GSN: A Universal Graph Neural Network Inspired by Spring Network

GUANYU CUI
Renmin University of China
cuiguanyu@ruc.edu.cn

ZHWEI WEI*
Renmin University of China
zhewei@ruc.edu.cn

Abstract

The design of universal Graph Neural Networks (GNNs) that operate on both homophilous and heterophilous graphs has received increased research attention in recent years. Existing heterophilous GNNs, particularly those designed in the spatial domain, lack a convincing theoretical or physical motivation. In this paper, we propose the Graph Spring Network (GSN), a universal GNN model that works for both homophilous and heterophilous graphs, inspired by spring networks and metric learning. We show that the GSN framework interprets many existing GNN models from the perspective of spring potential energy minimization with various metrics, which gives these models strong physical motivations. We also conduct extensive experiments to demonstrate our GSN framework’s superior performance on real-world homophilous and heterophilous datasets.

I. INTRODUCTION

With the growth of graph-structured data, Graph Neural Networks (GNNs) has recently emerged as a research hotspot in the machine learning field. In the computer vision field, Convolutional Neural Networks (CNNs, or ConvNets) showed significant advantages and made success on image classification tasks [13, 12, 24, 23, 8]. However, the spatial convolution operation defined on Euclidean data is difficult to be generalized to graphs. In order to generalize CNNs onto graph-structured data, [4] proposed ChebNet, which approximates the graph spectral convolution with Chebyshev polynomials. [10] simplifies ChebNet with the first-order Chebyshev polynomial and obtain the well-known Graph Convolutional Networks (GCN), which achieved state-of-the-art on semi-supervised node classification tasks. Following GCN, various GNN models are proposed to solve different graph related tasks, including node classification [10], physics simulation [22], traffic forecasting [15] and recommendation system [31, 27], etc. These GNN models can be roughly classified into four categories based on their design domain (spectral or spatial) and whether they support homophilous or heterophilous graphs (see Table 1).

GNNs designed for homophilous graphs. Inspired by the message-passing nature of GCN, many GNNs are designed in the spatial domain via different message-passing schemes. For example, GraphSAGE [7] focuses on inductive learning on large graphs using neighborhood sampling and the aggregator estimate method. Graph Attention Networks (GAT) [25] introduces attention mechanics in designing GNNs. Motivated by random walk on graphs, Personalized Propagation of Neural Predictions (PPNP) and its approximation version APPNP [11] use the PageRank proximity when designing their message-passing schemes. Simple Graph Convolution (SGC) [26] simplifies GCN by removing the non-linear activation functions between convolution layers. Some researchers notice that GCN loses expressive power rapidly when piling up too many layers due to the so-called over-smoothing phenomenon [14, 18]. GCNII [2] introduces initial residue and identity mapping into its message-passing scheme to relieve the over-smoothing phenomenon. Moreover, ElasticGNN [17] aims to enhance the local smoothness of GNN via $\ell_1$ based graph smoothing.

GNNs designed for heterophilous graphs. A possible explanation for the over-smoothing problem is that many GNN frameworks have the homophily assumption when aggregating neighbor nodes [32]. Therefore, it is practical to design GNNs for heterophilous graphs, which means nodes with different labels tend to have connections. To capture long-range context and to improve performance on heterophilous graphs, Geom-GCN [20] maps all nodes to a latent space and aggregates neighboring nodes in that space. H$_2$GCN [32] sep-
Table 1: A rough classification of Graph Neural Networks.

| Spectral Domain | Spatial Domain |
|-----------------|----------------|
| Homophilous     | GCN [10], SOC [26] |
| Heterophilous   | GraphSAGE [7], GAT [25], (A)PPNP [11], GCNII [2], ELASTIC-GNN [17] |
|                 | ChemNet [14], GPR-GNN [8], BernNet [2] |
|                 | Geom-GCN [13], H-GCN [14], TWIRLS [29], DMP [28], GSN (ours) |

arates ego-embedding with neighbor-embedding and considers higher-order neighborhoods in its message-passing scheme. Inspired by two classical iterative algorithms, TWIRLS [29] designs GNN layers to overcome the over-smoothing problem and generalize GNNs to heterophilous graphs. Diverse Message Passing (DMP) [28] tries to learn weights on each edge during the message-passing process. Another explanation for the over-smoothing problem is that the graph Laplacian behaves like a low-pass filter and smoothens the node features [14]. Therefore, learning a complex graph filter rather than designing it will alleviate this problem. Motivated by this, Generalized PageRank Graph Neural Networks (GPR-GNN) [3] introduces the Generalized PageRank (GPR) and allows the model to learn GPR coefficients. Moreover, BernNet [9] uses the Bernstein basis to approximate any arbitrary graph filter. In Section VII we will analyze these previous models in a more detailed manner.

While many GNNs for heterophilous graphs have been proposed, a common drawback of them, particularly those created in the spatial domain (e.g., H2GCN and DMP), is that they lack a strong physical motivation. In this paper, we show that spring network is a natural physical motivation for modeling both homophilous and heterophilous GNNs designed in spatial domain. In particular, we begin by discussing what constitutes an effective representation for GNNs. After that, we propose the Graph Spring Network (GSN) framework, which is a simple message-passing mechanism employs the spring’s physical nature to pull homophilous nodes together and pushes heterophilous nodes further away. We then study the invariance qualities of the energy function in GSN and provide a physical explanation for the rationale of the regularization term. Additionally, we demonstrate theoretically that several existing GNNs can be simulated interpreted by the GSN framework. Additionally, we conducted experiments on homophilous and heterophilous graphs to demonstrate the GSN framework’s effectiveness.

II. Preliminaries

Mathematical Notations. Capital letters in boldface, such as X, denote matrices, whereas lowercase characters, such as x, denote (column) vectors. For matrices, X_i denotes the i-th row of X, X_j denotes the j-th column of X, X_{ij} denotes the ij-th element of X, and X^T denotes the transposition of X. The trace of a matrix X, represented by tr(X), is the sum of all its diagonal elements, i.e. tr(X) := \sum_{ij} X_{ij}. The Frobenius norm of a matrix X, indicated by the symbol ||X||_F, is equal to the square root of the sum of the squares of all its elements, i.e. ||X||_F := \sqrt{\sum_{ij} |X_{ij}|^2}. For vectors, x_i denotes the i-th element or component of x, whereas x^T denotes the transposition of x. The p-norm ||x||_p of a vector x is defined as ||x||_p := (\sum_i |x_i|^p)^{1/p}. The Hardmard product is the element-wise product of two matrices or vectors. To be more precise, (X \circ Y)_{ij} = X_{ij}Y_{ij}, and (x \circ y)_i = x_iy_i.

Concepts from Graph Theory. A graph G is an ordered pair (\mathcal{V}, \mathcal{E}) comprising two sets, where \mathcal{V} representing the node set and \mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} representing the edge set. Given an arbitrary order for the nodes, we may denote the node set \mathcal{V} as \{1, 2, \ldots, |\mathcal{V}|\}, and use the adjacency matrix A \in \{0, 1\}^{\mathcal{V} \times \mathcal{V}} to represent the edges \mathcal{E}. The ij-th element of the adjacency matrix A_{ij} = 1 if edge (i, j) \in \mathcal{E}, otherwise A_{ij} = 0. Note that for an undirected graph, (i, j) \in \mathcal{E} \iff (j, i) \in \mathcal{E}, resulting in a symmetric adjacency matrix A. Unless otherwise noted, the graphs below are undirected. The neighborhood of a node i is defined as the set of nodes that share an edge with it, i.e. \mathcal{N}(i) := \{j : (i, j) \in \mathcal{E}\}, whereas the closed neighborhood of i is defined as \overline{\mathcal{N}(i)} := \mathcal{N}(i) \cup \{i\}. The degree of a node i is defined as the number of nodes in its neighborhood, i.e. d_i := |\mathcal{N}(i)| = |\{j : (i, j) \in \mathcal{E}\}|. And the degree ma-
where $Y$ is a vector. We occasionally use the symmetric normalized Laplacian $L$ to express the eigenvectors $\Lambda$ and $\lambda$.

The graph Laplacian has two essential and critical properties: it is positive semidefinite; and according to the spectral theorem, it can be decomposed as $L = U^T \Lambda U$, where $\Lambda = \text{diag}(\lambda_1, \lambda_2, \cdots, \lambda_{|V|})$ denotes the eigenvalues of the Laplacian, and $U$ consists of the eigenvectors of $L$. We sometimes refer to eigenvalues as frequencies. The graph Laplacian has two essential and critical properties: it is positive semidefinite; $\lambda_1 = 0$ is the smallest eigenvalue of $L$ and $1$ is the corresponding eigenvector.

We occasionally use the symmetric normalized Laplacian $L^{\text{sym}} := D^{-\frac{1}{2}}L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, and the symmetric normalized propagation matrix $P^{\text{sym}} = D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$.

III. Design Motivation

What Constitutes a Good Node Embedding? In this paper, we investigate the node classification problem by asking the following fundamental question:

Q: What constitutes a good node representation or embedding?

We can answer this question according to our intuition:

A: A good node embedding will draw any two connected homophilous nodes close and will push any two connected heterophilous nodes far away.

For example, in Figure 1 we have two isomorphic graphs where nodes are arranged as their embedding vectors. Intuitively, the right one is a better representation since it is easier to separate the nodes into different classes. In other words, it should decrease the intra-class distance while increasing the inter-class distance.

While existing GNN models obtain node embeddings on homophilous and heterophilous graphs using low- or high-pass filters, they may be unable to handle more complex graphs with coexisting homophilous and heterophilous edges. To see this, note that $1$ is the corresponding eigenvector of the smallest eigenvalue $\lambda_1 = 0$, so the perfectly smooth vector $1$ has the lowest frequency or energy. On the contrary, if a vector is not smooth, it will contain high-frequency components in the spectral domain. Convolution operation in GCN behaves like a low-pass filter, and the nodes’ features are smoothed each time they pass through a convolution layer. The convolution operation draws neighboring nodes closer if we treat every node feature as its spatial position in $\mathbb{R}^F$, consistent with the mean aggregation in GCN’s message-passing formula. On the other hand, a high-pass filter pushes all nodes further away. Therefore, we cannot just apply low- or high-pass filters to create a suitable GNN design for graphs with coexisting homophilous and heterophilous edges. In the spectral domain, a well-designed GNN should learn a complex filter; whereas in the spatial domain, it should learn a good push-pull relation between nodes.

Node Embeddings and Spring Networks. Recall that a good node embedding will draw any two connected homophilous nodes close and will push any two connected heterophilous nodes far away. A natural idea is to use spring to model the edge between homophilous or heterophilous nodes, which also push close objects further and pull distant objects closer. The spring network (or elastic network) models have been widely used in the biophysics and biochemistry fields to analyze a protein’s properties [6, 16, 1]. According to our previous discussion, the push and pull relationship we need to learn should act like follows: “similar objects appeal, dissimilar objects repel”, thus the spring network is an ideal tool to model homophily and heterophily of graphs.

To be concrete, we have a homophilous spring network and a heterophilous spring network in Figure 2 where the edges between nodes are springs. Initially, homophilous edges are extended springs, while het-
erophilous edges are compressed springs. Letting the spring network evolve freely, the total potential energy of the system is minimized. Nodes with the same attributes are pulled closer, while nodes with the different attributes are pushed further, making the nodes with different labels easier to be separated, regardless of the homophily of the graph.

Let us examine the physics behind spring networks. As an ideal physical model in physics, the spring is governed by Hooke's Law.

### Hooke's Law

The restoring force $F$ of a spring is proportional to the displaced amount $x$ (the length $l$ of the spring now minus the relaxed length $l_0$ of it) of the spring from its relaxed position, and the restoring force points towards the relaxed position, i.e., $F = -kx$.

We can easily calculate the potential energy a spring possesses when the displaced amount is $x$, by taking the integral of the restoring force w.r.t $x$:

$$E_p = -W_F = -\int_0^x Fdx = \frac{1}{2}k|x|^2.$$  

Note that there are two different position with the same energy, thus, the spring network is an excellent model for both homophilous and heterophilous graphs. In Figure 3 suppose we have two nodes whose coordinates are $r_1, r_2 \in \mathbb{R}^{d}$. According to Hooke's Law, the force node 1 exerts on node 2 is $F_{12} = k(||r_1 - r_2||_2 - l_0) \frac{r_1 - r_2}{||r_1 - r_2||_2}$, and the potential energy of this two-node system is

$$E_p = \frac{1}{2}k(||r_1 - r_2||_2 - l_0)^2. \quad (1)$$

Figure 3: Two nodes positioned in $r_1$ and $r_2$ with a compressed ($||r_1 - r_2||_2 < l_0$) spring connecting them. Each node feels a force pushing them further.

### IV. Potential Energy of Spring Network

Before presenting our GSN model, we first propose a potential energy function that models both homophilous and heterophilous graphs. We will also discuss the invariant property of it.

**Metric Matrix $M$.** For a graph $G = (\mathcal{V}, \mathcal{E})$ with an initial node embedding matrix $Z^{(0)}$, we can treat each edge $(i, j)$ as a spring whose initial length is $||Z_i^{(0)} - Z_j^{(0)}||_2$.

We assume each spring $(i, j)$ also stores an initial potential energy $E_{ij}$, resulting in a potential energy matrix $E \in \mathbb{R}^{n \times n}$ to be determined later. We also denote the relaxed length of each spring $(i, j)$ as $M_{ij}$, resulting in a relaxed length matrix, or the metric matrix $M \in \mathbb{R}^{n \times n}$. According to Equation 1 we have

$$E_{ij} = \frac{1}{2}k_{ij}(||Z_i^{(0)} - Z_j^{(0)}||_2 - M_{ij})^2,$$

then we have

$$M_{ij} = A_{ij}\left(||Z_i^{(0)} - Z_j^{(0)}||_2 + \sqrt{\frac{2E_{ij}}{k_{ij}}}\right),$$

and

$$= \begin{cases} 0, & (i, j) \notin \mathcal{E}, \\ ||Z_i^{(0)} - Z_j^{(0)}||_2 + \sqrt{\frac{2E_{ij}}{k_{ij}}}, & (i, j) \text{ compressed}, \\ ||Z_i^{(0)} - Z_j^{(0)}||_2 - \sqrt{\frac{2E_{ij}}{k_{ij}}}, & (i, j) \text{ stretched}. \end{cases}$$

Then, we introduce a soft variables called $\alpha_{ij} \in [-1, 1]$ to indicate whether the spring $(i, j)$ is initially stretched or compressed, and eliminate unnecessary variables $k_{ij}$ by reparameterizing $E_{ij} \equiv 2E_{ij}/k_{ij}$ for all $i$ and $j$. We obtain

$$M_{ij} = A_{ij}(||Z_i^{(0)} - Z_j^{(0)}||_2 - \alpha_{ij}\sqrt{E_{ij}}). \quad (2)$$
Note that $\alpha_{ij}$ approaches 1 when the spring $(i,j)$ is initially stretched and corresponds to an homophilous edge, whereas $\alpha_{ij}$ approaches $-1$ when the spring $(i,j)$ is initially compressed and corresponds to a heterophilous edge.

Designing Stretch/Compress Attention $\alpha_{ij}$. According to Equation 2 in order to calculate the metric matrix $M$, we have to derive $\alpha_{ij}$ first. Directly learning the $\alpha_{ij}$ on each edge is unfeasible, since it will introduce $|E|$ parameters, leading to overfitting. In GSN, we adopt three common attention methods to obtain $\alpha_{ij}$. Suppose that for an edge $(i,j)$, the embedding vectors of node $i$ and node $j$ are $h_i$ and $h_j$, and a function $g: \mathbb{R} \rightarrow [-1,1]$ such as $g(x) = \tanh(x)$ or $g(x) = 2\text{sigmoid}(x) - 1$ etc., we can get $\alpha_{ij}$ by

- “dot”: $\alpha_{ij} = g(h_i^\top h_j)$.
- “concat”: $\alpha_{ij} = g(a^\top [h_i^\top h_j])$, where $a \in \mathbb{R}^d$ is a parameter vector, and $\|$ denotes concatenation.
- “bilinear”: $\alpha_{ij} = g(h_i^\top Wh_j)$, where $W \in \mathbb{R}^{d \times d}$ is a parameter matrix.

Note that different from the GAT [25] and the TWIRLS [29], our edge attention $\alpha_{ij}$ can be negative, which corresponds to a compressed spring and heterophilous edge.

Designing Initial Energy Matrix $E$. To apply the spring network framework on GNN, we need to calculate the potential energy of the spring network for any intermediate node embedding $Z$, which requires the derivation of the initial energy matrix $E$. Intuitively, $E$ should be a function of the initial node embedding $Z^0$ and the Stretch/Compress Attention $\alpha_{ij}$. We will first discuss some physical and intuitive conditions that $E$ should fulfill:

1. (Non-negativity) Since energy in springs and the length of the springs are greater than 0, we have $E_{ij} \geq 0$ and $M_{ij} \geq 0 \iff ||Z_i^{(0)} - Z_j^{(0)}||_2 \geq \alpha_{ij} \sqrt{E_{ij}}$, leading to $0 \leq E_{ij} \leq \begin{cases} \frac{||Z_i^{(0)} - Z_j^{(0)}||_2^2}{\alpha_{ij}^2}, & 0 < \alpha_{ij} \leq 1 ; \\ +\infty, & -1 \leq \alpha_{ij} \leq 0 \end{cases}$

2. (Homophilous limit) We should minimize the distances of the nodes in the same class. In other words, when $\alpha \to 1^+$, the metric $M_{ij} \to 0$, i.e. $E_{ij} \to ||Z_i^{(0)} - Z_j^{(0)}||_2^2$.

3. (Heterophilous limit) We should maximize the distances of the nodes in different classes. In other words, when $\alpha \to -1^+$, $M_{ij} \to +\infty$ (or a large number), i.e. $E_{ij} \to +\infty$.

For simplicity, we set $E_{ij} = 4||Z_i^{(0)} - Z_j^{(0)}||_2$, which satisfies the above three conditions. It follows that $M_{ij} = ||Z_i^{(0)} - Z_j^{(0)}||_2 - 2\alpha_{ij}||Z_i^{(0)} - Z_j^{(0)}||_2 = \frac{1}{1-\alpha_{ij}}||Z_i^{(0)} - Z_j^{(0)}||_2^2$.

Embedding Potential Energy w.r.t a Metric $E_p(Z; \mathcal{G}, M)$. Having the metric matrix $M$, we can use Equation 4 to calculate the potential energy when the node embedding is $Z$,

$$E_p(Z; \mathcal{G}, M) = \sum_{(i,j) \in E} \frac{1}{2}k_{ij}(||Z_i - Z_j||_2 - M_{ij})^2$$

$$= \sum_{i,j} A_{ij}(||Z_i - Z_j||_2^2 - 2M_{ij}||Z_i - Z_j||_2 + M_{ij})$$

$$= \text{tr}(Z^\top LZ) - 2\sum_{i,j} M_{ij}||Z_i - Z_j||_2 + ||M||_F^2$$

$$= \text{tr}(Z^\top LZ) - 21^\top (M \odot D^2(Z) \odot \frac{1}{2}) 1 + ||M||_F^2,$$

where $\mathcal{D}^2: \mathbb{R}^{n \times d} \rightarrow \mathbb{R}^{n \times n}$, $Y \mapsto \text{diag}(YY^\top)1^\top + 1\text{diag}(YY^\top)^\top - 2YY^\top$ is the square of node embedding distance operation, i.e., $(\mathcal{D}^2(Y))_{ij} = ||Y_i - Y_j||_2^2$, and $Y\odot^k$ is the Hardmard (element-wise) power, that is, $(Y^\odot^k)_{ij} = (Y_{ij})^k$.

Obviously $E_p(Z; \mathcal{G}, M) \geq 0$ always holds, and $E_p = 0$ if and only if $\forall (i,j) \in E, ||Z_i - Z_j||_2 = M_{ij}$, i.e., the nodes are arranged in a way that the distances between them are the same with the metric matrix $M$.

**E(d) Invariance of $E_p(Z; \mathcal{G}, M)$**. Before we continue to the optimization objection of GSN, we examine the invariance property of the energy function $E_p(Z; \mathcal{G}, M)$ first. We will show that the energy function $E_p(Z; \mathcal{G}, M)$ is invariant under $d$-dimensional orthogonal (including rotation and reflection) transformations $O(d)$ and translation transformations $T(d)$, or $E(d)$ transformations for the two types of transformations. The critical observation is that $E_p$ is invariant under isometric, or distance-preserving transformations. For the convenience of discussion, we define the isometric transformations first.
Definition IV.1. A transformation \( T \) is called isometric w.r.t metric \( d \), if for any vector \( u, v \in \mathbb{R}^d \),
\[ d(T(u), T(v)) = d(u, v) \]
always holds.

For any matrix \( X \), we define the row-wise transformation \( T(X) := [T(X_{1,:}), T(X_{2,:}), \cdots, T(X_{n,:})]^\top \). We have the following theorem:

Theorem IV.2. For any node embedding matrix \( Z \), the energy function of GSN is invariant under any \( l_2 \)-isometric transformation \( T \), i.e., \( E_p(T(Z); \mathcal{G}, M) = E_p(Z; \mathcal{G}, M) \).

Proof. We can express the embedding potential energy as
\[
E_p(Z; \mathcal{G}, M) = \frac{1}{2} \sum_{(i,j) \in E} (\|Z_{ij} - Z_{ji}\|^2 - M_{ij})^2
\]
\[
= \frac{1}{2} \sum_{(i,j) \in E} (\|T(Z_{ij}) - T(Z_{ji})\|^2 - M_{ij})^2.
\]
\( T \) is isometric if \( \|T(Z) - T(Z')\|_2 = \|Z - Z'\|_2 \) for any \( Z, Z' \) (or the energy function has \( l_2 \)-isometric property), the transformation \( T \) preserves distances; and for any translation matrix \( S = 1s^\top \in \mathcal{T}(d) \) (where \( s \in \mathbb{R}^d \) stands for the translation vector), the transformation \( T(Z) = Z + S \) also preserves distances. Therefore, the energy function is invariant under any \( d \)-dimensional Euclidean transformation \( T(Z) = ZQ + 1s^\top \), or the energy function has \( E(d) \) invariance.

V. THE GSN MODEL

Below is the propagation rule of our GSN. \( L^\text{sym}_H \) is defined as \( D^{-\frac{1}{2}}LHD^{-\frac{1}{2}} \), where \( L_H = \text{diag}(H1) - H \) and \( H = M \odot \mathbb{Z}(D^{-\frac{1}{2}}Z)^\odot - \frac{1}{2} \). In this section, we will derive this rule from the energy function of GSN.

Optimization Objective. Many physical processes tend to minimize the total potential energy of the system. For example, rock tends to roll down to a lower position, which minimizes the gravitational potential energy. A physical law called Minimum Total Potential Energy Principle governs these processes, and the spring network is not an exception. Imagine that we put a spring network into a medium with resistance and let the system evolve freely. When the system is at the equilibrium state, the final embedding \( Z^* \) should be \( Z^* = \arg \min \max \ E_p(Z; \mathcal{G}, M) \). Recall that the energy function reaches its minimum zero when the nodes are arranged with the guidance of the metric matrix. Therefore, GSN aims to find a good metric between nodes and arrange them according to the metric in order to classify the nodes easily. This is the point at which GSN is inspired by metric learning.

On the other hand, many GNN models can be viewed as proximal descent methods that tries to minimize different energy functions on graphs \([29]\). For example, in Section [VII] we will show that many existing GNN models are designed with \( M = 0 \). Therefore, we can make reasonable speculation that the superior performance of GSN comes from this carefully designed energy function.

In order to be consistent with the symmetric normalized Laplacian, which is widely used in various GNN models such as GCN \([10]\), APPNP \([11]\), etc., we need to reparameterize \( Z \) as \( Z = D^{-1/2}Z \). Then we define the “normalized” energy as
\[
E_p(Z; \mathcal{G}, M) = \text{tr}(Z^\top L_Z) - 2\sum_{i,j} M_{ij} \|D^{-\frac{1}{2}}Z_{ij} - D^{-\frac{1}{2}}Z_{ji}\|^2 + \|M\|^2_F
\]
\[
= \text{tr}(Z^\top L^\text{sym}_Z) - 2\alpha^\top \left( M \odot D^2(D^{-\frac{1}{2}}Z)^\odot - \frac{1}{2} \right) 1 + \|M\|^2_F
\]
(3)

From Theorem [IV.2] above, we know that if \( Z^* \) minimizes \( E_p(Z; \mathcal{G}, M) \), then \( Z^* = Z^*Q + S \) also minimize the energy function. According to the locality concept in physics, we think the “best” solution should be close to the initial embedding \( Z(0) \), resulting in a trade-off regularization term \( \alpha \|Z - Z(0)\|_F^2 \). Now we have the optimization objection \( E = (1 - \alpha)E_p(Z; \mathcal{G}, M) + \alpha \|Z - Z(0)\|_F^2 \).

Propagation Rule of GSN

\[
Z^{(k+1)} = \sigma \left( \left( (1 - \alpha)P^\text{sym} + 2\beta L^\text{sym}_H \right) Z^{(k)} + \alpha Z^{(0)} \right) W^{(k)}.
\]
We define \[ W \] Therefore, the Theorem follows.

**Theorem V.1.** The propagation rule of GSN will minimize the energy function with regularization term \( E \).

**Proof.** First compute the gradient of \( E \) w.r.t the node embedding \( Z \), and we get (denoting \( H = M \odot \Omega^2(D^{-\frac{1}{2}}Z) \odot -\frac{1}{2} \) for short)

\[
\frac{\partial E}{\partial Z} = 2(1-\alpha) \left( L^\text{sym} - 2L^\text{sym} \right) Z + 2\alpha(Z - Z^{(0)}),
\]

where \( L_H \) stands for \( \text{diag}(H1) - H \), since if we treat \( H \) as an adjacency matrix of some graph, then \( \text{diag}(H1) - H \) can be though of the Laplacian of that graph.

By setting the gradient to be zero, it leads to

\[
\frac{\partial E}{\partial Z} = O
\]

\[
\Leftrightarrow (1-\alpha) \left( (I - P^\text{sym}) - 2L^\text{sym}_H \right) Z + \alpha(Z - Z^{(0)}) = O
\]

\[
\Leftrightarrow Z = (1-\alpha)P^\text{sym}Z + 2(1-\alpha)L^\text{sym}_H Z + \alpha Z^{(0)}
\]

We define \( P^\text{sym}Z \) part as “Topological Message”, and \( L^\text{sym}_H Z \) part as “Positional Message”. Finally, by decoupling coefficients of two types of messages, using the renormalization trick [10] to topological message part and introducing non-linearity, we obtain the update rule of a GSN Propagation layer:

\[
Z^{(k+1)} = \sigma \left( \left( (1-\alpha)P^\text{sym} + 2\beta L^\text{sym}_H \right) Z^{(k)} + \alpha Z^{(0)} \right) \cdot W^{(k)} \right). \tag{4}
\]

Therefore, the Theorem follows.

The derivation of the message passing scheme of GSN is provided in Appendix A.

**Time Complexity of the Message-passing Operation.** Assume that the arithmetic operations such as addition and multiplication between real numbers can be done in \( O(1) \) time. According to the message-passing scheme of GSN in Appendix A computing \( Z_i^{(k+1)} \) needs \( O(d) + |N(i)|O(d) + |N(i)|O(d) = 2(d_i + 1)O(d) \), where \( d \) is the dimension of the node embedding vector. Therefore, for the whole graph, we have

\[
T(\mathcal{G}) = \sum_{i \in \mathcal{V}} 2(d_i + 1)O(d)
\]

\[
= O(|\mathcal{E}| + |\mathcal{V}|d).
\]

**Structure of the Framework.** First, to obtain the initial embedding \( Z^{(0)} \), we need to map the node feature matrix \( X \) into the latent space and reduce its dimension. In Appendix C we list some existing methods to obtain the initial embedding of nodes. In our GSN framework, the node feature \( X \) is passed through a single layer linear layer or GCN layer (denoted as \( f(X; \mathcal{G}, \mathcal{W}) \)) to obtain the initial embedding \( Z^{(0)} \). Then it is passed through a metric layer to calculate the metric matrix between nodes. Meanwhile, the initial embedding is passed through \( K \) propagation layers. An illustration of the whole GSN framework structure is shown in Appendix A.

**VI. RELATED WORK AND UNIVERSALITY OF GSN**

In this section, we review some of the related GNN models and show that they can be simulated by GSN directly or under minor modification, which provides physical motivation for these models.

**Simulate GCN and SGC.** In Equation 4 if we set the metric matrix \( M = O \) (or \( \beta = 0 \)) and \( \alpha = 0 \), which means we only consider the topological message. It leads to \( Z^{(k+1)} = \sigma \left( PZ^{(k)} \cdot W^{(k)} \right) \), which is the same as the propagation rule of GCN [10]. SGC [26] removes the non-linear activation function in GCN; thus, it can be thought of as a special case of GSN, and our GSN framework can also simulate SGC by doing the same. If we treat GCN and SGC as the special cases of GSN, their metric \( M \) is \( O \). Therefore, if we stack too many layers of them, the distance between node \( i \) and \( j \) will be close in the final embedding, causing the over-smoothing problem.

**Simulate PPNP and APPNP.** If we only set the metric matrix \( M = O \) (or \( \beta = 0 \)), which means that we both care about the topological message and the initial embedding, and we do not care about the positional message. Moreover, we remove the non-linear activation functions and the linear transformations, which leads to APPNP’s propagation rule \( Z^{(k+1)} = (1-\alpha)PZ^{(k)} + \alpha Z^{(0)} \) in [11]. Since the APPNP model is the approximate version of PPNP, one can stack infinity APPNP layers to reach PPNP, and the GSN framework can also achieve this by doing the same. Compared to GCN and
SGC above, PPNP and APPNP introduce the regularization term $\|Z - Z^{(0)}\|_F^2$ which may alleviate the over-smoothing problem. However, it does not change the metric $M = O$; thus, it can not deal with heterophilous graphs.

**Modify GSN to Simulate GCNII.** Furthermore, we can modify our GSN framework to simulate GCNII proposed by [2]. We can introduce the identity mapping by substituting $W^{(k)}$ in Equation 3 with $((1 - \gamma)I + \beta W^{(k)})$, leading to the augmented propagation rule of GSN with identity mapping (which we name it GSN$^∗$). Note that when $\gamma = 1$, it degenerates to GSN. If we set $\beta = 0$ in GSN$^∗$’s propagation equation, we will get $Z^{(k+1)} = \sigma \left( \left( (1 - \alpha)PZ^{(k)} + \alpha Z^{(0)} \right) \left( (1 - \gamma)I + \gamma W^{(k)} \right) \right)$, which is the same as GCNII’s propagation rule. Therefore, the modified version GSN$^∗$ can simulate GCNII. GCNII can be regarded as APPNP with linear transformation and identity mapping, but it also does not change the metric, so it is not good at heterophilous graphs.

**Simulate ElasticGNN (with $\ell_{21}$ Regularization Term).** Interestingly, suppose we set $M = 11^T$ and add self-loops to the graph. In that case, the second term of our normalized energy $E_p(Z; G, M)$ will be $\sum_{(i,j) \in E} \|Z_i - Z_j\|_2$ (see Equation 3), which is the same as the $\ell_{21}$ regularization term $\|\Delta F\|_{21}$ defined in Elastic Graph Neural Network (ElasticGNN) [17]. Therefore, by changing the coefficients of regularization terms and setting $M = 11^T$, our GSN framework can also simulate the ElasticGNN with $\ell_{21}$ norm. However, the limitation of fixed $M$ also leads to its failure on the heterophilous datasets.

In Appendix D we compare our GSN with the models above and summarize the shortages and defects which possibly cause the over-smoothing problem. Furthermore, in Appendix E we will discuss how to modify our GSN to simulate GPR-GNN, ChebNet and BernNet.

**VII. Experiments**

**Datasets.** For homophilous graphs, we use three classic citation networks: CORA, CiteSeer, and PubMed from the paper [30]. For heterophilous graphs, we choose the WebKB networks: Cornell, Texas, and Wisconsin from [20], and the Wikipedia networks: Chameleon, and Squirrel from [21]. The statistics of the datasets are listed in Table 2.

| Dataset  | $|V|$ | $|E|$ | $F$ | $C$ | $\mathcal{H}$ |
|----------|------|------|-----|-----|------------|
| CORA     | 2,708| 5,429| 1,433| 7   | 0.81       |
| CiteSeer | 3,327| 4,732| 3,703| 6   | 0.74       |
| PubMed   | 19,717| 44,338| 500 | 3   | 0.80       |
| Cornell  | 183  | 298  | 1,703| 5   | 0.30       |
| Texas    | 183  | 325  | 1,703| 5   | 0.09       |
| Wisconsin| 251  | 515  | 1,703| 5   | 0.19       |
| Chameleon| 2,277| 36,101| 2,325| 5   | 0.23       |
| Squirrel | 5,201| 217,073| 2,089| 5   | 0.22       |

1 Originally for regression, preprocessed as in [20].

**Baseline Models.** To compare our GSN framework with previous works, we choose models including GCN [10], GraphSAGE [7], GAT [25], GCNII [2], H2GCN [22], GPR-GNN [3] and DMP-1/2-Sum [28] as baselines. We implement our GSN model in Python with the PyTorch and PyTorch Geometric [5] libraries.

**Supervised Node Classification.** We run supervised node classification experiments on baseline models and our GSN framework. Following the experiment setting in [28], we randomly generate ten train/validation/test splits with the 48%/32%/20% ratio for all datasets. We train our model for 500 epochs on a split and repeat it ten times for a specific hyperparameter group.

Table 3 lists the classification accuracy on homophilous and heterophilous datasets. Our GSN framework performs well both on homophilous and heterophilous datasets. Notably, on the two large heterophilous graphs Chameleon and Squirrel, GSN outperforms all competitors by about 9% and 15%, demonstrating its ability to handle heterophilous graphs.

**Alleviating the Over-smoothing Problem.** We also perform experiments on GCN, GCNII, and GSN of 4, 16 and 64 layers with 64 hidden channels on three relatively large datasets: PubMed, Chameleon and Squirrel.
Table 3: Mean accuracy (%) of full-supervised node classification tasks. The results of GCN, GraphSAGE, GAT, H₂GCN, GPR-GNN and DMP are taken from [28]. We use bold font to highlight the best result, and use underline for the second.

|       | CORA  | CiteSeer | PubMed | CORNELL | TEXAS  | WISCONSIN | CHAM. | SQUIRREL |
|-------|-------|----------|--------|---------|--------|------------|-------|----------|
| GCN   | 85.51 | 75.53    | 84.71  | 54.05   | 64.86  | 56.86      | 53.51 | 32.28    |
| GraphSAGE | 86.90 | 76.04    | 88.45  | 75.95   | 82.43  | 81.18      | 58.73 | 41.61    |
| GAT  | 82.68 | 75.46    | 84.68  | 58.92   | 58.38  | 55.29      | 54.69 | 30.62    |
| GCNII | 87.90 | 75.14    | 88.47  | 72.97   | 73.78  | 81.00      | 61.32 | 40.42    |
| H₂GCN | 86.92 | 77.04    | 89.40  | 82.16   | 84.86  | 86.67      | 57.11 | 36.42    |
| GPR-GNN | 88.14 | 74.07    | 88.27  | 91.14   | 90.49  | 85.33      | 51.63 | 32.44    |
| DMP-1-Sum | 82.28 | 76.13    | 88.13  | 70.27   | 86.48  | 86.27      | 50.21 | 43.42    |
| DMP-2-Sum | 85.31 | 76.27    | 88.15  | 83.78   | 78.37  | 84.31      | 55.92 | 32.18    |
| GSN (ours) | 87.99±1.04 | 76.72±1.74 | 89.58±0.70 | 84.32±4.15 | 86.49±6.07 | 86.40±4.72 | 70.11±2.19 | 58.66±1.09 |

Table 4: Mean accuracy (%) of three models with various layers. ↑ (%) denotes the accuracy of the 64-layer model minus that of the 4-layer model.

| Dataset | Method | Layers | ↑ (%) |
|---------|--------|--------|-------|
| CORNELL | GCN    | 85.38  | 41.40 |
|         | GCNII  | 88.03  | 41.40 |
|         | GSN    | 88.32  | 41.40 |
| TEXAS   | GCN    | 87.54  | 41.66 |
|         | GCNII  | 61.34  | 59.19 |
|         | GSN    | 70.07  | 69.96 |
| WISCONSIN | GCN    | 42.77  | 31.78 |
|         | GCNII  | 39.26  | 31.78 |
|         | GSN    | 58.02  | 58.10 |

Ablation Study  The main difference between GCNII and GSN is that GSN introduces the positional message term $L^\text{sym}_H Z$. In order to show the effectiveness of our design, we conduct experiments with GSN on three small heterophilous datasets. We use a 2-layer GSN with 64 hidden units, fix $\alpha = 0.2$ and vary the hyperparameter $\beta$ from 0 to 0.5 to adapt the ratio of the positional message in the message-passing process.

Table 5: Mean accuracy (%) of GSN on three heterophilous datasets with $\alpha = 0.1$ and various $\beta$.

| Dataset | 0   | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
|---------|-----|-----|-----|-----|-----|-----|
| CORNELL | 67.03 | 73.51 | 75.95 | 73.24 | 71.62 | 69.73 |
| TEXAS   | 62.43 | 64.86 | 76.22 | 79.46 | 80.81 | 82.16 |
| WISCONSIN | 66.80 | 70.20 | 76.40 | 80.60 | 77.60 | 80.20 |

VIII. Conclusion

In this paper, inspired and motivated by the famous spring network model, we propose a GNN framework called Graph Spring Network. We discover many beautiful properties, including invariance of the energy function and universality of GSN through mathematical derivations. We have also done experiments with our GSN, and the results confirm the superior performance of it.
REFERENCES

[1] Romain Amyot, Yuichi Togashi, and Holger Flechsig. Analyzing fluctuation properties in protein elastic networks with sequence-specific and distance-dependent interactions. *Biomolecules*, 9(10), 2019.

[2] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[3] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. Adaptive universal generalized pagerank graph neural network. In *International Conference on Learning Representations*, 2021.

[4] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, 2016.

[5] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[6] Turkan Haliloglu, Ivet Bahar, and Burak Erman. Gaussian dynamics of folded proteins. *Phys. Rev. Lett.*, 79:3090–3093, Oct 1997.

[7] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[9] Mingguo He, Zhewei Wei, Zengfeng Huang, and Hongteng Xu. Bernnet: Learning arbitrary graph spectral filters via bernstein approximation. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[10] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[11] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations (ICLR)*, 2019.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] Q. Li, Z. Han, and X.-M. Wu. Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning. In *The Thirty-Second AAAI Conference on Artificial Intelligence*. AAAI, 2018.

[15] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *International Conference on Learning Representations*, 2018.

[16] Tu-Liang Lin and Guang Song. Generalized spring tensor models for protein fluctuation dynamics and conformation changes. In *2009 IEEE International Conference on Bioinformatics and Biomedicine Workshop*, pages 136–143, 2009.

[17] Xiaorui Liu, Wei Jin, Yao Ma, Yaxin Li, Hua Liu, Yiqi Wang, Ming Yan, and Jiliang Tang. Elastic graph neural networks. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[18] Kenta Oono and Taiji Suzuki. Graph neural networks exponentially lose expressive power for node classification. In *International Conference on Learning Representations*, 2020.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.
[20] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In International Conference on Learning Representations, 2020.

[21] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-Scale attributed node embedding. Journal of Complex Networks, 9(2), 05 2021.

[22] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks. In International Conference on Machine Learning, 2020.

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In International Conference on Learning Representations, 2015.

[24] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1–9, 2015.

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. International Conference on Learning Representations, 2018.

[26] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In Proceedings of the 36th International Conference on Machine Learning, pages 6861–6871. PMLR, 2019.

[27] Shu Wu, Yuyuan Tang, Yanqiao Zhu, Liang Wang, Xing Xie, and Tieniu Tan. Session-based recommendation with graph neural networks. In The Thirty-Third AAAI Conference on Artificial Intelligence, pages 346–353, 2019.

[28] Liang Yang, Mengzhe Li, Liyang Liu, Bingxin Niu, Chuan Wang, Xiaochun Cao, and Yuanfang Guo. Diverse message passing for attribute with heterophily. In Advances in Neural Information Processing Systems, 2021.

[29] Yongyi Yang, Tang Liu, Yangkun Wang, Jinjing Zhou, Quan Gan, Zhewei Wei, Zheng Zhang, Zengfeng Huang, and David Wipf. Graph neural networks inspired by classical iterative algorithms. In Proceedings of the 38th International Conference on Machine Learning, 2021.

[30] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting Semi-Supervised Learning with Graph Embeddings. In Proceedings of the 33rd International Conference on Machine Learning, pages 40–48. JMLR.org, 2016.

[31] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 974–983, 2018.

[32] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. Advances in Neural Information Processing Systems, 33, 2020.
A. AN ILLUSTRATION OF GSN’S STRUCTURE

Figure 4 is an illustration of the structure of the whole GSN framework.

![Figure 4: Structure of the GSN framework.](image)

B. DERIVATION OF MESSAGE-PASSING SCHEME OF GSN

Given the matrix-form update rule of a GSN layer (ignore the linear transformation and non-linear activation function first),

$$Z^{(k+1)} = (1 - \alpha)P_{sym}^{(k)}Z^{(k)} + 2\beta L_{H}^{sym} Z^{(k)} + \alpha Z^{(0)},$$

we are interested in the message-passing scheme of GSN. For an arbitrary node $i$, we have

$$Z_{i}^{(k+1)} = \alpha Z_{i}^{(0)} + (1 - \alpha) \sum_{j \in \mathcal{N}(i)} \frac{Z_{j}^{(k)}}{\sqrt{D_{ii}D_{jj}}} + 2\beta \sum_{j=1}^{(|V|)} (L_{H}^{sym})_{ij} Z_{j}^{(k)}.$$

Note that $L_{H}^{sym} = D^{-\frac{1}{2}}L_{H}D^{-\frac{1}{2}}$, $L_{H} = \text{diag}(H1) - H$, and $H = M \odot \mathcal{D}^2(D^{-1}Z)$, thus

$$(L_{H}^{sym})_{ij} = \begin{cases} \frac{\sum_{k \neq j} M_{ik} \left( \frac{Z_{i}}{\sqrt{D_{ii}}} - \frac{Z_{k}}{\sqrt{D_{kk}}} \right)}{\sqrt{D_{ii}D_{jj}}} , & i = j \\ \frac{\sum_{k \neq j} M_{ij} \left( \frac{Z_{i}}{\sqrt{D_{ii}}} - \frac{Z_{j}}{\sqrt{D_{jj}}} \right)}{\sqrt{D_{ii}D_{jj}}} , & i \neq j, (i,j) \in E \\ 0, & \text{otherwise}. \end{cases}$$
Then the third term can be rearranged as

\[
\sum_{j=1}^{\lvert V \rvert} (L^\text{sym}_H)^{ij} Z^{(k)}_j = (L^\text{sym}_H)^{ii} Z^{(k)}_i + \sum_{j=1, \neq i}^{\lvert V \rvert} (L^\text{sym}_H)^{ij} Z^{(k)}_j
\]

\[
= \sum_{j \in N(i)} \frac{M_{ij}}{\sqrt{D_{ii} D_{jj}}} \| \frac{Z^{(k)}_i - Z^{(k)}_j}{\sqrt{D_{ii}}} \|_2
\]

Therefore, we have the message-passing scheme of GSN:

\[
Z^{(k+1)}_i = a Z^{(0)}_i + (1 - a) \sum_{j \in N(i)} \frac{Z^{(k)}_j}{\sqrt{D_{ii} D_{jj}}} + 2\beta \sum_{j \in N(i)} \frac{M_{ij}(Z^{(k)}_i - Z^{(k)}_j)}{\sqrt{D_{ii} D_{jj}}} \| \frac{Z^{(k)}_i - Z^{(k)}_j}{\sqrt{D_{ii}}} \|_2.
\]

C. METHODS TO OBTAIN THE INITIAL EMBEDDING

There are many different methods to obtain the initial embedding of nodes in various existing models. For example, GCN \[10\] directly set \(Z^{(0)} = X\) and the dimension of the matrix is reduced using a trainable matrix \(W^{(0)}\) in the first convolution layer; PPNP, APPNP \[11\] and GPR-GNN \[3\] use a neural network \(f_\theta\) to get initial embedding; H2GCN \[32\] uses a graph-agnostic dense layer to generate node embedding, i.e., \(Z^{(0)} = \sigma(XW_\varepsilon)\); and BernNet \[9\] uses a 2-layer MLP to do the same.

D. ANALYSIS OF POSSIBLE DEFECT OF EXISTING MODELS

Some previous works, such as \[14\] and \[18\], try to explain why GCN-like models have over-smoothing problems from the perspective of Laplacian smoothing or dynamic systems. Here we give another possible explanation from a different perspective.

In Section VI we have concluded that many GNN models can be regarded as minimizing the potential energy of a spring network with respect to a given metric \(M\). For GCN and SGC, their metric matrices are simply \(M = 0\). Thus, if we stack too many layers, the distance between node \(i\) and \(j\) is close in the final embedding, making the nodes hard to appropriately classified.

PPNP and APPNP are equivalent to adding a trade-off regularization term \(\|Z - Z^{(0)}\|_2^2\) into energy function, which leads to the initial residue \(Z^{(0)}\) in its propagation rule \(Z^{(k+1)} = (1 - \alpha)PZ^{(k)} + \alpha Z^{(0)}\). This regularization term may alleviate the node embeddings to be too close, but it never changes the metric \(M = 0\) between nodes.

Compared with APPNP, GCNII introduces identity mapping into its framework, which alleviates the over-smoothing problem in graph learning. However, it does not change the metric \(M\) to a non-zero matrix, which may be why GCNII does not perform well on heterophilous graphs.

ElasticGNN introduces \(\ell_{21}\) regularization term in its optimization objection. However, the authors do not realize the physical meaning of \(\ell_{21}\) regularization term, it is, in fact, equivalent to a special case (fix \(M = 11^T\)) of our GSN. This shortage may be the reason why ElasticGNN (\(\ell_{21}\)) does not perform well on heterophilous graphs.
Inspired by spring networks and metric learning, our GSN framework first change the target metric between nodes to a non-zero matrix, which may benefit the performance of GNNs on heterophilous datasets. Meanwhile, our GSN framework also keeps the initial residue module, which alleviates the over-smoothing problem.

Table 6 lists the components and targets of some GNN models.

Table 6: Components and targets of some GNN models.

| Components / Targets | GCN | SGC | (A)PPNP | GCNII | ElasticGNN (ℓ_21) | GSN | GSN^+ |
|----------------------|-----|-----|---------|-------|-------------------|-----|-------|
| Non-zero metric? (M) | ✓   | ✓   | ✓       | X     | ✓, but limited to 11 | ✓   | ✓     |
| Initial residue? (Z(0)) | X   | X   | ✓       | ✓     | ✓                | ✓   | ✓     |
| Identity mapping?     | ✓   | ✓   | ✓       | ✓     | ✓                | ✓   | ✓     |
| Linear transformations? (W(k)) | ✓ | ✓   | ✓       | ✓     | ✓                | ✓   | ✓     |
| Non-linear activation functions? (σ) | ✓ | ✓   | ✓       | ✓     | ✓                | ✓   | ✓     |
| Designed for homophilous graphs? | ✓ | ✓   | ✓       | ✓     | ✓                | ✓   | ✓     |
| Designed for heterophilous graphs? | ✓ | ✓   | X       | X     | X                | ✓   | ✓     |

E. More on the Universality of GSN

Generalize GSN to Simulate GPR-GNN. We can also generalize the GSN framework to simulate a recent heterophilous GNN framework - GPR-GNN proposed in [3]. We remove linear transformations and non-linear activation functions, set β = 0 in GSN, then allow each layer to learn different coefficients of PZ(k) and Z(0). As a result, we get the propagation rule

\[ Z^{(k)} = a_{k-1}PZ^{(k-1)} + b_{k-1}Z^{(0)}. \]

Inductively using the propagation rule for K times, we get the following equation (we define b_1 = 1):

\[ Z^{(K)} = \sum_{k=0}^{K} \left( \prod_{i=K-k}^{K-1} a_i \right) b_{K-1-k}P^kZ^{(0)}. \]

Compare it with a K layer GPR-GNN’s propagation rule

\[ Z^{(K)} = \sum_{k=0}^{K} \gamma_kP^kZ^{(0)} \]

from [3], if for any series of \( \gamma_k \) there exists corresponding \( a_k \) and \( b_k \), such that \( \forall k \in \{0, 1, \ldots, K\}, \)

\[ \left( \prod_{i=K-k}^{K-1} a_i \right) b_{K-1-k} = \gamma_k \]

always holds, then GSN can simulate GPR-GNN. Obviously, if we set

\[ a_k = \begin{cases} \frac{\gamma_{k+1}}{\gamma_{K-k}}, & 0 \leq k < K-1 \\ \gamma_1, & k = K-1 \end{cases} \]

and

\[ b_k = \begin{cases} \text{sign}(\gamma_{K-k}), & 0 \leq k < K-1 \\ \gamma_0, & k = K-1 \end{cases} \]

the equation above holds; thus the modified GSN can also simulate GPR-GNN.
Simulate ChebNet and BernNet. From the discussions of the relation between GSN and GPR-GNN, we have known that our GSN can simulate any (spectral) polynomial filter by letting each propagation layer learn its coefficients. We examine the other two GNN models designed in the spectral domain: ChebNet [4] and BernNet [9]. The propagation rule of a $K$-order ChebNet can be formulated as

$$Z = \sum_{k=0}^{K} T_k(L)Z(0)W(k)$$

(where $T_k$ is the $k$-th order Chebyshev polynomial, that is, a polynomial with order $k$), and that for a $K$-order BernNet is

$$Z = \sum_{k=0}^{K} \frac{1}{2^k} \binom{K}{k} (2I - L)^{K-k}L^kZ(0).$$

Note that they are polynomial filters with order $K$ (if we ignore the linear transformation in ChebNet), our GSN can simulate them by allowing each propagation layer to learn their coefficients independently.

F. Hyperparameter Settings

Our hyperparameter selection setting is:

- **num_layers**: $\{2, 4\}$ for all datasets,
- **hidden**: $\{32, 64\}$ for all datasets,
- **learning_rate**: $\{1e^{-3}, 5e^{-3}, 1e^{-2}\}$ for all datasets,
- **dropout**: $\{0.15, 0.25, \cdots, 0.65\}$ for all datasets,
- **weight_decay**: $\{1e^{-6}, 1e^{-4}\}$ for all datasets,
- **attention_method**: ‘concat’ for all datasets,
- **alpha**: $\{0, 0.1, \cdots, 0.9\}$ for all datasets,
- **beta**: $\{0, 0.1, 0.2, \cdots, 0.5\}$ for all datasets,