Monotone Operator Theory-Inspired Message Passing for Learning Long-Range Interaction on Graphs

Justin Baker[1,*] **Qingsong Wang**[1,*] **Martin Berzins**[1] **Thomas Strohmer**[2] **Bao Wang**[1]
1. University of Utah 2. UC Davis

Abstract

Learning long-range interactions (LRI) between distant nodes is crucial for many graph learning tasks. Predominant graph neural networks (GNNs) rely on local message passing and struggle to learn LRI. In this paper, we propose DRGNN to learn LRI leveraging monotone operator theory. DRGNN contains two key components: (1) we use a full node similarity matrix beyond adjacency matrix – drawing inspiration from the personalized PageRank matrix – as the aggregation matrix for message passing, and (2) we implement message-passing on graphs using Douglas-Rachford splitting to circumvent prohibitive matrix inversion. We demonstrate that DRGNN surpasses various advanced GNNs, including Transformer-based models, on several benchmark LRI learning tasks arising from different application domains, highlighting its efficacy in learning LRI. Code is available at `https://github.com/Utah-Math-Data-Science/PR-inspired-aggregation`.

1 INTRODUCTION

Graph that models interacting entities is ubiquitous in real-world applications, e.g., social networks (Perozzi et al., 2014), citation networks (Hamilton et al., 2017), and molecular modeling (Gilmer et al., 2017). The predominant message-passing graph neural networks (MP-GNNs) – based on local message-passing mechanisms – are effective in leveraging inductive bias of graph structures, excelling in graph deep learning (Duvenaud et al., 2015; Kipf and Welling, 2017; Gilmer et al., 2017; Corso et al., 2020; Satorras et al., 2021; Thorpe et al., 2022).

Proceedings of the 27[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238. Correspond to: wangbaonj@gmail.com.

MP-GNNs, e.g., graph convolutional network (GCN) (Kipf and Welling, 2017), usually update the features of each node by aggregating the features of its immediate (1-hop) neighbors. MP-GNNs encode the graph structure into their message-passing mechanism. Nevertheless, the locality of the message-passing in MP-GNNs can overlook the **long-range interactions** (LRI between distant nodes (Ying et al., 2021). The failure in learning LRI is problematic for many graph learning tasks. Taking the molecular property prediction as an example, while the molecular graph is typically represented as a covalent bond network with atoms directly linked through these bonds, the 3D folding of the molecule is also governed by long-range non-covalent interactions between distant atoms (Gilmer et al., 2017). This underlying principle of LRI's importance extends beyond molecules, finding relevance in areas like semantic segmentation label prediction in computer vision (Dwivedi et al., 2020) or robotics (Kurin et al., 2020).

Increasing the depth of MP-GNNs appears to be an obvious and plausible solution for learning LRI, as it allows the message to iteratively propagate to distant nodes. However, various studies have shown that deep MP-GNNs perform much worse than shallow models (Li et al., 2018; Oono and Suzuki, 2020). Furthermore, deep MP-GNNs are expensive in computational and memory costs, especially in the backward pass, which linearly scales with the number of layers. In the context of LRI learning, MP-GNNs are shown to have unsatisfactory performance (Ying et al., 2021). Therefore, improving message-passing mechanisms for MP-GNNs that can learn LRI effectively is desirable.

1.1 Our Contribution

We present DRGNN – a new MP-GNN model for learning LRI effectively using a **full (node) similarity matrix** as the aggregation matrix for message passing rather than sparse adjacency matrices used in existing MP-GNNs. The full similarity matrix emerges from the Personalized PageRank matrix (Page et al., 1998) augmented by a rank-one matrix; see Equation 2

[*] JB and QW are co-first authors of this paper.

in Section 3. However, directly using the full similarity matrix for message passing requires the prohibitive matrix inversion.

We address the computational challenges in matrix inversion by recognizing the monotonicity of the full similarity matrix from a monotone operator viewpoint; see Proposition 1 in Section 4. In particular, we reformulate the node representation – obtained from repeated message passing with the full similarity matrix – as the fixed point of an iterative equation. This idea aligns with deep equilibrium models (DEQs) (Bai et al., 2019, 2022; Pokle et al., 2022; Winston and Kolter, 2020; Gu et al., 2020; Baker et al., 2023), where the model's depth is task dependent. We notice that with a tailored design of the equilibrium equation (Equation 3 in Section 4) and leveraging the **Douglas-Rachford (DR) splitting** (Douglas and Rachford, 1956), we can guarantee the fixed point exists and is unique and the computation of the fixed point can be computed efficiently without matrix inversion (See Subection 4.2.1). As a result, our model's computational complexity aligns with that of GCN. Moreover, DRGNN enjoys a constant memory footprint for backpropagation using implicit differentiation (Robinson, 1991).

1.2 Notations

We adopt the following notations throughout this paper: We represent vectors, matrices, and operators by lowercase boldface, uppercase boldface, and calligraphic letters, respectively. We use $\boldsymbol{I}_n$ and $\mathbf{1}_n$ to denote the identity matrix and all-one vectors of size n, respectively. We denote the adjacency matrix of graph G as $\boldsymbol{A}$, and we use $\boldsymbol{D}$ to denote the diagonal matrix whose i-th diagonal element is the sum of the i-th column of $\boldsymbol{A}$. For the augmented adjacency matrix, we introduce $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_n$. For the normalized adjacency matrix, we use $\bar{\boldsymbol{A}} = \tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1}$ to denote the random walk matrix and $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}$ to denote the symmetrically normalized adjacency matrix. We use the notation $\mathrm{diag}(\boldsymbol{B})$ to represent the vector of diagonal elements of $\boldsymbol{B}$.

1.3 Organization

We organize this paper as follows: In Section 2, we discuss the related work. In Sections 3 and 4, we present the two key components of the proposed model. In Section 5, we present the experimental results on various benchmark graph learning tasks. Missing details and technical proofs are deferred to the appendix.

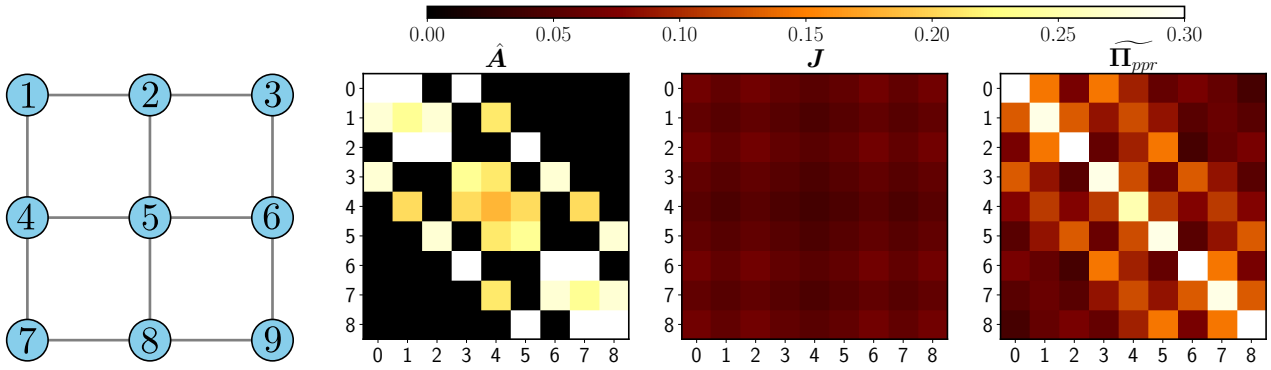
2 RELATED WORK

In this section, we review some related work. We first review some GNN models that expand the receptive field of message-passing beyond the immediate neighbors. We then discuss the fixed-point neural networks.

Multi-hop message passing: A particular idea to increase the receptive field of each node in MP-GNN is to employ higher orders of the adjacency matrices as the aggregation matrix (Wu et al., 2019; Gasteiger et al., 2019a,b; Zhu and Koniusz, 2021; Chien et al., 2021). A remarkable model is PPNP (Gasteiger et al., 2019a), which uses the personalized PageRank matrix $(1-\alpha)(\boldsymbol{I}_n - \alpha\hat{\boldsymbol{A}})^{-1}$ with $\alpha \in (0,1)$ as its aggregation matrix. Using the Neumann expansion of the matrix inverse, represented as $(1-\alpha)(\boldsymbol{I}_n - \alpha\hat{\boldsymbol{A}})^{-1} = (1-\alpha)\sum_{k=0}^{\infty}\alpha^k\hat{\boldsymbol{A}}^k$, we can deduce that PPNP's aggregation matrix attends to every node in the graph based on graph informed node similarities. However, the PPNP model is computationally expensive due to using the dense Personalized PageRank matrix. This challenge is further intensified when one seeks to stack multiple such layers to boost the expressivity of GNNs. To improve the efficiency of PPNP, Gasteiger et al. (2019a) further propose APPNP using a truncated Neumann approximation of the Personalized PageRank matrix. However, the truncated Neumann approximation inherently limits the receptive field of each node. This becomes particularly problematic when the depth required by the task is unknown. APPNP has been generalized in (Gasteiger et al., 2019b; Zhu and Koniusz, 2021; Chien et al., 2021) to incorporate generalized PageRank filters.

Graph Transformer: Transformer-based GNNs, e.g., (Dwivedi and Bresson, 2020; Ying et al., 2021; Kreuzer et al., 2021), leverage the self-attention mechanism between nodes and graph-related positional embeddings to learn node representations. Transformer-based GNNs enable full node connectivity and show remarkable performance in learning LRI (Ying et al., 2021). However, Transformer-based GNNs possess a weak inductive bias (Dosovitskiy et al., 2020); the positional embeddings may not reflect the graph structure, resulting in less meaningful attention weights. Moreover, the quadratic complexity of the self-attention mechanism makes Transformer-based GNNs computationally expensive even for moderately sized graphs.

Fixed-point neural networks: There has been a line of works that learn representations as the fixed point of some equilibrium equation; see, e.g., (Bai et al., 2019; Gu et al., 2020; Bai et al., 2022). This approach results in expressive models with adaptive depth while maintaining a constant memory footprint using implicit differentiation (Robinson, 1991; Bolte et al., 2021). For the correct implementation and training stability of the fixed-point neural networks, one needs to ensure the well-posedness of the fixed-point iteration – the existence and uniqueness of the fixed point. A general well-posedness result is established by Winston and Kolter (2020) that utilizes the



(a) 3×3 grid graph (b) heatmaps of the three matrices

Figure 1: Contrasting heatmaps (entries) of the three aggregation matrices for message passing on a grid graph. $\hat{\boldsymbol{A}}$ captures immediate neighbors, whereas $\boldsymbol{J}$ and $\widetilde{\boldsymbol{\Pi}}_{ppr}$ cover the full graph, with $\boldsymbol{J}$ influenced by node degrees and $\widetilde{\boldsymbol{\Pi}}_{ppr}$ reflecting fine details of the graph structure.

monotone operator theory and allows us to use operator splitting schemes to find the fixed point. The fixed-point neural networks are versatile and competitive across various tasks including language modeling (Bai et al., 2019), semantic segmentation (Bai et al., 2020), optical flow (Bai et al., 2022), and diffusion models (Pokle et al., 2022).

In the context of graph learning, IGNN (Gu et al., 2020) is a fixed-point GNN where each iteration in the fixed-point solving process is modeled as a message-passing step using the normalized adjacency matrix. Given the fact that the memory cost in backward propagation in each message-passing layer directly scales with the number of edges and is linearly proportional to the number of layers, IGNN's constant memory cost is favorable for graph learning tasks. This is particularly true for graph learning tasks that require LRI as the necessarily increased depth of a typical MP-GNN model will incur a significant memory However, it is pointed out that IGNN's well-posedness constrains the choices of weight matrix and poses instability in learning LRI (Baker et al., 2023).

Spectral GNNs: Spectral GNNs have also been designed to learn LRI; remarkable examples include BernNet He et al. (2021) and ChebyNet Defferrard et al. (2016).

3 FULL SIMILARITY MATRICES

In this section, we discuss the motivation of the new aggregation matrix for message passing in our model. Consider a connected graph $G = (V, E)$ with n nodes [1], one way to establish full node connectivity with the minimal extra computational cost is to utilize rank one matrices, e.g. $\frac{1}{n}\mathbf{1}_n\mathbf{1}_n^\top$, which aggregates the averaged features of all nodes. However, the averaged feature is graph agnostic. Instead, one can turn to use the matrix $\boldsymbol{J} = \frac{1}{\mathrm{tr}(\tilde{\boldsymbol{D}})}\mathrm{diag}(\tilde{\boldsymbol{D}}^{1/2})\mathrm{diag}(\tilde{\boldsymbol{D}}^{1/2})^\top$ to establish full node connectivity. Notice that matrix $\boldsymbol{J}$ accounts for node degrees; see the middle panel in Figure 1(b) for a visualization of using matrix $\boldsymbol{J}$ for a toy graph. We notice that rank-one modeling of the pairwise similarity has been used in efficient Transformers (Katharopoulos et al., 2020; Catania et al., 2023).

To further refine the above approach, we take inspiration from the Personalized PageRank matrix. Instead of directly relying on $\boldsymbol{J}$ for message passing, we integrate it into the Personalized PageRank matrix. This integration results in graph structure-aware full node similarities that are suitable for message passing between arbitrary nodes, as we will see later.

Modified Personalized PageRank matrix: Let $\bar{\boldsymbol{A}} := \tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1}$ be the random walk matrix of the augmented graph G. The Personalized PageRank matrix $\boldsymbol{\Pi}_{ppr}$ is characterized by the following equation:

$$\boldsymbol{\Pi}_{ppr}\boldsymbol{i}_x = \alpha\bar{\boldsymbol{A}}\,\boldsymbol{\Pi}_{ppr}\boldsymbol{i}_x + (1-\alpha)\boldsymbol{i}_x,$$

where $\boldsymbol{i}_x$ is the indicator vector of node x and $\alpha \in (0, 1)$ balances the random walk and restart probabilities. The Personalized PageRank matrix can be solved as $\boldsymbol{\Pi}_{ppr} = (1-\alpha)\left(\boldsymbol{I}_n - \alpha\bar{\boldsymbol{A}}\right)^{-1}$. The entries of $\boldsymbol{\Pi}_{ppr}$ represent nodes similarity characterized by the probability of reaching one node from another in a random walk with restart probability $1-\alpha$ (Fouss et al., 2016).

The Personalized PageRank matrix has been used in MP-GNN models, such as PPNP and APPNP (Gasteiger et al., 2019a). In these models,

[1]The results in this section apply to graphs with multiple connected components by considering the relevant matrix as a block diagonal for each component.

the random walk matrix $\bar{\boldsymbol{A}}$ is replaced by its symmetrically normalized variant $\hat{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}$. In our proposed DRGNN, we modify the Personalized PageRank matrix by additionally incorporating the rank one matrix $\boldsymbol{J}$ to enhance direct communication between distant nodes and to benefit the computational efficiency; see Section 4 for details. The modified Personalized PageRank matrix $\widetilde{\boldsymbol{\Pi}_{ppr}}$ satisfies

$$\widetilde{\boldsymbol{\Pi}_{ppr}}\boldsymbol{i}_x = (\alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}})\widetilde{\boldsymbol{\Pi}_{ppr}}\boldsymbol{i}_x + (1-\alpha-\beta)\boldsymbol{i}_x, \quad (1)$$

which resolves to:

$$\widetilde{\boldsymbol{\Pi}_{ppr}} = (1-\alpha-\beta)(\boldsymbol{I}_n - (\alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}}))^{-1}. \quad (2)$$

The entities of $\widetilde{\boldsymbol{\Pi}_{ppr}}$ characterize node similarities that are influenced by the graph structure with coefficients that balance the random walk, restart, and (degree weighted) global average. The matrix $\widetilde{\boldsymbol{\Pi}_{ppr}}$ enables multi-hop message-passing using a single MP-GNN layer, which follows from Neumann series expansion:

$$\big(\boldsymbol{I}_n - \alpha\boldsymbol{J} - \beta\hat{\boldsymbol{A}}\big)^{-1} = \sum_{k=0}^{\infty}\big(\alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}}\big)^k.$$

We illustrate matrix $\widetilde{\boldsymbol{\Pi}_{ppr}}$ in a simple graph in the right panel of Figure 1(b). However, as encountered in PPNP (Gasteiger et al., 2019a), computing the inverse of $(\boldsymbol{I}_n - \alpha\boldsymbol{J} - \beta\hat{\boldsymbol{A}})$ is prohibitive for large graphs. We address this computational bottleneck using monotone operator theory in the next section.

4 DRGNN

DRGNN learns node features by finding the fixed point $\boldsymbol{H}^*$ that satisfies the following equilibrium equation:

$$\boldsymbol{H} = \sigma\big(2\boldsymbol{H} - \gamma(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}})^{-1}\boldsymbol{H} - f_\Theta(\boldsymbol{X})\big), \quad (3)$$

where $|\beta| < 1$, $\alpha + \beta + 1 > 0$, and $\gamma > 1 + |\alpha| + |\beta|$ are learnable. Here, $\sigma(\cdot)$ is the activation function (e.g., ReLU), and $f_\Theta(\boldsymbol{X})$ is a transformation of the input node features $\boldsymbol{X}$ parameterized by Θ. The rationale of the design in Equation 3 will be discussed in the rest of this section.

Starting from $\boldsymbol{H}^{(0)}$, one approach to find the fixed point of Equation 3 is through the (damped) iteration using the Forward-Backward (FB) splitting (Passty, 1979), with a certain damping value $t > 0$, one iterates:

$$\begin{aligned}\boldsymbol{H}^{(i+1)} = \sigma\Big((1+t)\boldsymbol{H}^{(i)} - t\gamma(\boldsymbol{I}_n + \alpha\boldsymbol{J} \\ + \beta\hat{\boldsymbol{A}})^{-1}\boldsymbol{H}^{(i)} - tf_\Theta(\boldsymbol{X})\Big),\end{aligned} \quad (4)$$

for $i = 0, 1, \ldots, K$. This iteration terminates when the difference between two consecutive iterations is smaller than some threshold ϵ and then $\boldsymbol{H}^{(K+1)}$ is taken as the numerical fixed point $\boldsymbol{H}^*$. Each iteration can be viewed as one message-passing layer, then fixed point $\boldsymbol{H}^*$ is a result of multi-step message passing involving full node similarity matrix $(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}})^{-1}$ with its number of steps (depth) adaptive to the learning task. Therefore, $\boldsymbol{H}^*$ captures LRI between distant nodes based on node similarities and also incorporates the nonlinearities between message-passing steps. We verify the improved LRI learning ability of DRGNN over APPNP in Section 5; see also Figure 2 for a visual comparison of the learned features in APPNP and DRGNN.

However, it remains unclear whether the dampened Picard iteration converges to the fixed point $\boldsymbol{H}^*$ and the computation of the matrix inverse $(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}})^{-1}$ in each iteration is computationally expensive. We will address these two challenges in the next two subsections.

4.1 Well-posedness of DRGNN

We discuss the existence and uniqueness of the fixed point of Equation 3, i.e., the well-posedness of DRGNN. Our solution comes from a monotone operator theory viewpoint; see (Bauschke et al., 2011) or Appendix B for background on monotone operator theory. We will show that the fixed point $\boldsymbol{H}^*$ of Equation 3 exists and is unique, even when directly interpreting Equation 3 as an iterative scheme results in a non-contractive operator.

According to the monotone operator theory, finding the fixed point $\boldsymbol{H}^*$ for Equation 3 is equivalent to solving the following monotone inclusion problem (See Appendix B for details): find $\boldsymbol{0} \in (\mathcal{F} + \mathcal{G})(\boldsymbol{H})$ with $\mathcal{F}$ and $\mathcal{G}$ being two set-valued functions, given as follows

$$\begin{aligned}\mathcal{F}(\boldsymbol{H}) &= \Big(\gamma\big(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}}\big)^{-1} - \boldsymbol{I}_n\Big)\boldsymbol{H} \\ &\quad + f_\Theta(\boldsymbol{X}) \\ \text{and} \quad \mathcal{G} &= \partial g,\end{aligned} \quad (5)$$

where ∂g denotes the subgradient of a convex closed proper function g that satisfies $\sigma = \text{prox}_g^t$ for some $t > 0$. Here, the proximal operator is defined as $\text{prox}_g^t(x) := \arg\min_z \big\{\frac{1}{2}\|x - z\|^2 + t\,g(z)\big\}$ for any $t > 0$. Most commonly used activation functions, such as ReLU, satisfy this condition. Specifically, ReLU $= \text{prox}_g^t$ for $\forall t > 0$ with g being the indicator of the positive octant, i.e. $g(x) = I\{x \geq 0\}$.

It's worth noting that the inclusion of $2\boldsymbol{H}$ in equation 3 results in $-\boldsymbol{I}_n$ in $\mathcal{F}$ and it marks a significant difference from what is used in the previous work that connects monotone inclusion problem with fixed-point neural networks (Winston and Kolter, 2020; Baker

et al., 2023). This specific design will be a key factor in achieving computational efficiency; see Equation 8 in the subsequent section.

According to the monotone operator theory, the monotone inclusion problem Equation 5 admits a unique solution if the linear operator $\mathcal{F}$ is strongly monotone. In particular, we have the following result whose proof can be found in Appendix A.

Proposition 1. *The operator $\mathcal{F}$ is strongly monotone if $|\beta| < 1$, $\alpha+\beta+1 > 0$, and $\gamma > 1+|\alpha|+|\beta|$. Moreover, the monotone inclusion problem* find $\mathbf{0} \in (\mathcal{F}+\mathcal{G})(\mathbf{H})$ *with $\mathcal{F}$ and $\mathcal{G}$ defined in Equation 5 admits a unique solution, and Equation 3 admits a unique fixed point.*

4.2 Efficient Implementation of DRGNN

In this subsection, we present an efficient implementation of DRGNN. Specifically, in the forward pass, the fixed-point nature of the hidden encoding allows us to use any root finding solver for $\boldsymbol{H}^*$ not necessarily the Picard iteration (Equation 4) derived from FB splitting. Peaceman-Rachford (PR) splitting (Peaceman and Rachford, 1955) has also been studied in the context of fixed-point neural networks (Winston and Kolter, 2020; Baker et al., 2023). PR splitting in general converges faster than FB splitting, but it requires the computation of the matrix inverse that originates from finding the resolvent for the operator $\mathcal{F}$. As we will see in this subsection, our tailored design of $\mathcal{F}$ allows us to compute the resolvent efficiently by circumventing the computation of the matrix inverse, which is a sharp contrast to the previous work (Winston and Kolter, 2020; Baker et al., 2023). We additionally opt for DR splitting (Douglas and Rachford, 1956), which is an averaged version of PR splitting that has a better convergence guarantee in general (Ryu and Boyd, 2016), and we observed more robust performance in comparison to PR splitting for our setting. For the backward pass, the implicit differentiation method from (Geng et al., 2021) is employed.

4.2.1 Forward Pass via DR Splitting

We now elaborate on DR splitting for solving the monotone inclusion problem Equation 5 and hence to obtain the fixed point $\boldsymbol{H}^*$ of Equation 3. DR splitting is a powerful tool for solving monotone inclusion problems (Ryu and Boyd, 2016). With operators $\mathcal{F}$ and $\mathcal{G}$ defined in Equation 5, DR splitting solves the monotone inclusion problem "find $\mathbf{0} \in (\mathcal{F}+\mathcal{G})(\boldsymbol{H})$" by starting with an initialization[2] of the intermediate state $\boldsymbol{U}^{(0)}$, and then updates as follows:

$$\boldsymbol{U}^{(i+1)} = \frac{1}{2}\mathcal{C}_{\mathcal{F}}\mathcal{C}_{\mathcal{G}}(\boldsymbol{U}^{(i)}) + \frac{1}{2}\mathcal{I}, \tag{6}$$

where $\mathcal{I}$ denotes the identity operator, and the operators $\mathcal{R}_{\mathcal{G}}$ and $\mathcal{C}_{\mathcal{G}}$ denote the resolvent and Cayley operators of $\mathcal{G}$, respectively. They are defined as

$$\mathcal{R}_{\mathcal{G}} = (\mathcal{I}+t\mathcal{F})^{-1}, \quad \mathcal{C}_{\mathcal{G}} = 2\mathcal{R}_{\mathcal{G}} - \mathcal{I}, \ \forall t > 0.$$

Similarly, $\mathcal{C}_{\mathcal{F}}$ is the Cayley operator of the operator $\mathcal{F}$.

Then the fixed point $\boldsymbol{H}^*$ of Equation 3 is determined by $\boldsymbol{H}^* = \sigma(\boldsymbol{U}^*)$. It is worth noting that both resolvent and Cayley operators are non-expansive for any strongly monotone operator. DR splitting converges linearly to $\boldsymbol{H}^*$ for any choice of $t > 0$ when $\mathcal{F}$ is strongly monotone and Lipschitz; see (Ryu and Boyd, 2016, Section 6) or Appendix B for details.

Plugging the detailed form of $\mathcal{F}$ and $\mathcal{G}$ in Equation 5 into Equation 6, we obtain the following iterative schemes for the intermediate state $\boldsymbol{U}^{(i)}$:

$$\begin{aligned} \boldsymbol{U}^{(i+1/2)} &= 2\sigma(\boldsymbol{U}^{(i)}) - \boldsymbol{U}^{(i)} \\ \boldsymbol{U}^{(i+1)} &= \boldsymbol{U}^{(i)} + \boldsymbol{V}\big(\boldsymbol{U}^{(i+1/2)} - tf_{\Theta}(\boldsymbol{X})\big) - \sigma(\boldsymbol{U}^{(i)}) \end{aligned} \tag{7}$$

where

$$\boldsymbol{V} := \big(\boldsymbol{I}_n + t(\gamma(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}})^{-1} - \boldsymbol{I}_n)\big)^{-1}.$$

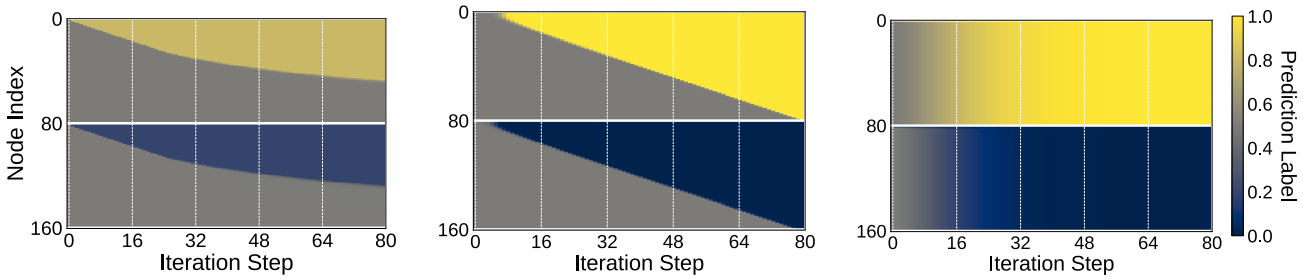
As discussed above, DR splitting converges to $\boldsymbol{H}^*$ linearly for any $t > 0$, and when $t = 1$, we have

$$\begin{aligned} \boldsymbol{V} &= \Big(\boldsymbol{I}_n + \gamma\big(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}}\big)^{-1} - \boldsymbol{I}_n\Big)^{-1} \\ &= \frac{1}{\gamma}(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}}). \end{aligned} \tag{8}$$

Therefore, when using DR splitting in Equation 7 to compute the fixed point of Equation 3, the needs of computing $\big(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}}\big)^{-1}$ and in the linear part $\boldsymbol{V}$ of the resolvent operator $\mathcal{R}_{\mathcal{F}}$ cancel out. This is a particular advantage of having $-\boldsymbol{I}_n$ in the operator $\mathcal{F}$ which comes from our tailored design of Equation 3.

Initializing with the fixed point of the previous forward pass: In training, the learned function $f_{\Theta}(\boldsymbol{X})$ tends to stabilize as the model converges. Consequently, the difference between fixed points in consecutive forward passes become negligible. Therefore, it is natural to leverage the numerical fixed point from the previous forward pass to initialize the iteration of the new forward pass. This strategy has been adopted to accelerate training fixed-point networks (Bai et al., 2022; Pokle et al., 2022).

[2]In practice, we initialize the solver with the fixed point found in the previous forward pass.



(a) IGNN (b) APPNP (c) DRGNN

Figure 2: Node classification results on the binary Synthetic Chain task. The tasks consist of two stacked chains from different classes, each of length 80, indexed from 0-79 and 80-159 respectively. The depicted results show the predicted labels over progressive iteration steps which stems from unrolling in APPNP or fixed point iteration in IGNN and DRGNN. We observe that IGNN fails to classify all nodes correctly and APPNP can correctly classify all nodes after 80 iterations. In contrast, DRGNN is able to classify all nodes correctly after only a few iterations, and the prediction probabilities of the two classes are well separated.

4.2.2 Backward Pass

By utilizing fixed-point encoding for learning node features, the memory consumption for backpropagation becomes independent of the number of iterations (layers) by using implicit differentiation. For simplicity, we denote the DR splitting iteration in Equation 7 as $\boldsymbol{U}^{(i+1)} = F_{DR}(\boldsymbol{U}^{(i)}; f_\Theta(\boldsymbol{X}))$, $\boldsymbol{H}^{(i+1)} = \sigma(\boldsymbol{U}^{i+1})$. Let θ represent a parameter in Θ. For any loss function ℓ, by using the implicit differentiation, we have

$$\frac{\partial \ell}{\partial \theta} = \frac{\partial \ell}{\partial \boldsymbol{H}^*} \frac{\partial \boldsymbol{H}^*}{\partial \boldsymbol{U}^*} \left(\boldsymbol{I}_n - \mathcal{J}_{F_{DR}}(\boldsymbol{U}^*)\right)^{-1} \times \frac{\partial F_{DR}(\boldsymbol{U}^*; f_\Theta(\boldsymbol{X}))}{\partial \theta}, \tag{9}$$

where $\mathcal{J}_{F_{DR}}(\boldsymbol{U}^*)$ is the Jacobian matrix of F_{DR} evaluated at the fixed-point $\boldsymbol{U}^*$. Note that the matrix $\boldsymbol{I}_n - \mathcal{J}_{F_{DR}}(\boldsymbol{U}^*)$ is invertible because the iteration of DR splitting is contractive. According to Equation 9, backpropagation only requires the gradient at the final fixed point. We can disable automatic differentiation during the fixed-point finding process and only run one more iteration with automatic differentiation to use the stored gradients for computing the true gradients through Equation 9. However, the inverse Jacobian term $(\boldsymbol{I}_n - \mathcal{J}_{F_{DR}}(\boldsymbol{U}^*))^{-1}$ can be computationally expensive (Bai et al., 2019; Winston and Kolter, 2020). In our implementation, we employ the inexact gradient method (Geng et al., 2021), which can maintain the descent direction. This method, also adopted in (Pokle et al., 2022; Anil et al., 2022), significantly accelerates backpropagation. This process involves locating the fixed-point $\boldsymbol{H}^*$ with automatic differentiation off, enabling it for a few more iterations, and using the stored gradient to update DRGNN directly.

4.3 Time and Memory Complexities

In this section, we analyze the time and memory complexities of DRGNN. Let n and $|E|$ denote the number of nodes and edges in the graph, respectively. We use d to denote the feature dimension. We assume the graph is sparse, which is common in graph learning. During the forward pass, DRGNN has a time complexity of $O(|E|Md)$, where M represents the maximum number of iterations. Correspondingly, its space complexity is $O(|E| + nd)$. DRGNN leverages the inexact gradients to accelerate the backpropagation. Consequently, the time complexity is $O(|E|Pd)$, where P indicates the number of inexact gradient computations. The space complexity during the backward pass is $O(P|E| + nd)$. Our experiments align with findings in (Pokle et al., 2022; Anil et al., 2022) that small values of P (less than five) maintain high performance.

5 EXPERIMENTAL RESULTS

In this section, we validate the performance of DRGNN on various graph learning tasks, including both synthetic datasets and real-world peptide molecular modeling datasets for LRI learning, as well as graph transductive node classifications. We implement DRGNN using PyG (Fey and Lenssen, 2019) and GraphGym (You et al., 2020) frameworks and conduct all experiments on a single NVIDIA RTX 3090 GPU.

5.1 Learning LRI In Synthetic Datasets

In this subsection, we consider two synthetic datasets that are designed to test the model's ability to learn LRI; the corresponding tasks provide controlled environments where LRI is crucial for the model to achieve

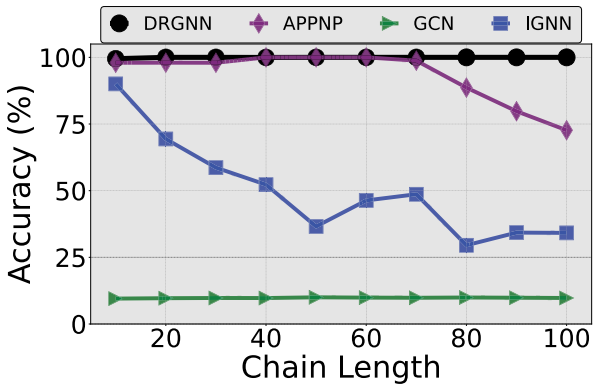


Figure 3: Accuracy for multi-class Synthetic Chains over varying lengths with 10 classes and 20 chains.

high performance. We further provide a visual understanding of the learning process to demonstrate the effectiveness of DRGNN in learning LRI.

5.1.1 Synthetic Chains Dataset

The Synthetic Chains node classification dataset (Gu et al., 2020) consists of a collection of chains where nodes within the same chain share identical labels. However, only the starting node on each chain has a meaningful feature as the one-hot indicator of the class, while all other nodes carry zero-valued features. To accurately classify all nodes on the chains, a model must be able to propagate the class indicator feature from the initial node to the distant nodes in the forward pass without information loss. Consequently, the Synthetic Chains task poses a significant challenge for numerous GNN models, particularly when the depth of GNN is not sufficient or the model is unstable in learning LRI. We consider both a binary Synthetic Chain task consisting of 2 classes of 10 chains with 80 nodes each and a multi-class Synthetic Chain task consisting of 10 classes of 10 chains each. In both tasks, we follow the training procedure in (Gu et al., 2020).

A visualization of the learning process in binary classification: In Figure 2, we provide a visualization of the learning process of APPNP, IGNN, and DRGNN for binary Synthetic Chain classification. The **iteration step** is characterized by the unrolling step in APPNP and the fixed point iteration step in IGNN and DRGNN. The prediction probabilities of the two classes are shown as color intensity normalized to $[0, 1]$. We first train each model for 100 epochs and then use the learned parameters to predict the node labels from the learned features on the test graph at each iteration step. Each model is provided with a total of 80 iteration steps to propagate the node features. We observe that IGNN fails to correctly classify all nodes and the predicted probabilities of the two classes are not well-separated. In contrast, both APPNP and DRGNN are able to correctly classify all nodes, which shows the effectiveness of using the personalized PageRank-style matrix for message passing in learning LRI. Additionally, DRGNN quickly obtains the correct prediction for all nodes while APPNP requires more iterations to achieve the same result. This demonstrates the effectiveness of the learned fixed point in DRGNN.

Multi-class classification: The multi-class classification task poses a more challenging problem for the models to learn LRI. Figure 3 illustrates the mean test accuracy, calculated over 10 random initializations, for baseline models trained on multi-class Synthetic Chains with chain lengths varying from 10 to 100. In this experiment, each architecture is tuned at chain length 100 with 200 iterations. Although each model is capable of fully propagating the node information, only DRGNN consistently classifies chains accurately across all investigated lengths. Additional results for various models can be found in Appendix C.

5.1.2 Color-counting Dataset

The Color-counting dataset (Liu et al., 2022) introduces a node classification task that is more challenging than the Synthetic Chains. This dataset uses the same chain-like structure but rather than determining node labels based on the starting node's color, node labels within each chain are assigned according to the majority node color, encoded as one-hot feature vectors. This setup emphasizes the need for GNNs to acquire multiscale information in LRI learning for optimal performance. In light of this, Liu et al. (2022) presented MGNNI, which surpasses other baseline models on this dataset.

We compare DRGNN with MGNNI as well as APPNP and GCN on this task. Therefore, the necessity of identifying the majority color during message passing raises additional challenges to learn LRI. We report the mean test accuracy over 10 random initializations for Color-counting chains of length 10 to 200 in Figure 4. We observe that DRGNN obtains close to full accuracy across all chain lengths, while all other models fail to learn LRI or identify the majority color. This validates the effectiveness of the DRGNN message-passing scheme in complicated LRI learning tasks.

5.2 Learning LRI In Molecular Modeling

We explore the effectiveness of DRGNN in real-world scenarios by evaluating its performance on the Peptides-Func and Peptides-Struct molecular modeling datasets (Dwivedi et al., 2022). These datasets are designed to test the model's ability to learn LRI and it

Model	Peptides-func		Peptides-struct	
	Train AP ↑	Test AP ↑	Train MAE ↓	Test MAE ↓
GCN*	0.8840 ± 0.0131	0.5930 ± 0.0023	0.2939 ± 0.0055	0.3496 ± 0.0013
GINE*	0.7682 ± 0.0154	0.5498 ± 0.0079	0.3116 ± 0.0047	0.3547 ± 0.0045
GatedGCN*	0.8695 ± 0.0402	0.5864 ± 0.0077	0.2761 ± 0.0032	0.3420 ± 0.0013
GatedGCN+RWSE*	0.9131 ± 0.0321	0.6069 ± 0.0035	0.2578 ± 0.0116	0.3357 ± 0.0006
Transformer+LapPE*	0.8438 ± 0.0263	0.6326 ± 0.0126	0.2403 ± 0.0066	0.2529 ± 0.0016
SAN+LapPE*	0.8217 ± 0.0280	0.6384 ± 0.0121	0.2822 ± 0.0108	0.2683 ± 0.0043
SAN+RWSE*	0.8612 ± 0.0219	0.6439 ± 0.0075	0.2680 ± 0.0038	0.2545 ± 0.0012
GraphGPS	NA	0.6535 ± 0.0041	NA	0.2500 ± 0.0005
DRGNN (ours)	$\mathbf{0.9481 \pm 0.0237}$	$\mathbf{0.6586 \pm 0.0042}$	$\mathbf{0.2193 \pm 0.0151}$	$\mathbf{0.2495 \pm 0.0015}$

Table 1: Performance comparison of different models on LRI learning tasks with the Peptides-Func and Peptides-Struct datasets. The results of the models marked with * are taken from (Dwivedi et al., 2022) and GraphGPS results are taken from (Rampášek et al., 2022) and training performance is not available. GCN, GINE, GatedGCN, and GatedGCN+RWSE are MP-GNN baselines while Transformer+LapPE, SAN+LapPE, SAN+RWSE, and GraphGPS are Transformer-based models. DRGNN outperforms both MP-GNN and Transformer-based models on both datasets in terms of both training and test performance, validating DRGNN's ability to learn LRI.

Model	ogbn-arxiv	Computers	Photo	CS	Physics
GCN*	71.74 ± 0.27	89.65 ± 0.52	92.70 ± 0.20	92.92 ± 0.12	96.18 ± 0.07
APPNP	71.74 ± 0.29	90.18 ± 0.17	94.32 ± 0.14	94.49 ± 0.07	96.54 ± 0.07
Graphormer*	OOM	OOM	OOM	92.74 ± 0.14	OOM
SAN*	OOM	89.83 ± 0.16	94.86 ± 0.10	94.51 ± 0.15	OOM
GraphGPS*	OOM	OOM	95.06 ± 0.13	93.93 ± 0.12	OOM
NAGphormer*	NA	91.22 ± 0.14	95.49 ± 0.11	$\mathbf{95.75 \pm 0.09}$	$\mathbf{97.34 \pm 0.03}$
DRGNN (ours)	$\mathbf{71.90 \pm 0.04}$	$\mathbf{95.15 \pm 0.58}$	$\mathbf{95.56 \pm 0.47}$	94.89 ± 0.34	96.96 ± 0.16

Table 2: Performance comparison of different models on transductive node classification of Citation Networks. Results marked with * are reported in (Chen et al., 2023). Models that run out of memory are reported with OOM and models without existing results with NA. GCN and APPNP are MP-GNN baselines, Graphormer, SAN, and GraphGPS are Transformer-based models, and NAGphormer is a mixture of MP-GNN and Transformer-style models. DRGNN achieves state-of-the-art results on Amazon-Computers and outperforms the baseline MP-GNN models on each task.

is shown that Transformer-based models are more effective than MP-GNNs. Both datasets contain 15,535 graphs with a total of 2.3 million nodes, each representing a peptide with atoms as nodes and covalent bonds as edges; see Figure 5 for an illustration. On average, peptide graphs from these datasets consist of about 150 nodes with a typical node degree nearing 2, making them chain-like structures. Peptide-func is a multi-label graph classification task and the goal is to predict the functional properties of peptides. Peptide-struct is a graph regression task and the goal is to predict the 3D structure of peptides. In both cases, capturing non-covalent LRI becomes vital for a GNN model to excel in this setting.

We compare DRGNN against several strong baseline models, including GCN (Kipf and Welling, 2017), GINE (Xu et al., 2019), GatedGCN (Bresson and Laurent, 2017) and its variant with random walk structural encoding (RWSE) GatedGCN+RWSE, Transformer (Vaswani et al., 2017) with Laplacian position encoding (LapPE), SAN variants (Kreuzer et al., 2021) (SAN+LapPE and SAN+RWSE), and GraphGPS (Rampášek et al., 2022). We summarize the mean training and test performance over 10 random initializations in Table 1, showing that DRGNN outperforms all baseline models on both datasets in terms of both training and test performance.

5.3 Learning Transductive Datasets

We further demonstrate the effectiveness of DRGNN in learning LRI across five transductive node classification tasks, including the citation networks Coauthor-

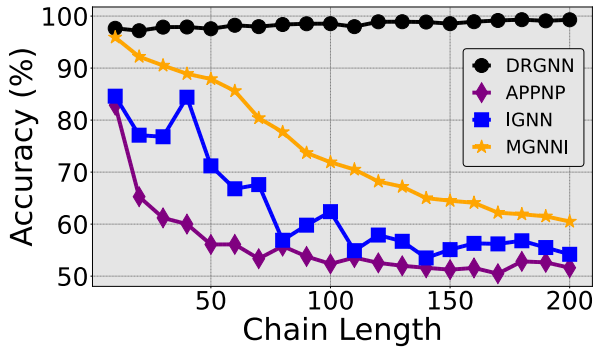


Figure 4: Accuracy of DRGNN vs. a few strong baseline models for the Color-counting dataset over varying lengths. DRGNN achieves close to full accuracy in all experiments, while all other models' performance significantly degrades as the chain length increases.

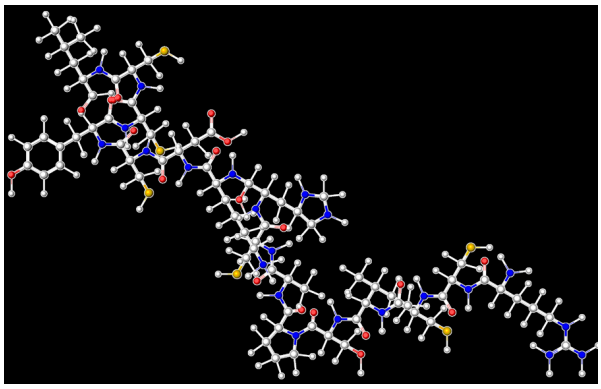


Figure 5: Schematic representation of a peptide graph. Atoms are depicted as nodes connected by covalent bonds (edges). It is worth noting that the graph's 3D structure is further influenced by non-covalent interactions, which are not inherently depicted in its graph representation.

CS/Physics (Shchur et al., 2018), ogbn-arxiv (Hu et al., 2020), and the co-purchasing networks Amazon-Computers/Photo (Shchur et al., 2018). These tasks consist of single large graphs with up to 170K nodes, 1.2M edges, and 23 length diameters, and contrasting the prior mini-batch training on multiple molecular graphs. Comprehensive dataset details can be found in Appendix D.1. Consequently, the transductive tasks bring to light memory challenges for several mainstream Transformer-based architectures and illustrate the memory efficiency of DRGNN. We compare against existing results from (Chen et al., 2023) which serves to illustrate the difference between MP-GNN and Transformer-based models. In particular, the MP-GNN baselines are GCN and APPNP, the Transformer-based architectures include Graphormer, SAN, and GraphGPS, and the NAGphormer (Chen et al., 2023) is a mixture of MP-GNN and Transformer-style models.

Table 2 verifies the memory issues for prominent Transformer-based models and illustrates the strength of DRGNN. In particular, DRGNN achieves the best performance on ogbn-arxiv and Amazon-Computers/Photo and outperforms the baseline MP-GNNs on all five tasks. DRGNN also outperforms many Transformer-based models (only slightly inferior to NAGphormer on Coauthor-CS/Physics), highlighting its superior memory efficiency and effectiveness in learning LRI for inductive tasks.

6 CONCLUSION

The 1-hop message-passing nature of many MP-GNNs limits their ability to learn LRI, which is crucial for many graph learning tasks. To effectively learn LRI, We propose DRGNN leveraging a modified Personalized PageRank matrix – a full node similarity matrix. DRGNN is designed and efficiently implemented through the lens of monotone operator theory, which allows us to circumvent the matrix inversion that comes from the use of a full node similarity matrix. We demonstrate the effectiveness of DRGNN over existing strong baseline models on various graph learning tasks, including synthetic datasets and benchmark real-world datasets.

Acknowledgement

This material is based on research sponsored by NSF grants DMS-2152762, DMS-2208361 and DMS-2219956, and DOE grant DE-SC0023490.

References

Anil, C., Pokle, A., Liang, K., Treutlein, J., Wu, Y., Bai, S., Kolter, J. Z., and Grosse, R. B. (2022). Path independent equilibrium models can better exploit test-time computation. *Advances in Neural Information Processing Systems*, pages 7796–7809.

Bai, S., Geng, Z., Savani, Y., and Kolter, J. Z. (2022). Deep equilibrium optical flow estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 620–630.

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

Bai, S., Koltun, V., and Kolter, J. Z. (2020). Multiscale deep equilibrium models. *Advances in Neural Information Processing Systems*, 33:5238–5250.

Baker, J., Wang, Q., Hauck, C. D., and Wang, B. (2023). Implicit graph neural networks: A monotone operator viewpoint. In *International Conference on Machine Learning*, pages 1521–1548.

Bauschke, H. H., Combettes, P. L., et al. (2011). *Convex analysis and monotone operator theory in Hilbert spaces*, volume 408. Springer.

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

Bresson, X. and Laurent, T. (2017). Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*.

Catania, F., Spitale, M., and Garzotto, F. (2023). Conversational agents in therapeutic interventions for neurodevelopmental disorders: a survey. *ACM Computing Surveys*, 55(10):1–34.

Chen, J., Gao, K., Li, G., and He, K. (2023). NAGphormer: A tokenized graph transformer for node classification in large graphs. In *The Eleventh International Conference on Learning Representations*.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. (2020). Simple and deep graph convolutional networks. In *Proceedings of the 37th International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 1725–1735.

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

Chien, E., Peng, J., Li, P., and Milenkovic, O. (2021). Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*.

Corso, G., Cavalleri, L., Beaini, D., Liò, P., and Veličković, P. (2020). Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Douglas, J. and Rachford, H. H. (1956). On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American mathematical Society*, (2):421–439.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*.

Dwivedi, V. P. and Bresson, X. (2020). A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*.

Dwivedi, V. P., Joshi, C. K., Luu, A. T., Laurent, T., Bengio, Y., and Bresson, X. (2020). Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*.

Dwivedi, V. P., Rampášek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. (2022). Long range graph benchmark. *Advances in Neural Information Processing Systems*, pages 22326–22340.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Fouss, F., Saerens, M., and Shimbo, M. (2016). *Algorithms and models for network data and link analysis*. Cambridge University Press.

Gasteiger, J., Bojchevski, A., and Günnemann, S. (2019a). Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations*.

Gasteiger, J., Weißenberger, S., and Günnemann, S. (2019b). Diffusion improves graph learning. *Advances in neural information processing systems*.

Geng, Z., Zhang, X.-Y., Bai, S., Wang, Y., and Lin, Z. (2021). On training implicit models. *Advances in Neural Information Processing Systems*, pages 24247–24260.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR.

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

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

He, M., Wei, Z., Xu, H., et al. (2021). Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. *Advances in Neural Information Processing Systems*, 34:14239–14251.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. (2020). Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

Kreuzer, D., Beaini, D., Hamilton, W., Létourneau, V., and Tossou, P. (2021). Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, pages 21618–21629.

Kurin, V., Igl, M., Rocktäschel, T., Boehmer, W., and Whiteson, S. (2020). My body is a cage: the role of morphology in graph-based incompatible control. *arXiv preprint arXiv:2010.01856*.

Li, Q., Han, Z., and Wu, X.-M. (2018). Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*.

Liu, J., Hooi, B., Kawaguchi, K., and Xiao, X. (2022). Mgnni: Multiscale graph neural networks with implicit layers. *arXiv preprint arXiv:2210.08353*.

Liu, J., Kawaguchi, K., Hooi, B., Wang, Y., and Xiao, X. (2021). EIGNN: Efficient infinite-depth graph neural networks. In *Advances in Neural Information Processing Systems*.

Loshchilov, I. and Hutter, F. (2017). Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

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

Page, L., Brin, S., Motwani, R., and Winograd, T. (1998). The pagerank citation ranking: Bring order to the web. Technical report, Technical report, stanford University.

Passty, G. B. (1979). Ergodic convergence to a zero of the sum of monotone operators in hilbert space. *Journal of Mathematical Analysis and Applications*, 72(2):383–390.

Peaceman, D. W. and Rachford, Jr, H. H. (1955). The numerical solution of parabolic and elliptic differential equations. *Journal of the Society for industrial and Applied Mathematics*, 3(1):28–41.

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

Pokle, A., Geng, Z., and Kolter, J. Z. (2022). Deep equilibrium approaches to diffusion models. *Advances in Neural Information Processing Systems*, pages 37975–37990.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. (2022). Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515.

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

Ryu, E. K. and Boyd, S. (2016). Primer on monotone operator methods. *Appl. comput. math*, 15(1):3–43.

Satorras, V. G., Hoogeboom, E., and Welling, M. (2021). E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR.

Shchur, O., Mumme, M., Bojchevski, A., and Günnemann, S. (2018). Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868.

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*.

Winston, E. and Kolter, J. Z. (2020). Monotone operator equilibrium networks. In *Advances in neural information processing systems*, pages 10718–10728.

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

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

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. (2021). Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, pages 28877–28888.

You, J., Ying, R., and Leskovec, J. (2020). Design space for graph neural networks. In *NeurIPS*.

Zhu, H. and Koniusz, P. (2021). Simple spectral graph convolution. In *International conference on learning representations*.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]
2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]
3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes]
   (d) Information about consent from data providers/curators. [Yes]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]
5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Appendix for

Monotone Operator Theory-Inspired Message Passing for Learning Long-Range Interaction on Graphs

The appendix is organized as follows. In Section A we provide the missing proofs for the theoretical results in the main paper. In Section B we provide some background knowledge on monotone operator theory and operator splitting schemes. In Section C we provide experimental results on an extended version of the multi-class Synthetic Chain classification task. In Section D we present more details about the experimental setup, including the dataset description in Section D.1, the model architecture details in Section D.2, the training procedure in Section D.3, and the details about hyperparameters in Section D.4.

A Proof of Proposition 1

Proof of Proposition 1. Given that $|\beta| < 1$ and $\alpha + \beta + 1 > 0$, we first show that the linear operator $\mathcal{F}$ is well defined, that is the matrix inverse $(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}})^{-1}$ exists. Note that $\boldsymbol{J}$ is a rank-one matrix with eigenvalues 0 with multiplicity $n-1$ and 1 with multiplicity 1 and with eigenvector $\mathrm{diag}(\tilde{\boldsymbol{D}}^{1/2})$. In particular, $\mathrm{diag}(\tilde{\boldsymbol{D}}^{1/2})$ is also an eigenvector of $\hat{\boldsymbol{A}}$ with eigenvalue 1. Hence, all the eigenvectors of $\hat{\boldsymbol{A}}$ are also eigenvectors of $\boldsymbol{J}$. Therefore the condition $|\beta| < 1$ and $\alpha + \beta + 1 > 0$ guarantees that the eigenvalues of $\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}}$ are all positive and hence the matrix inverse $(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}})^{-1}$ exists. This in particular shows that the operator $\mathcal{F}$ is well-defined.

Now we show that the linear operator $\mathcal{F}$ is strongly monotone whenever $\gamma > 1 + |\alpha| + |\beta|$. Let $\boldsymbol{F}$ be the linear part of $\mathcal{F}$, i.e., $\boldsymbol{F} := \gamma\left(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}}\right)^{-1} - \boldsymbol{I}_n$, which is symmetric. We then need to show that the symmetric matrix $\boldsymbol{F}$ is positive definite.

Let $-1 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n \leq 1$ be the eigenvalues of $\hat{\boldsymbol{A}}$ sorted in increasing order and the corresponding eigenvectors be $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$. Let n be the index such that $\lambda_n = 1$ with eigenvector $\mathrm{diag}(\tilde{\boldsymbol{D}}^{1/2})$. Then the eigenvalue η_i of $\boldsymbol{F}$ that corresponds with the eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ are given as follows: $\eta_i = \frac{\gamma}{1+\beta\lambda_i}$ when $i \neq n$ and $\eta_n = \frac{\gamma}{1+\alpha+\beta\lambda_i} - 1$. In particular, as $\gamma > 1 + |\alpha| + |\beta|$, we have $\eta_i > 0$ for each $i = 1, \ldots, n$. Therefore, $\boldsymbol{F}$ is positive definite, and hence $\mathcal{F}$ is strongly monotone.

Along with the Lipschitzness for any linear operator, $\mathcal{F}$, in this case, is both strongly monotone and Lipschitz. Consequently, its Cayley operator $\mathcal{C}_{\mathcal{F}}$ contractive, and hence the monotone inclusion problem Equation 5 admits a unique solution that can be solved using, e.g., Douglas-Rachford splitting (see Appendix B.3). □

B Some Background about Monotone Operator Theory and Operator Splitting

B.1 Monotone operators

This section presents some results about monotone operators. An operator $\mathcal{T}$ on $\mathbb{R}^d$ is said to be *monotone* if

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

Furthermore, we say the operator $\mathcal{T}$ is *maximal monotone* if there is no other monotone operator $\mathcal{S}$ s.t. the graph of $\mathcal{S}$ contains the graph of $\mathcal{T}$ properly. It is easy to verify that every continuous monotone function $\mathcal{F} : \mathbb{R}^d \to \mathbb{R}^d$ is maximal monotone. We call an operator $\mathcal{T} : \mathbb{R}^d \Rightarrow \mathbb{R}^d$ *m-strongly monotone* or m-coercive if $m > 0$ and

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

We say $\mathcal{T}$ is strongly monotone if it is m-strongly monotone for some unspecified constant $m \in (0, \infty)$. The following characterization of the monotonicity of linear and subdifferential operators is useful for our analysis of the monotonicity in Section 4.

- Let $\mathcal{F}(\boldsymbol{x}) = \boldsymbol{G}\boldsymbol{x} + \boldsymbol{h}$ be a linear operator where $\boldsymbol{G} \in \mathbb{R}^{d\times d}$ and $\boldsymbol{h} \in \mathbb{R}^d$. Then $\mathcal{F}$ is maximal monotone if and only if $\boldsymbol{G} + \boldsymbol{G}^\top$ is semi-positive definite, denoted as $\boldsymbol{G} + \boldsymbol{G}^\top \succeq \boldsymbol{0}$ ($\boldsymbol{0}$ stands for the matrix whose entries are all zero) and m-strongly monotone if $\frac{1}{2}(\boldsymbol{G} + \boldsymbol{G}^\top) \succeq m\boldsymbol{I}$, that is $\frac{1}{2}(\boldsymbol{G} + \boldsymbol{G}^\top) - m\boldsymbol{I}$ is semi-positive definite.

- Meanwhile, a subdifferential operator ∂f is maximal monotone if and only if f is a convex closed proper function.

Recall the resolvent and Cayley operator of an operator $\mathcal{G}$ are defined by

$$\mathcal{R}_{\mathcal{G}} = (\mathcal{I} + t\mathcal{F})^{-1}, \quad \mathcal{C}_{\mathcal{G}} = 2\mathcal{R}_{\mathcal{G}} - \mathcal{I},$$

for any t, where $\mathcal{I}$ is the identity operator. The following results for the resolvent of linear and subdifferential operators are used in Section 4:

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

$$\mathcal{R}_{\mathcal{F}}(\boldsymbol{x}) = \left(\boldsymbol{I}_n + t\boldsymbol{G}\right)^{-1}(\boldsymbol{x} - t\boldsymbol{h}).$$

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

$$\mathcal{R}_{\mathcal{F}}(\boldsymbol{x}) = \text{prox}_f^t(\boldsymbol{x}) := \arg\min_{\boldsymbol{z}} \left\{ \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{z}\|^2 + tf(\boldsymbol{z}) \right\}.$$

If an operator $\mathcal{F}$ is strongly monotone, its resolvent $\mathcal{R}_{\mathcal{F}}$ is contractive, and its Cayley operator is non-expansive (for any $t > 0$). If $\mathcal{F}$ is strongly monotone and Lipschitz, then its Cayley operator is also contractive, see (Ryu and Boyd, 2016, Section 6).

B.2 Forward-Backward splitting

This section provides additional details for Section 4 about the monotone inclusion problem and the Forward-Backward splitting method. We will first introduce the Forward-Backward splitting method and then show that the monotone inclusion problem is equivalent to the fixed-point problem of the layer update Equation 3. Then we derive the damped Picard iteration used in Equation 4.

Forward-backward (FB) splitting is a technique used to find a zero in the sum of operators, i.e., locate $\boldsymbol{x}$ such that

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

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

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

Where the last step uses the definition of the resolvent operator $\mathcal{R}_{\mathcal{G}} := (\mathcal{I} + t\mathcal{G})^{-1}$. Thus, $\boldsymbol{x}$ is a solution if and only if it is a fixed point of $\mathcal{R}_{\mathcal{G}}(\mathcal{I} - t\mathcal{F})$ for any t.

Equivalence between the fixed-point problem (Equation 3) and the monotone inclusion problem (Equation 5). Recall that the monotone inclusion problem used in Equation 5 reads

$$\mathcal{F}(\boldsymbol{H}) = \gamma\left(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}}\right)^{-1}\boldsymbol{H} - \boldsymbol{H} + f_\Theta(\boldsymbol{X}) \quad \text{and} \quad \mathcal{G} = \partial g,$$

where $\mathcal{G}$ is such that $\mathcal{R}_{\mathcal{G}}$ is the activation function σ. Then for any $t > 0$, the FB splitting updates as the following damped variant of Picard iteration

$$\boldsymbol{H}^{(i+1)} = \sigma\left(\boldsymbol{H}^{(i)} - t\left(\gamma\left(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}}\right)^{-1}\boldsymbol{H}^{(i)} - \boldsymbol{H}^{(i)} + f_\Theta(\boldsymbol{X})\right)\right), \tag{10}$$

$$= \sigma\left((1+t)\boldsymbol{H}^{(i)} - t\gamma\left(\boldsymbol{I}_n + \alpha\boldsymbol{J} + \beta\hat{\boldsymbol{A}}\right)^{-1}\boldsymbol{H}^{(i)} - tf_\Theta(\boldsymbol{X})\right). \tag{11}$$

Therefore, the FB splitting iteration recovers exactly the fixed-point formulation in Equation 3 when $t = 1$ and the fixed-point in Equation 3 can be computed via a suitable damping factor t.

B.3 Douglas-Rachford splitting

The Douglas-Rachford splitting method is derived by averaging the composition of two Cayley operators. For any $t > 0$, the following equivalence relation holds:

$$\mathbf{0} \in (\mathcal{F} + \mathcal{G})(\boldsymbol{x}) \iff \left(\frac{1}{2}\mathcal{I} + \frac{1}{2}\mathcal{C}_{\mathcal{F}}\mathcal{C}_{\mathcal{G}}\right)(\boldsymbol{y}) = \boldsymbol{y}, \quad \boldsymbol{x} = \mathcal{R}_{\mathcal{G}}(\boldsymbol{y})$$

This can be further expanded as:

$$\begin{aligned}
\mathbf{0} \in (\mathcal{F} + \mathcal{G})(\boldsymbol{x}) &\iff 0 \in (I + t\mathcal{F})\boldsymbol{x} - (I - t\mathcal{G})\boldsymbol{x} \\
&\iff 0 \in (I + t\mathcal{F})\boldsymbol{x} - \mathcal{C}_{\mathcal{G}}(I + t\mathcal{G})\boldsymbol{x} \\
&\iff 0 \in (I + t\mathcal{F})\boldsymbol{x} - \mathcal{C}_{\mathcal{G}}\boldsymbol{y}, \quad \boldsymbol{y} \in (I + t\mathcal{G})\boldsymbol{x} \\
&\iff \mathcal{C}_{\mathcal{G}}\boldsymbol{y} \in (I + t\mathcal{F})\mathcal{R}_{\mathcal{G}}\boldsymbol{y}, \quad \boldsymbol{x} = \mathcal{R}_{\mathcal{G}}\boldsymbol{y} \\
&\iff \mathcal{R}_{\mathcal{F}}\mathcal{C}_{\mathcal{G}}\boldsymbol{y} = \mathcal{R}_{\mathcal{G}}\boldsymbol{y}, \quad \boldsymbol{x} = \mathcal{R}_{\mathcal{G}}\boldsymbol{y} \\
&\iff \mathcal{C}_{\mathcal{F}}\mathcal{C}_{\mathcal{G}}\boldsymbol{y} = \boldsymbol{y}, \quad \boldsymbol{x} = \mathcal{R}_{\mathcal{G}}\boldsymbol{y} \\
&\iff \left(\frac{1}{2}\mathcal{I} + \frac{1}{2}\mathcal{C}_{\mathcal{F}}\mathcal{C}_{\mathcal{G}}\right)(\boldsymbol{y}) = \boldsymbol{y}, \quad \boldsymbol{x} = \mathcal{R}_{\mathcal{G}}(\boldsymbol{y})
\end{aligned}$$

Then the fixed point iteration for this setup is:

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

C Additional Experimental Results on Multi-class Chain Classification

In Figure 6 we extend the multi-class Synthetic Chain task in Figure 3 to more models. This includes baseline MP-GNNs GCNII (Chen et al., 2020), more multi-hop style MP-GNNs such as SGC (Wu et al., 2019), S^2GC (Zhu and Koniusz, 2021), GPRGNN (Chien et al., 2021), as well as fixed point graph neural networks EIGNN (Liu et al., 2021), GIND (Chen et al., 2022), MIGNN (Baker et al., 2023). We further include the model IGNN(Init) that represents the IGNN model with integrated technique as reusing fixed point initialization from the previous epoch. We observe that DRGNN outperforms all other models on this task, validating the model's ability to learn LRI effectively.

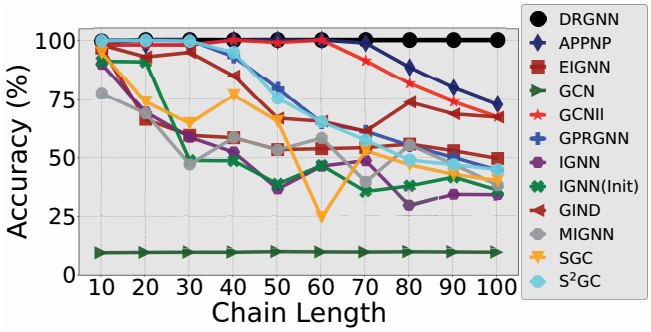


Figure 6: Node classification results on the multi-class Synthetic Chain task with 10 chains from 10 classes each. The length of each chain varies from 10 to 100. We observe that DRGNN outperforms all other models on this task validating the model's ability to learn LRI.

D Additional Experimental Details

D.1 Dataset details

Synthetic Chains. The Synthetic Chains dataset (Gu et al., 2020) is created to assess LRI learning capabilities of different models. The dataset is designed for both binary and multiclass classification tasks. Suppose there are c classes, meaning there are c distinct types of chains. Aside from the one-hot class indicating features on the first node of each chain, all other nodes only have zero features. Given c classes, n_c chains per class, and l

nodes per chain, the total number of nodes in the chain dataset is $c \times n_c \times l$. In this task, we use the 5/15/80% train/valid/test splits.

Color-counting. The Color-counting dataset (Liu et al., 2022) is introduced to further test the model's ability to learn LRI while processing multiscale information. The dataset contains a collection of chains, with each chain containing 10 randomly selected nodes that are colored into one of two colors. Each node's label is determined by the majority color of the chain containing it. The dataset is balanced with 50% of the nodes in each class. In this task, we use the 5/15/80% train/valid/test splits.

Peptides-func. The peptides-func is a multi-label graph classification task introduced in Dwivedi et al. (2022). The dataset consists of 15,535 graphs with a total of 2.3 million nodes. Each graph is constructed from a peptide sequence, where each node represents the heavy atom while the edges represent the chemical bonds between them. The OGB library (Hu et al., 2020) molecular features are used in this dataset. There is a total of 10 classes that represent peptide functions, e.g., anti-bacterial, anti-viral, cell-cell communication, etc. on average to 1.65 of the 10 classes. The labels are imbalanced, only 16.5% of the data is in the positive class, with the richest class having 62.7% positives and the poorest 1.9%. The license of this dataset is CC BY-NC 4.0.

Peptide-struct. The peptide-struct is also introduced in Dwivedi et al. (2022) and contains the same graphs as in the peptides-func dataset. This graph regression task aims to predict the peptide's graph level 3D structure, including inertia mass, inertia valence, length, sphericity, etc. The license of this dataset is CC BY-NC 4.0.

Transductive Datasets. The ogbn-arxiv is a large citation network with 169343 nodes and 1,166,243 edges introduced in (Hu et al., 2020). Each node is an arXiv computer science paper, and each directed edge indicates the source node cites the target node. The node feature is a 128-dimensional vector obtained from the word embedding of the title and abstract of its corresponding paper. The Coauthor CS and Physics datasets are introduced in (Shchur et al., 2018) represent co-authorship graphs from Microsoft Academic Graph. Each node represents an author, and each edge indicates two authors have co-authored a paper. The node feature represents the keywords of the corresponding authors' papers, and the label of the node is the author's most active field of study. The Amazon Computers and Amazon Photo datasets are introduced in (Shchur et al., 2018) and represent products bought together on Amazon. Each node represents a product, and each edge indicates two products are frequently bought together. The node feature is constructed from the bag-of-words of the product's reviews, and the label of the node is the product's category.

D.2 Model architectures

This section details the encoder, DRGNN, and decoder architectures used in Section 5. Additional hyperparameter details are provided in Appendix D.4.

The general architecture for DRGNN includes three auxiliary learnable parameters α, β, γ as in Equation 3 and a bias term $f_\Theta(\cdot)$. The forward method is then performed via Equation 7. Fixed point iteration is first calculated *without gradients* until convergence or the maximum number of iterations is reached. The iteration is called again *with gradients* until the number of phantom gradients is reached. The maximum number of forward iterations and the number of phantom gradient steps may be adjusted. The forward method also allows for an optional initial guess to the iterative solver, which can significantly improve the computational efficiency of DRGNN (see Section 5). DRGNN automatically stores an initial guess for static graphs. For dynamic graphs, it is recommended that this value is augmented to the data and passed to the model with the batched data. We provide a data augmentation transform using the PyG library (Fey and Lenssen, 2019) in the publically available code.

Synthetic Chains. In the Synthetic Chains dataset, we use a single linear layer without bias followed by a dropout layer for encoding. We then use the DRGNN layer with ReLU activation and a bias term of two sequential linear layers without bias separated by a ReLU activation function. The DRGNN layer is followed by a decoding process of a ReLU activation function followed by a linear layer without bias. Finally, model predictions are generated using a logarithmic softmax activation.

Color-counting. The color-counting dataset uses the same encoder and decoder architecture as the Synthetic Chains dataset. The DRGNN layer uses a ReLU activation function and a bias term of two sequential linear layers without bias separated by a ReLU activation function.

Peptides-func. The atomic information provided by the peptides-func dataset is first encoded using the Atom encoder as in (Dwivedi et al., 2022). Then the feature is passed through four layers of DRGNN with LeakyReLU activation and a GCN-style bias term. The decoder architecture is a two sequential linear layer. The hidden dimension is chosen to be 300 for all layers to fit within the 500k parameter limit. The key hyperparameters are provided in Table 3.

Peptides-struct. Similar to the peptides-func dataset, the atomic information provided by the peptides-struct dataset is first encoded using the Atom encoder as in (Dwivedi et al., 2022). Then the feature is passed through three layers of DRGNN with LeakyReLU activation and a GCN-style bias term. The decoder architecture is a three-sequential linear layer. The hidden dimension is chosen to be 300 for all layers to fit within the 500k parameter limit. The key hyperparameters are provided in Table 3.

Transductive Datasets. The transductive datasets use the same encoder and decoder architecture as the Synthetic Chains dataset. The DRGNN layer uses a LeakyReLU activation function.

D.3 Training procedure details

In this section, we provide the training details for the experiments in Section 5. All experiments are run using 8 NVIDIA RTX3090 GPUs and timing and memory experiments run on single A100 GPUs provided by Google Colab.

Synthetic Chains. The dataset is pre-processed by taking the symmetric normalized adjacency matrix. In these tasks, we utilize the Adam optimizer (Kingma and Ba, 2014), cross entropy loss, and early stopping based on the validation accuracy. Specifically, we train with a maximum of 2000 forward epochs with early stopping if the validation accuracy does not improve after 100 epochs. We save the model at the epoch with the best validation accuracy and use the saved model for testing.

Color-counting. The training procedure for the color-counting dataset is identical to the synthetic chains dataset.

Peptides-func and Peptides-struct. We follow the same training procedure as in (Dwivedi et al., 2022). The optimizer is AdamW (Loshchilov and Hutter, 2017) with a total of 250 epochs. The learning rate is halved when the validation loss does not improve after 20 epochs. The fixed 75/15/15% train/valid/test splits are used. The batch size is 128 for Peptides-func and Peptides-struct.

Transductive Datasets. The optimizer is AdamW (Loshchilov and Hutter, 2017) with a total of 100 epochs. The learning rate follows a cosine scheduler with 5 warmup epochs. The random 60/20/20% train/valid/test splits are used.

D.4 Hyperparameter details

To find the optimal hyperparameters of each model we use the hyperparameter searching range reported in Table 3. In the interest of space, when the spacing between values is linear, we use the notation {min:max:step}. For instance `dropout` may be abbreviated {0.0:0.8:0.1}. We use `lr` to denote the initial learning rate and `wd` to denote the weight decay. For DRGNN, we use $\alpha_{\texttt{init}}$ to denote the initialization of α, $\beta_{\texttt{init}}$ to denote the initialization of β, $\gamma_{\texttt{init}}$ to denote the initialization of γ. `max_iter` to denote the maximum number of forward iterations, `phantom_grad` to denote the number of steps used at the Phantom gradient stage, `tol` to denote the tolerance used for fixed-point iteration.

Synthetic Chains. For all models we use 16 hidden features. Each model is tuned using Bayesian meta-learning to maximize the test accuracy for 50 iterations. For binary classification, the models are tuned with a

Hyperparameter Search Range					
	lr	weight_decay		dropout	
	{1e-3, 5e-3, 8e-3, 9e-3, 1e-2}	{0, 1e-12, 1e-8, 1e-6, 1e-5, 1e-3, 1e-2}		{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8}	
Model	alpha	gamma	kappa	theta	K/layers/num_eigenvec
APPNP	{0.0:0.8:0.1}	—	—	—	{10:200:10}
EIGNN	—	{0.0:0.8:0.1}	—	—	{10:100:10}
GCN	—	—	—	—	{10:200:10}
GCNII	{0.0:0.8:0.1}	—	—	{0.0:0.8:0.1}	{10:200:10}
IGNN	—	—	{0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99}	—	—
SGC	—	—	—	—	{10:200:10}
S^{2}GC	{0.0:0.8:0.1}	—	—	—	{10:200:10}

Table 3: Search space for hyperparameter tuning using Bayesian meta-learning.

maximum of 200 aggregation steps, for multi-class classification the models are tuned at chain length 100. The optimal hyperparameters are shown in Table 4.

Model	Hyperparameters
APPNP	{lr : 0.01, wd : 0.00001, hidden_channels : 16, hidden_layers : 1, dropout : 0.8, alpha : 1.0, K : 200}
EIGNN	{lr : 0.01, wd : 0, hidden_channels : 16, dropout : 0.5, num_eigenvec : 100, gamma : 0.8}
GCN	{lr : 0.01, wd : 0.001, hidden_channels : 16, dropout : 0.0, hidden_layers : 150}
GCNII	{lr : 0.01, wd : 1e-5, hidden_channels : 16, dropout : 0.0, hidden_layers : 170, alpha : 0.0, theta : 0.0}
IGNN/IGNN(Init)	{lr : 0.01, wd : 1e-6, hidden_channels : 16, dropout : 0.5, kappa : 0.99}
SGC	{lr : 0.009, wd : 1e-8, hidden_channels : 16, dropout : 0.8, hidden_layers : 1, K : 190}
S^{2}GC	{lr : 0.009, wd : 1e-8, hidden_channels : 16, dropout : 0.3, hidden_layers : 1, alpha : 0.8, K : 200}
DRGNN (Ours)	{lr : 0.009, wd : 0, hidden_channels : 16, dropout : 0.2, phantom_grad : 10, beta : 0.9, gamma : 0.0}

Table 4: Synthetic Chains hyperparameters.

Color-counting. The same DRGNN hyperparameters are used in the color-counting task as in the Synthetic Chains task.

Peptides-func and Peptides-struct We report the parameter count and the hyperparameters for the best-performing models in Table 5. The hyperparameters are chosen based on the validation performance.

Dataset	Hyperparameters	Parameter Count
Peptides-func	{lr : 0.001, wd : 1e-8, beta_init : 2.0, dropout : 0.2, gamma_init : 0.9, hidden_channels : 300, max_iter : 50, phantom_grad : 5, tol : 5e-5}	507k
Peptides-struct	{lr : 0.0005, wd : 0, beta_init : 0.9, dropout : 0.3, gamma_init : 0.5, hidden_channels : 128, max_iter : 50, phantom_grad : 10, tol : 1e-5}	507k

Table 5: DRGNN hyperparameters on Peptides-func, Peptides-struct

Transductive Datasets. We report the hyperparameters for the best-performing models in Table 6. The hyperparameters are chosen based on the validation performance.

Dataset	Hyperparameters
ogbn-arxiv	{lr : 0.01, wd : 1e-3, alpha_init : -0.4, beta_init : 0.4, gamma_init : -10, dropout : 0.0, hidden_channels : 256, max_iter : 20, phantom_grad : 5, tol : 4e-5}
Computers	{lr : 0.01, wd : 1e-3, alpha_init : -0.4, beta_init : 0.9, gamma_init : 0.5, dropout : 0.4, hidden_channels : 512, max_iter : 25, phantom_grad : 3, tol : 4e-5}
Photo	{lr : 0.01, wd : 1e-3, alpha_init : -0.34, beta_init : 0.39, gamma_init : -9.42, dropout : 0.4, hidden_channels : 512, max_iter : 35, phantom_grad : 9, tol : 4e-5}
CS	{lr : 0.01, wd : 1e-12, alpha_init : -0.31, beta_init : 0.38, gamma_init : -20.0, dropout : 0.1, hidden_channels : 512, max_iter : 32, phantom_grad : 5, tol : 4e-5}
Physics	{lr : 0.01, wd : 1e-12, alpha_init : -0.31, beta_init : 0.38, gamma_init : -20.0, dropout : 0.1, hidden_channels : 512, max_iter : 32, phantom_grad : 5, tol : 4e-5}

Table 6: DRGNN hyperparameters on Transductive datasets.