product (Theorem D.2) tells us that the eigenvalues of $\boldsymbol{K}$ are the products of the eigenvalues of $\boldsymbol{L}$ and $\left(\frac{1}{2}(\boldsymbol{W} + \boldsymbol{W}^\top)\right)$. Therefore, the MIGNN model satisfies the well-posedness condition in Proposition 2.1 once

$$\lambda_i\left(\frac{1}{2}\boldsymbol{L}\right)\lambda_j\left(\frac{1}{2}(\boldsymbol{W} + \boldsymbol{W}^\top)\right) \leq 1 - m$$

for all eigenvalues from $\frac{1}{2}\boldsymbol{L}$ and $\left(\frac{1}{2}(\boldsymbol{W} + \boldsymbol{W}^\top)\right)$. Notice that $\frac{1}{2}\boldsymbol{L}$ is positive semi-definite and all its eigenvalues are within $[0, 1]$. Therefore, $\boldsymbol{W}$ guarantees the well-posedness of MIGNN as long as all eigenvalues satisfy

$$\lambda_i\left(\frac{1}{2}(\boldsymbol{W} + \boldsymbol{W}^\top)\right) \leq 1 - m.$$

When $\boldsymbol{W} = (1-m)\boldsymbol{I} - \boldsymbol{C}\boldsymbol{C}^\top + \boldsymbol{F} - \boldsymbol{F}^\top$, we have $\frac{1}{2}(\boldsymbol{W} + \boldsymbol{W}^\top) = (1-m)\boldsymbol{I} - \boldsymbol{C}\boldsymbol{C}^\top$. As $\boldsymbol{C}\boldsymbol{C}^\top$ is positive semi-definite, all eigenvalues of $\frac{1}{2}(\boldsymbol{W} + \boldsymbol{W}^\top)$ are no more than $(1-m)$. □

The following properties of the Cayley map are used in this paper.

Proposition E.2. *Let $\boldsymbol{S}$ be a skew-symmetric matrix. Then its image under the Cayley map* $\mathrm{Cay}(\boldsymbol{S}) := (\boldsymbol{I} - \boldsymbol{S})(\boldsymbol{I} + \boldsymbol{S})^{-1}$ *is an orthogonal matrix, and hence the magnitude of all its eigenvalues is* 1.

Proof. To verify that the Cayley map is well-defined, it suffices to show that -1 is not an eigenvalue of $\boldsymbol{S}$. This can be derived from the general fact that each eigenvalue of any skew-symmetric matrix is purely imaginary. To see this, let λ be an eigenvalue of $\boldsymbol{S}$ with corresponding eigenvector $\boldsymbol{v}$ where both λ and $\boldsymbol{v}$ possibly contain complex numbers. Let $\boldsymbol{v}^{\mathrm{H}}$ and $\boldsymbol{S}^{\mathrm{H}}$ denote the conjugate transpose of the vector $\boldsymbol{v}$ and the matrix $\boldsymbol{S}$ respectively. We then have

$$\boldsymbol{v}^{\mathrm{H}}\boldsymbol{S}\boldsymbol{v} = \boldsymbol{v}^{\mathrm{H}}(\lambda\boldsymbol{v}) = \lambda|\boldsymbol{v}|_{\mathbb{C}}^2,$$

where $|\cdot|_{\mathbb{C}}$ denotes the Euclidean norm for a complex vector. At the same time, one has

$$\boldsymbol{v}^{\mathrm{H}}\boldsymbol{S}\boldsymbol{v} = (\boldsymbol{S}^{\mathrm{H}}\boldsymbol{v})^{\mathrm{H}}\boldsymbol{v} = (-\boldsymbol{S}\boldsymbol{v})^{\mathrm{H}}\boldsymbol{v} = -\bar{\lambda}|\boldsymbol{v}|_{\mathbb{C}}^2,$$

where $\bar{\lambda}$ denotes the complex conjugate of λ. Hence $\lambda = -\lambda$, that is λ is purely imaginary. This concludes the proof that $(\boldsymbol{I} - \boldsymbol{S})(\boldsymbol{I} + \boldsymbol{S})^{-1}$ is well-defined.

Note that $(\boldsymbol{I}-\boldsymbol{S})(\boldsymbol{I}+\boldsymbol{S})^{-1}\left((\boldsymbol{I} - \boldsymbol{S})(\boldsymbol{I} + \boldsymbol{S})^{-1}\right)^\top = (\boldsymbol{I}-\boldsymbol{S})(\boldsymbol{I}+\boldsymbol{S})^{-1}(\boldsymbol{I}+\boldsymbol{S})(\boldsymbol{I}-\boldsymbol{S})^{-1} = \boldsymbol{I}$. Therefore, $(\boldsymbol{I}-\boldsymbol{S})(\boldsymbol{I}+\boldsymbol{S})^{-1}$ is (real) orthogonal.

In the last part, we present a short proof that the magnitude of all eigenvalues of a (real) orthogonal matrix $\boldsymbol{O}$ equals 1. Let $\lambda_{\boldsymbol{O}}$ be an eigenvalue of $\boldsymbol{O}$ and $\boldsymbol{w}$ is its eigenvector. Then we have

$$|\lambda_{\boldsymbol{O}}||\boldsymbol{w}|^2_{\mathbb{C}} = (\boldsymbol{O}\boldsymbol{w})^{\mathrm{H}}(\boldsymbol{O}\boldsymbol{w}) = \boldsymbol{w}^{\mathrm{H}}\boldsymbol{O}^{\mathrm{H}}\boldsymbol{O}\boldsymbol{w} = (\boldsymbol{O}\boldsymbol{w})^{\mathrm{H}}(\boldsymbol{O}\boldsymbol{w}) = \boldsymbol{w}^{\mathrm{H}}\boldsymbol{O}^\top\boldsymbol{O}\boldsymbol{w} = |\boldsymbol{w}|^2_{\mathbb{C}}.$$

Hence, $|\lambda_{\boldsymbol{O}}| = 1$. □

E.4. Proofs for Section 4

The following result about Kronecker product is adapted from [39] which we include here for completeness.

Proof of Equation (9) used in Section 4. Since $\boldsymbol{G}^\top$ is symmetric, it admits an eigen-decomposition $\boldsymbol{G}^\top = \boldsymbol{Q}_{\boldsymbol{G}^\top}\boldsymbol{\Lambda}_{\boldsymbol{G}^\top}\boldsymbol{Q}_{\boldsymbol{G}^\top}^\top$ where $\boldsymbol{Q}_{\boldsymbol{G}^\top}$ is orthogonal and hence satisfies $\boldsymbol{Q}_{\boldsymbol{G}^\top}^{-1} = \boldsymbol{Q}_{\boldsymbol{G}^\top}$. As $\boldsymbol{W}$ is diagonalizable, it admits a eigen-decomposition $\boldsymbol{W} = \boldsymbol{Q}_{\boldsymbol{W}}\boldsymbol{\Lambda}_{\boldsymbol{W}}\boldsymbol{Q}_{\boldsymbol{W}}^{-1}$. Then we can write

$$\begin{aligned}\boldsymbol{G}^\top \otimes \boldsymbol{W} &= [\boldsymbol{Q}_{\boldsymbol{G}^\top}\boldsymbol{\Lambda}_{\boldsymbol{G}^\top}\boldsymbol{Q}_{\boldsymbol{G}^\top}^\top] \otimes [\boldsymbol{Q}_{\boldsymbol{W}}\boldsymbol{\Lambda}_{\boldsymbol{W}}\boldsymbol{Q}_{\boldsymbol{W}}^{-1}] \\ &= [\boldsymbol{Q}_{\boldsymbol{G}^\top} \otimes \boldsymbol{Q}_{\boldsymbol{W}}][\boldsymbol{\Lambda}_{\boldsymbol{G}^\top} \otimes \boldsymbol{\Lambda}_{\boldsymbol{W}}][\boldsymbol{Q}_{\boldsymbol{G}^\top}^\top \otimes \boldsymbol{Q}_{\boldsymbol{W}}^{-1}]\end{aligned}$$

Let $n = \dim(\boldsymbol{G})$ and $d = \dim(\boldsymbol{W})$, we have

$$\boldsymbol{I}_{nd} = \boldsymbol{I}_n \otimes \boldsymbol{I}_d = [\boldsymbol{Q}_{\boldsymbol{G}^\top}\boldsymbol{I}_n\boldsymbol{Q}_{\boldsymbol{G}^\top}^\top] \otimes [\boldsymbol{Q}_{\boldsymbol{W}}\boldsymbol{I}_m\boldsymbol{Q}_{\boldsymbol{W}}^{-1}] = [\boldsymbol{Q}_{\boldsymbol{G}^\top} \otimes \boldsymbol{Q}_{\boldsymbol{W}}][\boldsymbol{I}_n \otimes \boldsymbol{I}_m][\boldsymbol{Q}_{\boldsymbol{G}^\top}^\top \otimes \boldsymbol{Q}_{\boldsymbol{W}}^{-1}]$$

Therefore, for some matrix $\boldsymbol{B} \in \mathbb{R}^{d\times n}$,

$$\begin{aligned}\boldsymbol{V}(\mathrm{vec}(\boldsymbol{U})) &= \frac{1}{1+\alpha}\left(\boldsymbol{I}_{nd} - \frac{\alpha}{1+\alpha}(\boldsymbol{G}^\top \otimes \boldsymbol{W})\right)^{-1}(\mathrm{vec}(\boldsymbol{U})) \\ &= \frac{1}{1+\alpha}\left(\boldsymbol{I}_{nd} - \frac{\alpha}{1+\alpha}(\boldsymbol{G}^\top \otimes \boldsymbol{W})\right)^{-1}(\mathrm{vec}(\boldsymbol{U})) \\ &\frac{1}{1+\alpha}\left([\boldsymbol{Q}_{\boldsymbol{G}^\top} \otimes \boldsymbol{Q}_{\boldsymbol{W}}]\left[\boldsymbol{I}_{nd} - \frac{\alpha}{1+\alpha}\boldsymbol{\Lambda}_{\boldsymbol{G}^\top} \otimes \boldsymbol{\Lambda}_{\boldsymbol{W}}\right][\boldsymbol{Q}_{\boldsymbol{G}^\top}^\top \otimes \boldsymbol{Q}_{\boldsymbol{W}}^{-1}]\right)^{-1}(\mathrm{vec}(\boldsymbol{U})) \\ &\frac{1}{1+\alpha}\left([\boldsymbol{Q}_{\boldsymbol{G}^\top} \otimes \boldsymbol{Q}_{\boldsymbol{W}}]\left[\boldsymbol{I}_{nd} - \frac{\alpha}{1+\alpha}\boldsymbol{\Lambda}_{\boldsymbol{G}^\top} \otimes \boldsymbol{\Lambda}_{\boldsymbol{W}}\right]^{-1}[\boldsymbol{Q}_{\boldsymbol{G}^\top}^\top \otimes \boldsymbol{Q}_{\boldsymbol{W}}^{-1}]\right)(\mathrm{vec}(\boldsymbol{U}))\end{aligned}$$

Note that $\left[\boldsymbol{I}_{nd} - \frac{\alpha}{1+\alpha}\boldsymbol{\Lambda}_{\boldsymbol{G}^\top} \otimes \boldsymbol{\Lambda}_{\boldsymbol{W}}\right]$ is a diagonal matrix whose inverse is given by the diagonal matrix $\mathrm{Diag}(\mathrm{vec}(\boldsymbol{H}))$ where the entires of $\boldsymbol{H}$ is given as $H_{ij} := 1/\left(1 - \frac{\alpha}{1+\alpha}(\boldsymbol{\Lambda}_{\boldsymbol{W}})_{ii}(\boldsymbol{\Lambda}_{\boldsymbol{G}^\top})_{jj}\right)$. Here the notation $\mathrm{Diag}(\boldsymbol{v})$ denotes the diagonal matrix that has $\boldsymbol{v}$ as its diagonal for any vector $\boldsymbol{v}$. From this we have,

$$\begin{aligned}\boldsymbol{V}(\mathrm{vec}(\boldsymbol{U})) &= \frac{1}{1+\alpha}\left([\boldsymbol{Q}_{\boldsymbol{G}^\top} \otimes \boldsymbol{Q}_{\boldsymbol{W}}]\,\mathrm{Diag}(\mathrm{vec}(\boldsymbol{H}))\left[\boldsymbol{Q}_{\boldsymbol{G}^\top}^\top \otimes \boldsymbol{Q}_{\boldsymbol{W}}^{-1}\right]\right)(\mathrm{vec}(\boldsymbol{U})) \\ &= \frac{1}{1+\alpha}\left([\boldsymbol{Q}_{\boldsymbol{G}^\top} \otimes \boldsymbol{Q}_{\boldsymbol{W}}]\,\mathrm{Diag}(\mathrm{vec}(\boldsymbol{H}))\,\mathrm{vec}(\boldsymbol{Q}_{\boldsymbol{W}}^{-1}\boldsymbol{U}\boldsymbol{Q}_{\boldsymbol{G}^\top}\right) \\ &= \frac{1}{1+\alpha}[\boldsymbol{Q}_{\boldsymbol{G}^\top} \otimes \boldsymbol{Q}_{\boldsymbol{W}}]\,\mathrm{vec}\left(\boldsymbol{H} \odot [\boldsymbol{Q}_{\boldsymbol{W}}^{-1}\boldsymbol{U}\boldsymbol{Q}_{\boldsymbol{G}^\top}]\right) \\ &= \frac{1}{1+\alpha}\mathrm{vec}\left(\boldsymbol{Q}_{\boldsymbol{W}}[\boldsymbol{H} \odot [\boldsymbol{Q}_{\boldsymbol{W}}^{-1}\boldsymbol{U}\boldsymbol{Q}_{\boldsymbol{G}^\top}]]\boldsymbol{Q}_{\boldsymbol{G}^\top}^\top\right)\end{aligned}$$

where $\odot$ denotes entry-wise multiplication. □

F. MIGNN via Anderson-Accelerated Operator Splitting Schemes

In this section, we present the pseudocodes of Anderson-accelerated MIGNN operator splitting schemes discussed in Section 4.

F.1. Pseudocode for MIGNN with operator splitting schemes

FB Splitting. The detail of the FB splitting scheme iteration function Equation (6) of solving MIGNN is presented in Algorithm 1.

Algorithm 1 FB-forward-MIGNN

```
$\boldsymbol{Z} := 0$;    err $:= 1$
while err $> \epsilon$ do
    $\boldsymbol{Z}^{(+)} := (1-\alpha)\boldsymbol{Z} + \alpha \boldsymbol{W}\boldsymbol{Z}\boldsymbol{G} + \alpha g_{\boldsymbol{B}}(\boldsymbol{X})$
    $\boldsymbol{Z}^{(+)} := \mathrm{prox}_f^\alpha(\boldsymbol{Z}^{(+)})$
    err $:= \frac{\|\boldsymbol{Z}^{(+)} - \boldsymbol{Z}\|}{\|\boldsymbol{Z}^{(+)}\|}$
    $\boldsymbol{Z} := \boldsymbol{Z}^{(+)}$
end while
return $\boldsymbol{Z}$
```

PR splitting. The details of the PR splitting scheme encoded in the iteration function Equation (7) of solving MIGNN is presented in Algorithm 2.

Algorithm 2 PR-forward-MIGNN

```
$\boldsymbol{z}, \boldsymbol{u} = \mathrm{vec}(\boldsymbol{U}) := 0$;    err $:= 1$;    $\boldsymbol{V} := (\boldsymbol{I} + \alpha(\boldsymbol{I} - \boldsymbol{G}^\top \otimes \boldsymbol{W}))^{-1}$
while err $> \epsilon$ do
    $\boldsymbol{z}^{(1/2)} := \mathrm{prox}_f^\alpha(\boldsymbol{u})$
    $\boldsymbol{u}^{(1/2)} := 2\boldsymbol{z}^{(1/2)} - \boldsymbol{u}$
    $\boldsymbol{z}^{(+)} := \boldsymbol{V}(\boldsymbol{u}^{(1/2)} + \alpha\, \mathrm{vec}(g_{\boldsymbol{B}}(\boldsymbol{X})))$
    $\boldsymbol{u}^{(+)} := 2\boldsymbol{z}^{(+)} - \boldsymbol{u}^{(1/2)}$
    err $:= \frac{\|\boldsymbol{u}^{(+)} - \boldsymbol{u}\|}{\|\boldsymbol{u}^{(+)}\|}$
    $\boldsymbol{z}, \boldsymbol{u} := \boldsymbol{z}^{(+)}, \boldsymbol{u}^{(+)}$
end while
return $\mathrm{prox}_f^\alpha(\boldsymbol{u})$
```

F.2. More details on backward propagation

In the backward propagation, the following result from [57] allows us to convert the computing of the inverse Jacobian term $(\boldsymbol{I} - \boldsymbol{J}(\boldsymbol{G}^\top \otimes \boldsymbol{W}))^{-\top}$ to the (transpose of) matrix inverse term $\boldsymbol{V} = (\boldsymbol{I} - \boldsymbol{G}^\top \otimes \boldsymbol{W})^{-1}$ which is already calculated in the forward pass.

Proposition F.1 (Adapted from Theorem 3 in [57])**.** *Let* $\mathrm{vec}(\boldsymbol{Z}^*)$ *be the fixed point of the MIGNN model (2) and* $\boldsymbol{J}$ *is the Jacobian* σ *of the non-linearity at the* $\boldsymbol{G}^\top \otimes \boldsymbol{W}\mathrm{vec}(\boldsymbol{Z}^*) + \mathrm{vec}(g_{\boldsymbol{B}}(\boldsymbol{X}))$*. For any* $\boldsymbol{v} \in \mathbb{R}^n$ *the solution* $\boldsymbol{u}^*$ *of the equation*

$$\boldsymbol{u}^* = (\boldsymbol{I} - \boldsymbol{J}(\boldsymbol{G}^\top \otimes \boldsymbol{W}))^{-\top}\boldsymbol{v}$$

is given by

$$\boldsymbol{u}^* = \boldsymbol{v} + (\boldsymbol{G} \otimes \boldsymbol{W}^\top)\tilde{\boldsymbol{u}}^*$$

where $\tilde{\boldsymbol{u}}$ *is a solution of the operator splitting problem* $0 \in (\tilde{F} + \tilde{G})(\tilde{u})$*, with operators defined as*

$$\tilde{F}(\tilde{\boldsymbol{u}}) = \left(\boldsymbol{I} - \boldsymbol{G} \otimes \boldsymbol{W}^\top\right)(\tilde{\boldsymbol{u}}), \quad \tilde{G}(\tilde{\boldsymbol{u}}) = \boldsymbol{D}\tilde{\boldsymbol{u}} - \boldsymbol{v} \tag{18}$$

where $\boldsymbol{D}$ *is the diagonal matrix defined by* $\boldsymbol{J} = (\boldsymbol{I} + \boldsymbol{D})^{-1}$ *(where* $D_{ii} = \infty$ *if* $J_{ii} = 0$*).*

Note that, since the non-linearity σ is applied entry-wise, the Jacobian $\boldsymbol{J}$ is a diagonal matrix, and its diagonal entries consist of the vectorization of the Jacobian $\frac{\partial \sigma(\boldsymbol{W}\boldsymbol{Z}\boldsymbol{G}^\top)}{\partial \boldsymbol{Z}}|_{\boldsymbol{Z}^*}$. Therefore, the Jacobian $\boldsymbol{J}$ and hence $\boldsymbol{D}$ can be efficiently computed. We provide the pseudo-codes of FB and PR splitting schemes for the backward propagation described in the above proposition as Algorithm 3 and Algorithm 4 respectively and their Anderson-accelerated version can be found in Algorithm 7 and Algorithm 8.

Remark F.2. It is worth noting that the non-linearity function σ commonly used in practical applications may not be differentiable. However, versions of the generalized implicit function theorem [47; 13] exist, allowing us to apply the implicit differentiation theorem in our setting. Please refer to Section 3 of [13] for further details.

FB backward propagation We now present the pseudo-code of FB splitting method (Algorithm 3) for the backward propagation with the procedure described in Proposition F.1.

Algorithm 3 FB-backward-MIGNN

```
$\boldsymbol{u} = \mathrm{vec}(\boldsymbol{U}) := 0$;   err := 1;   $\boldsymbol{v} := \frac{\partial \ell}{\partial \mathrm{vec}(\boldsymbol{Z}^*)}$
while err $> \epsilon$ do
    $\boldsymbol{u}^{(+)} := (1-\alpha)\boldsymbol{u} + \alpha\, \mathrm{vec}(\boldsymbol{W}^\top \boldsymbol{U} \boldsymbol{G}^\top)$
    $u_i^{(+)} := \begin{cases} \frac{u_i^{(+)} + \alpha v_i}{1+\alpha(1+D_{ii})} & \text{if } D_{ii} < \infty \\ 0 & \text{if } D_{ii} = \infty \end{cases}$
    err := $\frac{\|\boldsymbol{u}^{(+)} - \boldsymbol{u}\|}{\|\boldsymbol{u}^{(+)}\|}$
    $\boldsymbol{u} := \boldsymbol{u}^{(+)}$
end while
Set $\boldsymbol{U} := \mathrm{vec}^{-1}(\boldsymbol{u})$
return $\boldsymbol{v} + \mathrm{vec}(\boldsymbol{W}^\top \boldsymbol{U} \boldsymbol{G}^\top)$
```

Let $\boldsymbol{u}^{(k)}$ be the intermediate variable, the procedure of applying FB splitting on monotone splitting problem Equation (18) can be summarized as finding the fixed-point $\boldsymbol{u}^*$ of the following iteration function

$$\boldsymbol{u}^{(k+1)} := B_\alpha^{\mathrm{FB}}(\boldsymbol{u}^{(k)}) = (\boldsymbol{I} + \alpha \boldsymbol{D})^{-1}((1-\alpha)\boldsymbol{u}^{(k)} + \alpha \boldsymbol{W}^\top \boldsymbol{v}). \tag{19}$$

PR backward propagation We now present the pseudo-code of PR splitting method (Algorithm 4) for the backward propagation with the procedure described in Proposition F.1. Let $\boldsymbol{y}^{(k)}$ be the intermediate variable, the procedure of applying PR splitting on Equation (18) can be summarized as first finding the fixed-point $\boldsymbol{y}^*$ of the following iteration function

Algorithm 4 PR-backward-MIGNN

```
$\boldsymbol{y} := 0; \boldsymbol{u} = \mathrm{vec}(\boldsymbol{U}) := 0$;   err := 1;   $\boldsymbol{v} := \frac{\partial \ell}{\partial \mathrm{vec}(\boldsymbol{Z}^*)}$;   $\boldsymbol{V} := (\boldsymbol{I} + \alpha(\boldsymbol{I} - \boldsymbol{G}^\top \otimes \boldsymbol{W}))^{-1}$
while err $> \epsilon$ do
    $u_i^{(1/2)} := \begin{cases} \frac{y_i + \alpha v_i}{1+\alpha(1+D_{ii})} & \text{if } D_{ii} < \infty \\ 0 & \text{if } D_{ii} = \infty \end{cases}$
    $\boldsymbol{y}^{(1/2)} := 2\boldsymbol{u}^{(1/2)} - \boldsymbol{y}$
    $\boldsymbol{u}^{(+)} := \boldsymbol{V}^\top \boldsymbol{y}^{(1/2)}$
    $\boldsymbol{y}^{(+)} := 2\boldsymbol{u}^{(+)} - \boldsymbol{y}^{(1/2)}$
    err := $\frac{\|\boldsymbol{y}^{(+)} - \boldsymbol{y}\|}{\|\boldsymbol{y}^{(+)}\|}$
    $\boldsymbol{y}, \boldsymbol{u} := \boldsymbol{y}^{(+)}, \boldsymbol{u}^{(+)}$
end while
Compute $\boldsymbol{u}$ where $u_i := \begin{cases} \frac{y_i + \alpha v_i}{1+\alpha(1+D_{ii})} & \text{if } D_{ii} < \infty \\ 0 & \text{if } D_{ii} := \infty \end{cases}$
Set $\boldsymbol{U} := \mathrm{vec}^{-1}(\boldsymbol{u})$
return $\boldsymbol{v} + \mathrm{vec}(\boldsymbol{W}^\top \boldsymbol{U} \boldsymbol{G}^\top)$
```

$$\boldsymbol{y}^{(k+1)} := B_\alpha^{\mathrm{PR}}(\boldsymbol{y}^{(k)}) = 2\boldsymbol{V}^\top \left(2(\boldsymbol{I} + \alpha\boldsymbol{D})^{-1}(\boldsymbol{y}^{(k)} + \alpha \boldsymbol{v}) - \boldsymbol{y}^{(k)}\right) - 2(\boldsymbol{I} + \alpha \boldsymbol{D})^{-1}(\boldsymbol{y}^{(k)} + \alpha \boldsymbol{v}) + \boldsymbol{y}^{(k)} \tag{20}$$

and then the final solution of the operator splitting problem is $\tilde{\boldsymbol{u}} = (\boldsymbol{I} + \alpha \boldsymbol{D})^{-1}(\boldsymbol{y}^* + \alpha \boldsymbol{v})$.

F.3. Anderson acceleration

We first introduce the general Anderson acceleration scheme. Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a function s.t. the Lipschitz constant $L(f) < 1$. Therefore, the function f admits a unique fixed point and can be obtained through Picard iteration. Let $h(\boldsymbol{x}) = f(\boldsymbol{x}) - \boldsymbol{x}$ be the residual function. Let $\boldsymbol{x}^{(0)}$ be the initial guess, $\beta \in (0, 1)$ be a relaxation parameter, and $m > 1$ be an integer parameter. Then the Anderson acceleration update $\boldsymbol{x}^{(k)}$ as

$$\boldsymbol{x}^{(k+1)} = (1 - \beta) \sum_{i=0}^{m} \gamma_i^{(k)} \boldsymbol{x}^{(k-m+i)} + \beta \sum_{i=0}^{m} \gamma_i^{(k)} h\left(\boldsymbol{x}^{(k-m+i)}\right) \tag{21}$$

where the coefficients $\boldsymbol{\gamma}^{(k)} = \left(\gamma_0^{(k)}, \ldots, \gamma_m^{(k)}\right)^\top$ are determined by a least-square problem as the following:

$$\min_{\boldsymbol{\gamma}=(\gamma_0,\ldots,\gamma_m)^\top} \left\| \sum_{i}^{m} h(\boldsymbol{x}^{(k-m+i)})\gamma_i \right\| \text{ s.t.} \sum_{i=0}^{m} \gamma_i = 1.$$

Note that, when $\beta = 1$, the trivial weight $\boldsymbol{\gamma}^{(k)} = (0, \ldots, 0, 1)^\top$ recovers Picard iteration. Therefore, when the Picard iteration converges, the Anderson acceleration also converges and typically faster.

In Algorithm 5, we present the FB MIGNN forward propagation with Anderson acceleration on the FB iteration function F_α^{FB} which is introduced in Section 4 and recalled here:

$$\boldsymbol{Z}^{(k+1)} := F_\alpha^{\mathrm{FB}}(\boldsymbol{Z}^{(k)}) := \operatorname{prox}_f^\alpha \left(\boldsymbol{Z}^{(k)} - \alpha \cdot \left(\boldsymbol{Z}^{(k)} - \boldsymbol{W}\boldsymbol{Z}^{(k)}\boldsymbol{G} - g_{\boldsymbol{B}}(\boldsymbol{X})\right)\right).$$

Algorithm 5 MIGNN-FB-Forward: FB MIGNN forward propagation

```
Input: initial point $\boldsymbol{Z}^{(0)} := \mathbf{0}$, FB damping parameter $\alpha$, AA relaxation parameter $\beta$, max storage size $m \geq 1$.
Compute $\boldsymbol{F}^{(0)} = F_\alpha^{\mathrm{PB}}(\boldsymbol{Z}^{(0)}), \boldsymbol{H}^{(0)} = \boldsymbol{F}^{(0)} - \boldsymbol{Z}^{(0)}$.
for $k = 1, \ldots, K$ do
    Set $m_k = \min(m, k)$
    Compute $\boldsymbol{F}^{(k)} = F_\alpha^{\mathrm{PB}}\left(\boldsymbol{Z}^{(k)}\right)$, $\boldsymbol{H}^{(k)} = \boldsymbol{F}^{(k)} - \boldsymbol{Z}^{(k)}$
    Update $\boldsymbol{H} := (\boldsymbol{H}^{(k-m_k)}, \ldots, \boldsymbol{H}^{(k)})$
    Determine $\boldsymbol{\gamma}^{(k)} = \left(\gamma_0^{(k)}, \ldots, \gamma_{m_k}^{(k)}\right)^\top$ that solves
        $$\min_{\boldsymbol{\gamma}=\left(\gamma_0,\ldots,\gamma_{m_k}\right)^\top} \|\boldsymbol{H}\boldsymbol{\gamma}\| \text{ s.t.} \sum_{i=0}^{m_k} \gamma_i = 1.$$
    Set
        $$\boldsymbol{Z}^{(k+1)} := \beta \sum_{i=0}^{m_k} \gamma_i^{(k)} F_\alpha^{\mathrm{PB}}(\boldsymbol{Z}^{((k-m_k)+i)}) + (1-\beta) \sum_{i=0}^{m_k} \gamma_i^{(k)} \boldsymbol{Z}^{((k-m_k)+i)}.$$
end for
return $\boldsymbol{Z}^{(k+1)}$
```

In Algorithm 6, we present the PR MIGNN forward propagation with Anderson acceleration on the PR iteration function F_α^{PR} which is introduced in Section 4 and recalled here:

$$\boldsymbol{u}^{(k+1)} := F_\alpha^{\mathrm{PR}}(\boldsymbol{u}^{(k)}) = 2\boldsymbol{V}\left(2\operatorname{prox}_f^\alpha(\boldsymbol{u}^{(k)}) - \boldsymbol{u}^{(k)} + \alpha \operatorname{vec}(g_{\boldsymbol{B}}(\boldsymbol{X}))\right) - 2\operatorname{prox}_f^\alpha(\boldsymbol{u}^{(k)}) + \boldsymbol{u}^{(k)}, \tag{22}$$

Algorithm 6 MIGNN-PR-forward: PR MIGNN forward propagation

```
Input: initial point $\boldsymbol{u}^{(0)} = \mathrm{vec}(\boldsymbol{U}^{(0)}) := \mathbf{0}$, PR damping parameter $\alpha$, AA relaxation parameter $\beta$, max storage size $m \geq 1$.
Compute $\boldsymbol{f}^{(0)} := F_\alpha^{\mathrm{PR}}(\boldsymbol{u}^{(0)}), \boldsymbol{h}^{(0)} := \boldsymbol{f}^{(0)} - \boldsymbol{u}^{(0)}$.
for $k = 1, \ldots, K$ do
    Set $m_k := \min(m, k)$
    Compute $\boldsymbol{f}^{(k)} := F_\alpha^{\mathrm{PR}}\left(\boldsymbol{u}^{(k)}\right), \boldsymbol{h}^{(k)} := \boldsymbol{f}^{(k)} - \boldsymbol{u}^{(k)}$
    Update $\boldsymbol{H} := (\boldsymbol{h}^{(k-m_k)}, \ldots, \boldsymbol{h}^{(k)})$
    Determine $\boldsymbol{\gamma}^{(k)} = \left(\gamma_0^{(k)}, \ldots, \gamma_{m_k}^{(k)}\right)^\top$ that solves
        $$\min_{\boldsymbol{\gamma}=\left(\gamma_0,\ldots,\gamma_{m_k}\right)^\top} \|\boldsymbol{H\gamma}\| \text{ s.t.} \sum_{i=0}^{m_k} \gamma_i = 1.$$
    Set
        $$\boldsymbol{u}^{(k+1)} := \beta \sum_{i=0}^{m_k} \gamma_i^{(k)} F_\alpha^{\mathrm{PR}}(\boldsymbol{u}^{((k-m_k)+i)}) + (1-\beta) \sum_{i=0}^{m_k} \gamma_i^{(k)} \boldsymbol{u}^{((k-m_k)+i)}.$$
end for
Set $\boldsymbol{U}^{(k+1)} := \mathrm{vec}^{-1}(\boldsymbol{u}^{(k+1)})$
return $\mathrm{prox}_f^\alpha(\boldsymbol{U}^{(k+1)})$
```

The FB iteration function for the backpropagation B_α^{FB} is introduced in Appendix F.2 and recalled here:

$$\boldsymbol{u}^{(k+1)} := B_\alpha^{\mathrm{FB}}(\boldsymbol{u}^{(k)}) = (\boldsymbol{I} + \alpha \boldsymbol{D})^{-1}((1-\alpha)\boldsymbol{u}^{(k)} + \alpha \boldsymbol{W}^\top \boldsymbol{v}). \tag{23}$$

We present the Anderson-accelerated FB MIGNN backward propagation as Algorithm 7.

Algorithm 7 MIGNN-FB-Backward: FB MIGNN backward propagation

```
Input: initial point $\boldsymbol{u}^{(0)} := \mathrm{vec}(\boldsymbol{U}) := \mathbf{0}$, $\boldsymbol{v} := \frac{\partial \ell}{\partial \mathrm{vec}(\boldsymbol{Z}^*)}$, PR damping parameter $\alpha$, AA relaxation parameter $\beta$, max storage size $m \geq 1$.
Compute $\boldsymbol{f}^{(0)} := B_\alpha^{\mathrm{FB}}(\boldsymbol{u}^{(0)}), \boldsymbol{h}^{(0)} := \boldsymbol{f}^{(0)} - \boldsymbol{u}^{(0)}$.
for $k = 1, \ldots, K$ do
    Set $m_k := \min(m, k)$
    Compute $\boldsymbol{f}^{(k)} := B_\alpha^{\mathrm{FB}}\left(\boldsymbol{u}^{(k)}\right), \boldsymbol{h}^{(k)} := \boldsymbol{f}^{(k)} - \boldsymbol{u}^{(k)}$
    Update $\boldsymbol{H} := (\boldsymbol{h}^{(k-m_k)}, \ldots, \boldsymbol{h}^{(k)})$
    Determine $\boldsymbol{\gamma}^{(k)} = \left(\gamma_0^{(k)}, \ldots, \gamma_{m_k}^{(k)}\right)^\top$ that solves
        $$\min_{\boldsymbol{\gamma}=\left(\gamma_0,\ldots,\gamma_{m_k}\right)^\top} \|\boldsymbol{H\gamma}\| \text{ s.t.} \sum_{i=0}^{m_k} \gamma_i = 1.$$
    Set
        $$\boldsymbol{u}^{(k+1)} := \beta \sum_{i=0}^{m_k} \gamma_i^{(k)} B_\alpha^{\mathrm{FB}}(\boldsymbol{u}^{((k-m_k)+i)}) + (1-\beta) \sum_{i=0}^{m_k} \gamma_i^{(k)} \boldsymbol{u}^{((k-m_k)+i)}.$$
end for
Set $\boldsymbol{U}^{(k+1)} := \mathrm{vec}^{-1}(\boldsymbol{u}^{(k+1)})$
return $\boldsymbol{v} + \mathrm{vec}(\boldsymbol{W}^\top \boldsymbol{U}^{(k+1)} \boldsymbol{G}^\top)$
```

The PR iteration function for the backpropagation B_α^{PR} is introduced in Appendix F.2 and recalled here: let $\boldsymbol{y}^{(k)}$ be the intermediate variable,

$$\boldsymbol{y}^{(k+1)} := B_\alpha^{\mathrm{PR}}(\boldsymbol{y}^{(k)}) = 2\boldsymbol{V}^\top \left(2(\boldsymbol{I}+\alpha\boldsymbol{D})^{-1}(\boldsymbol{y}^{(k)}+\alpha\boldsymbol{v}) - \boldsymbol{y}^{(k)}\right) - 2(\boldsymbol{I}+\alpha\boldsymbol{D})^{-1}(\boldsymbol{y}^{(k)}+\alpha\boldsymbol{v}) + \boldsymbol{y}^{(k)} \tag{24}$$

and then the final solution of the operator splitting problem is $\tilde{u} = (\boldsymbol{I}+\alpha\boldsymbol{D})^{-1}(\boldsymbol{y}^* + \alpha\boldsymbol{v})$. We now present the Anderson-accelerated PR MIGNN backward propagation as Algorithm 8.

Algorithm 8 MIGNN-PR-Backward: PR MIGNN backward propagation

```
Input: initial point $\boldsymbol{y}^{(0)} := \boldsymbol{0}$, $\boldsymbol{v} := \frac{\partial \ell}{\partial \mathrm{vec}(\boldsymbol{Z}^*)}$, PR damping parameter $\alpha$, AA relaxation parameter $\beta$, max storage size $m \geq 1$.
Compute $\boldsymbol{f}^{(0)} := B_\alpha^{\mathrm{PR}}(\boldsymbol{y}^{(0)}), \boldsymbol{h}^{(0)} := \boldsymbol{f}^{(0)} - \boldsymbol{y}^{(0)}$.
for $k = 1, \ldots, K$ do
    Set $m_k := \min(m, k)$
    Compute $\boldsymbol{f}^{(k)} := B_\alpha^{\mathrm{PR}}\left(\boldsymbol{y}^{(k)}\right), \boldsymbol{h}^{(k)} := \boldsymbol{f}^{(k)} - \boldsymbol{y}^{(k)}$
    Update $\boldsymbol{H} := (\boldsymbol{h}^{(k-m_k)}, \ldots, \boldsymbol{h}^{(k)})$
    Determine $\boldsymbol{\gamma}^{(k)} = \left(\gamma_0^{(k)}, \ldots, \gamma_{m_k}^{(k)}\right)^\top$ that solves
        $$\min_{\boldsymbol{\gamma}=\left(\gamma_0,\ldots,\gamma_{m_k}\right)^\top} \|\boldsymbol{H}\boldsymbol{\gamma}\| \text{ s.t.} \sum_{i=0}^{m_k} \gamma_i = 1.$$
    Set
        $$\boldsymbol{y}^{(k+1)} := \beta \sum_{i=0}^{m_k} \gamma_i^{(k)} B_\alpha^{\mathrm{PR}}(\boldsymbol{y}^{((k-m_k)+i)}) + (1-\beta) \sum_{i=0}^{m_k} \gamma_i^{(k)} \boldsymbol{y}^{((k-m_k)+i)}.$$
end for
Compute $\boldsymbol{u}^{(k+1)}$ where $u_i^{(k+1)} := \begin{cases} \frac{y_i^{(k+1)}+\alpha v_i}{1+\alpha(1+D_{ii})} & \text{if } D_{ii} < \infty \\ 0 & \text{if } D_{ii} = \infty \end{cases}$
Set $\boldsymbol{U}^{(k+1)} := \mathrm{vec}^{-1}(\boldsymbol{u}^{(k+1)})$
return $\boldsymbol{v} + \mathrm{vec}(\boldsymbol{W}^\top \boldsymbol{U}^{(k+1)} \boldsymbol{G}^\top)$
```

G. Effects of the Order of Neumann Series Expansion

In this section, we perform ablation studies on the effects of the order of the Neumann series for approximating matrix $(\boldsymbol{I} + \alpha(\boldsymbol{I} - \boldsymbol{G}^\top \otimes \boldsymbol{W}))^{-1}$ in MIGNN-NK. We study the performance of MIGNN-NK for synthetic directed chain classification, benchmark graph node classification, and graph classification.

G.1. Directed chain classification

Examining the Neumann series expansion for the synthetic chain classification task demonstrates the trade-off between accuracy and time complexity. We train MIGNN-NK for three-class classification, where the order K ranges from 1 to 5 in increments of 1. Fig. 9 plots the resulting test accuracy, number of iterations, and time elapsed for each training epoch.

We make three observations as the order of the Neumann series increases. First the accuracy increases with respect to the order with diminishing returns. Second the number of iterations increases relative to the order up 3. Finally, the time elapsed also increases with respect to the order up to 4 and 5 which are similar. These observations underscore the trade-off between accuracy and time complexity as the order increases.

G.2. Graph classification

In this subsection, we apply MIGNN-NK to classify the MUTAG dataset, where K ranges from 1 to 5 incrementing by 1. Fig. 10 plots the test accuracy, the number of iterations, and the time elapsed for training one fold of the 10-fold cross validation.

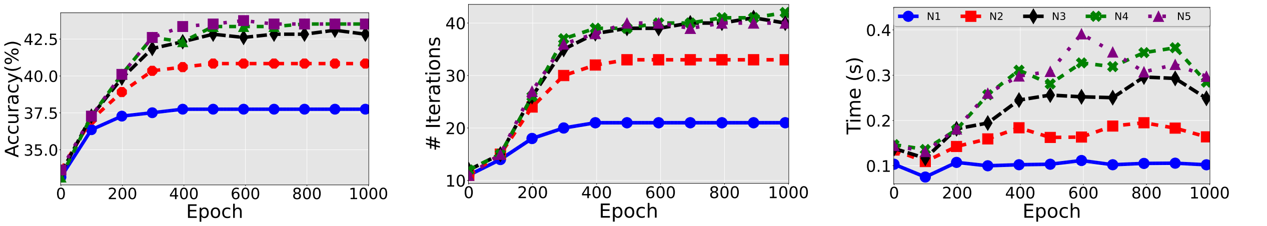


Figure 9. Comparison of Neumann expansion for accuracy, number of iterations, and elapsed time using three-class chain classifications with chain length 140.

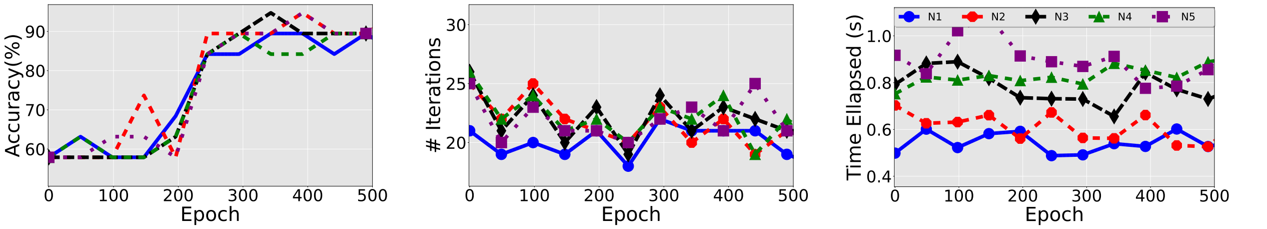


Figure 10. Comparison of Neumann expansion for accuracy, number of iterations and elapsed time using the first fold of the MUTAG graph data set.

Unlike the directed chains and node classification tasks, the graph classification does not show significant improvements from higher-order Neumann expansion on this fold. Although the accuracy and iteration count remain similar among all orders, the time elapsed still scales with the order.

H. More Discussion on When IGNNs Become Expressive for Learning LRD

In this section, we further confirm the interconnection between the accuracy of IGNN for classifying directed chains and the eigenvalues of $|\boldsymbol{W}|$. The accuracy and number of iterations of IGNN and the dynamics of the two leading eigenvalues are plotted in Figs. 11 and 12, respectively, for the binary and three-class cases. These results confirm the phenomena we have discussed in Sec. 1.

I. Details about datasets

Synthetic chains dataset. To evaluate the LRD learning ability of models, we construct synthetic chains dataset as in Gu et al. [24]. Both binary classification and multiclass classification are considered. Let c be the number of classes, that is, there are c types of chains. The label information is only encoded as a one-hot vector in the first c-dimensions of the node feature of the starting nodes of each chain. With c classes, n_c chains for each class, and l nodes in each chain, the chain dataset has $c \times n_c \times l$ nodes in total.

Bioinformatics datasets. MUTAG is a dataset of 188 mutagenic aromatic and heteroaromatic nitro compounds. PTC is a dataset of 344 chemical compounds that report carcinogenicity for male and female rats. COX2 is a dataset of 467 cyclooxygenase-2 (COX-2) inhibitors. PROTEINS is a dataset of 1113 secondary structure elements (SSEs). NCI1 is a public dataset from the National Cancer Institute (NCI) and is a subset of balanced datasets of chemical compounds screened for the ability to suppress or inhibit the growth of a panel of human tumor cell lines.

Amazon product co-purchasing network. This dataset contains 334863 nodes (representing goods), 925872 edges, and 58 label types. An edge is formed between two nodes if the represented goods have been purchased together [36].

Pore networks. The pore network is a simulated dataset that models fluid flow in porous media. Each porous network is randomly generated inside a cubic domain of width 0.1m by Delaunay or Voronoi tessellation. The prediction of equilibrium pressure in a pore network under physical diffusion is introduced as a GNN task in [43]. The GNN model prediction accuracy is compared with the ground truth obtained by solving the diffusion equation directly; see Appendix C in [43] for more details.

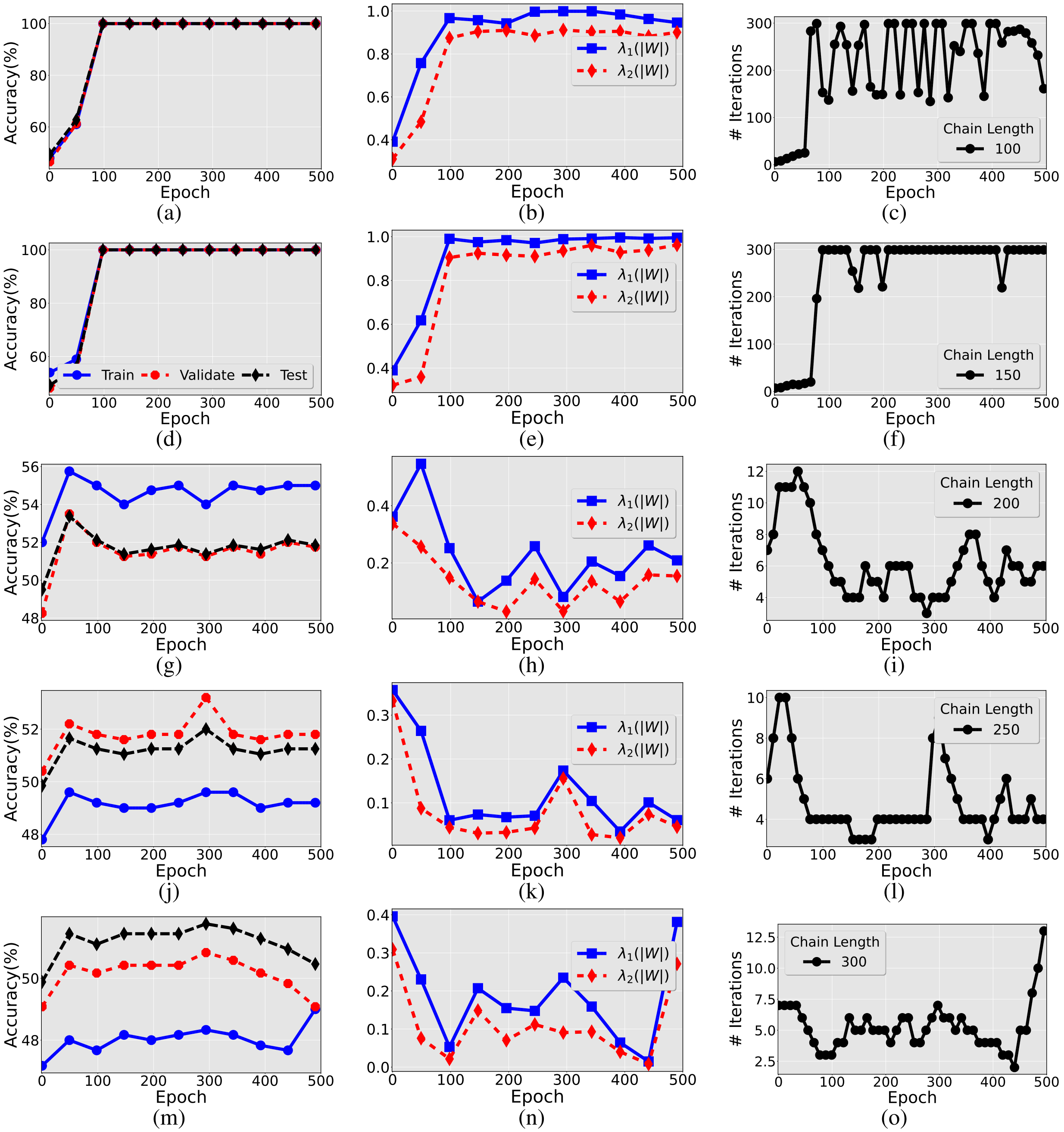
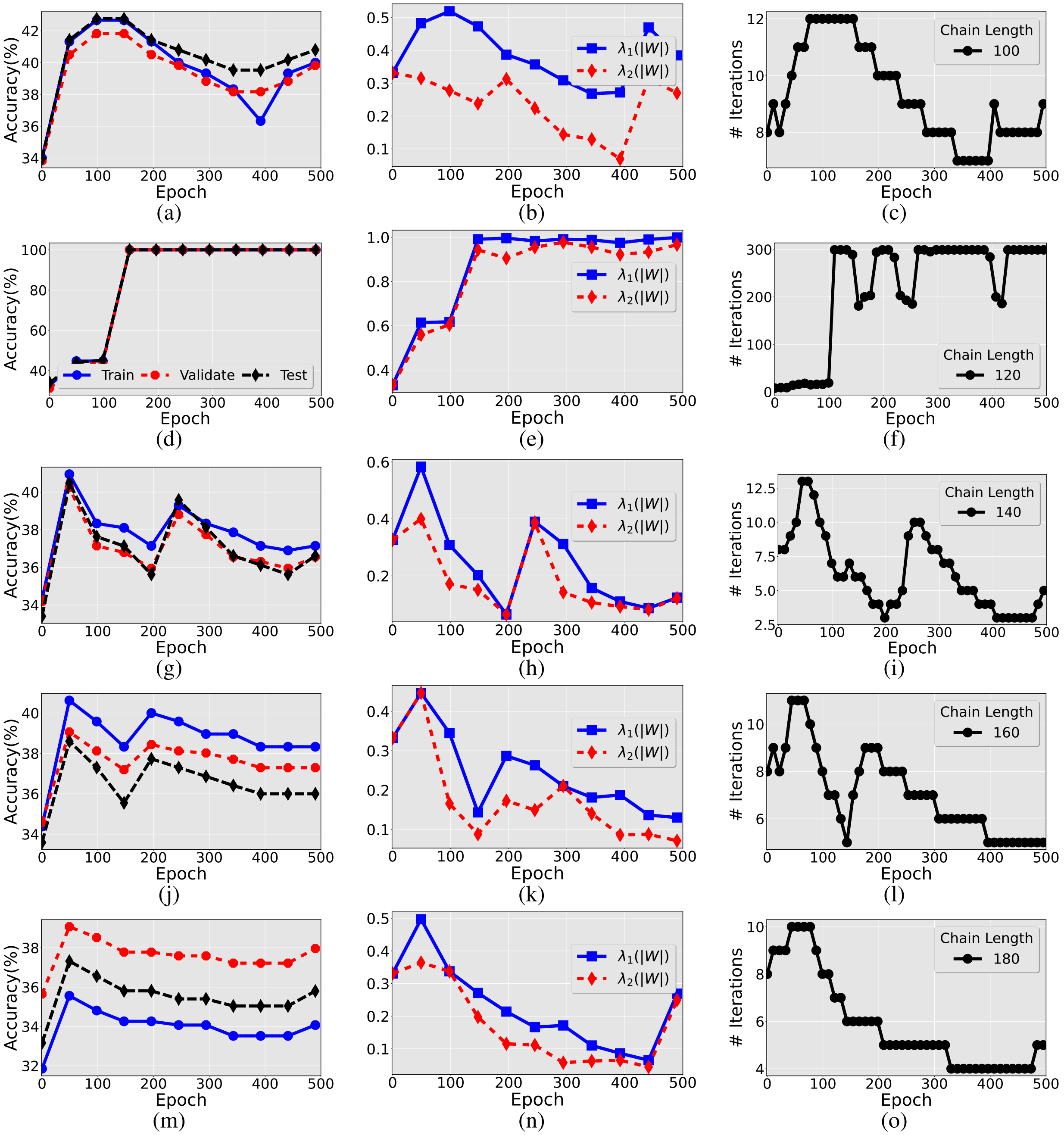


Figure 11. In the first column, the training, test, and validation accuracies of IGNN are depicted for several varying chain lengths. In the second column, the corresponding top two eigenvalues are plotted. The third column depicts the number of Picard iterations for each chain length. When IGNN becomes accurate for chain classification, the corresponding $\lambda_1(|\boldsymbol{W}|)$ becomes close to 1 and requires substantially more iterations for the Picard iteration to converge.

Citation dataset. Cora and Citeseer are large citation datasets that describe the presence of specific words in publications. Pubmed is a large citation dataset that contains information about papers classified for studying one of the three diabetes. The following table adapted from [21] describes the statistics of the three datasets.



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l) (m) (n) (o)

Figure 12. The first column shows the training, test, and validation accuracies of IGNN for several chain lengths of three classes. In the second column, we plot the corresponding top two eigenvalues. In the third column, we plot the number of Picard iterations for each chain length. As the maximum eigenvalue of the system approaches 1, IGNN becomes more accurate for chain classification at the cost of a significantly increased number of training iterations.

Dataset	Type	Classes	Features	Nodes	Edges	Label rate	Avg. SP
Cora	Citation	7	2879	2810	7981	0.047	5.27
Citeseer	Citation	6	3703	2110	3668	0.036	9.31
Pubmed	Citation	3	500	19717	44324	0.003	6.34

Table 4. Dataset statistics. The shortest path length is denoted by Avg. SP.

J. Training procedure details

We provide a short description of the training procedure for each taks and report the tuned hyperparameters for each model. The hyperparameters were selected using the Bayesian search feature of Weights&Bias [11] over a limited range of inputs. The hyperparameters considered are detailed in Table 5.

Hyperparameter	Options
lr	$\{0.001, 0.002, 0.003, 0.004, 0.005, 0.006, 0.007, 0.008, 0.009, 0.01\}$
weight_decay	$\{0, 5e\text{-}5, 3e\text{-}5, 1e\text{-}5, 5e\text{-}4, 3e\text{-}4, 1e\text{-}4\}$
dropout	$\{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$
hidden_features	$\{16, 32, 64, 96, 128, 160, 256\}$
lambda_max	$\{0.9, 0.95, 0.99, 1.0\}$
alpha	$\{0.5, 0.8, 0.9\}$
fp_tol	$\{1e\text{-}6, 3e\text{-}6\}$

Table 5. Hyperparameter tuning range for all tasks.

Synthetic chains dataset. The synthetic chains dataset is used for node classification with randomly generated training (5%), validation (10%) and test (85%) portions. The training procedure uses the Adam optimizer to minimize the negative log likelihood (NLL) loss between the model predictions and ground truth node labels. The model is trained for the full duration of 2000 epochs, at which point the test accuracy is reported.

	lr	weight_decay	dropout	hidden_features	lambda_max	alpha	fp_tol
IGNN	0.01	$5e$-4	0.5	16	0.99	-	$3e$-6
MIGNN-NK	0.01	$5e$-4	0	16	1.0	0.9	$1e$-6

Table 6. Hyperparameter selection for synthetic chains dataset.

Table 6 details the hyper parameters used for the various chain classification tasks. In particular, we use these hyperparameters for Figures 1, 2, 3, 4, 11 and 12.

Citation dataset. The citations dataset is used for node classification with training, validation and test splits comprising $48\%/32\%/20\%$ of the data respectively. We use the ten fixed data splits from Pei et al. [45], and use 10-fold cross validation to evaluate the model performance. Each model is trained using the Adam optimizer to minimize the NLL loss between the model predictions and ground truth node labels. The training will perform early stopping if the validation accuracy does not improve after 100 epochs.

	lr	weight_decay	dropout	hidden_features	lambda_max	alpha	fp_tol	K
				Cora				
MIGNN-Mon	0.005	$3e$-5	0.8	128	1.0	0.5	$3e$-6	—
MIGNN-NK	0.007	$5e$-5	0.8	128	1.0	0.5	$3e$-6	12
				Citeseer				
MIGNN-Mon	0.005	$3e$-5	0.8	128	1.0	0.5	$3e$-6	—
MIGNN-NK	0.005	$3e$-5	0.8	128	1.0	0.5	$3e$-6	12
				PubMed				
MIGNN-Mon	0.005	$3e$-5	0.8	128	1.0	0.5	$3e$-6	—
MIGNN-NK	0.005	$3e$-5	0.8	128	1.0	0.5	$3e$-6	12

Table 7. Hyperparameter selection for Citations datasets.

Bioinformatics datasets. The bioinformatics dataset is used for graph classification with training and test splits comprising 90% and 10% of the data respectively. We use ten randomly generated data folds to perform 10 fold cross validation. The training procedure uses the Adam optimizer to minimize the NLL loss between the model predictions and the ground truth graph labels. On each fold, the model is trained for the full duration of 200 epochs.

	lr	weight_decay	dropout	hidden_features	lambda_max	alpha	fp_tol
MUTAG							
MIGNN-Mon	0.001	0	0.1	128	0.99	0.5	$3e$-6
MIGNN-N1	0.01	0	0.1	128	1.0	0.5	$3e$-6
MIGNN-N3	0.01	0	0.1	128	1.0	0.5	$3e$-6
PTC							
MIGNN-Mon	0.001	0	0.4	128	0.99	0.5	$3e$-6
MIGNN-N1	0.001	0	0.4	128	1.0	0.5	$3e$-6
MIGNN-N3	0.001	0	0.4	128	1.0	0.5	$3e$-6
COX2							
MIGNN-Mon	0.001	0	0.1	128	0.99	0.5	$3e$-6
MIGNN-N1	0.01	0	0.1	128	1.0	0.5	$3e$-6
MIGNN-N3	0.01	0	0.1	128	1.0	0.5	$3e$-6
Proteins							
MIGNN-Mon	0.001	0	0.4	128	0.99	0.5	$3e$-6
MIGNN-N1	0.002	0	0.4	128	1.0	0.5	$3e$-6
MIGNN-N3	0.002	0	0.4	128	1.0	0.5	$3e$-6
NCI1							
MIGNN-Mon	0.001	0	0	128	0.99	0.5	$3e$-6
MIGNN-N1	0.001	0	0	128	1.0	0.5	$3e$-6
MIGNN-N3	0.001	0	0	128	1.0	0.5	$3e$-6

Table 8. Hyperparameter selection for Bioinformatics datasets.

Amazon product co-purchasing network. The Amazon product co-purchasing dataset is used for node classification with fixed training, validation and test splits provided by Gu et al. [24]. Notably the portions of the training data vary as described in Section 5.2. The training procedure uses the Adam optimizer to minimize the BCEwithLogitLoss provided by the Pytorch library. The model is trained for the full duration of 5000 epochs, at which point the F1-micro/F1-macro scores on the test data are reported.

	lr	weight_decay	dropout	hidden_features	lambda_max	alpha	fp_tol
MIGNN-N1	0.01	0	0	256	1.0	0.9	$1e$-6

Table 9. Hyperparameter selection for Amazon product co-purchasing dataset.

Pore networks. The pore networks are used for node classification where the data is generated on the fly following [43]. In particular, 32 training graphs are used for training in every forward epoch with a new training graph sampled every 32nd epoch. The training procedure uses the Adam optimizer to minimize the mean squared error (MSE) between the model predictions and the ground truth node pressures. The model is trained for the full duration of 1000 epochs, at which point the MSE for each test graph is reported.

	lr	weight_decay	dropout	hidden_features	lambda_max	alpha	fp_tol
MIGNN-Mon	0.001	0	0	16	1.0	0.9	$1e$-6
MIGNN-N1	0.001	0	0	16	1.0	0.9	$1e$-6
MIGNN-N3	0.001	0	0	16	1.0	0.9	$1e$-6

Table 10. Hyperparameter selection for pore network dataset.