Sparse Graph Learning with Eigen-gap for Spectral Filter Training in Graph Convolutional Networks

Jin Zeng
Dept. of Software Eng.
Tongji University
Shanghai, China
zengjin@tongji.edu.cn

Saghar Bagheri
Dept. of EECS
York University
Toronto, Canada
sagharb@yorku.ca

Yang Liu
WICT
Peking University
Beijing, China
1800011037@pku.edu.cn

Gene Cheung
Dept. of EECS
York University
Toronto, Canada
genec@yorku.ca

Wei Hu
WICT
Peking University
Beijing, China
forhuwei@pku.edu.cn

Abstract—It is now known that the expressive power of graph convolutional neural nets (GCN) does not grow infinitely with the number of layers. Instead, the GCN output approaches a subspace spanned by the first eigenvector of the normalized graph Laplacian matrix with the convergence rate characterized by the “eigen-gap”: the difference between the Laplacian’s first two distinct eigenvalues. To promote a deeper GCN architecture with sufficient expressiveness, in this paper, given an empirical covariance matrix \( C \) computed from observable data, we learn a sparse graph Laplacian matrix \( \tilde{L} \) closest to \( C^{-1} \) while maintaining a desirable eigen-gap that slows down convergence. Specifically, we first define a sparse graph learning problem with constraints on the first eigenvector (the most common signal) and the eigen-gap. We solve the corresponding dual problem greedily, where a locally optimal eigen-pair is computed one at a time via a fast approximation of a semi-definite programming (SDP) formulation. The computed \( \tilde{L} \) with the desired eigen-gap is normalized spectrally and used for supervised training of GCN for a targeted task. Experiments show that our proposal produced deeper GCNs and smaller errors compared to a competing scheme without explicit eigen-gap optimization.

Index Terms—Sparse graph learning, graph convolutional networks, graph signal processing

I. INTRODUCTION

Given an underlying graph structure, graph convolutional networks (GCN) [1] performs graph filtering and point-wise non-linear operation (e.g., ReLU) in a sequence of neural layers for a range of tasks, such as graph signal interpolation, denoising, and node classification [1]–[3]. It has been observed, however, that the expressive power of GCN does not grow infinitely with the number of layers; for a given task, a GCN starts to oversmooth as the number of layers grows, resulting in stagnant or even worsening performance [4], [5].

This undesirable phenomenon has been empirically observed [6] and studied analytically recently [7], [8]. Specifically, [7] proved that the GCN output approaches a subspace spanned by the first eigenvector of the normalized graph Laplacian matrix \( \tilde{L} \) with the convergence rate characterized by the “eigen-gap”: the difference between the first and second dominant eigenvalues\(^2\) of matrix \( P = I - \tilde{L} \)—typically the first two eigenvalues of \( \tilde{L} \). Given that sparser graphs in general have smaller eigen-gaps (e.g., a complete unweighted graph has maximum eigen-gap of 2), [7] showed that for random Erdős-Rényi graphs with edge probability \( p \), sparser graphs (smaller \( p \)) converge to the aforementioned subspace at a slower pace. Similar in approach to achieve graph sparsity, [8] randomly removed edges from a pre-chosen graph topology in layers during training, resulting in more expressiveness in the trained GCNs.

Orthogonally, graph signal processing (GSP) studies discrete signals residing on combinatorial graphs [11], [12]. A key assumption in GSP is that an underlying similarity graph capturing pairwise correlations is available as input, before spectral filters are designed and applied to signals on top based on spectral graph theory [13]. Given a training set of graph signal observations generated from the same statistical model, there exist many graph learning algorithms [14]–[16] that compute a most likely sparse inverse covariance matrix (interpreted as a generalized graph Laplacian matrix), which is subsequently used for GSP processing like compression [17], [18], denoising [19], [20] and interpolation [21].

In this paper, leveraging a recent spectral graph learning scheme [22], we learn a sparse graph from empirical data with a tunable eigen-gap, in order to promote expressiveness in the trained GCN and optimize performance. Unlike works [7], [8] that indirectly induced eigen-gaps by sparsifying graphs heuristically in the nodal domain, we directly engineer an eigen-gap by optimizing one eigen-pair of the graph Laplacian at a time in the spectral domain.

Specifically, we first formulate a sparse graph learning problem similar to graphical lasso (GLASSO) [14], but with additional constraints on the first eigenvector (the most common signal) and the eigen-gap. We solve the corresponding dual problem greedily, where each locally optimal eigen-pair is computed via a fast approximation of a semi-definite programming relaxation (SDR) [23]. The learned graph Laplacian \( \tilde{L} \) with the desired eigen-gap is normalized spectrally and used for supervised training of GCN at a targeted task. Experimental results show that when the pre-set eigen-gap was

---

\(^1\)A dominant eigenvalue is the largest eigenvalue in magnitude.

\(^2\)The relationship between convergence rate and eigen-gap of a matrix is found also in Perron-Frobenius theorem for a discrete-time Markov chain [9] and the power iteration method in numerical linear algebra [10].
smaller, the optimal number of GCN network layers increased. Further, compared to [8], our computed graph achieved smaller predictive error, since by creating an eigen-gap in the spectral domain, we avoided random dropping of strong correlation edges that are important for graph filtering.

II. PRELIMINARIES

A. Graph Definitions

A graph \( \mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{W}) \) is defined by a set of \( N \) nodes \( \mathcal{V} = \{1, \ldots, N\} \), edges \( \mathcal{E} = \{(i, j)\} \), and an adjacency matrix \( \mathcal{W} \). \( W_{i,j} \in \mathbb{R} \) is the edge weight if \( (i, j) \in \mathcal{E} \), and \( W_{i,j} = 0 \) otherwise. Self-loops may exist, in which case \( W_{i,i} \in \mathbb{R}^+ \) is the weight of the self-loop for node \( i \). Degree matrix \( \mathcal{D} \) has diagonal entries \( D_{i,i} = \sum_j W_{i,j} \). A combinatorial graph Laplacian matrix \( \mathcal{L} \) is defined as \( \mathcal{L} = \mathcal{D} - \mathcal{W} \), which is positive semi-definite (PSD) for a positive graph [12]. If self-loops exist, then the generalized graph Laplacian matrix \( \tilde{\mathcal{L}} \), defined as \( \tilde{\mathcal{L}} = \mathcal{D} - \mathcal{W} + \text{diag}(\mathcal{W}) \), is typically used.

B. Hilbert Space Definitions

We first define a vector space \( S \) of real, symmetric matrices in \( \mathbb{R}^{N \times N} \). We next define an inner product \( \langle \cdot, \cdot \rangle \) for two matrices \( \mathbf{A}, \mathbf{B} \in S \) as

\[
\langle \mathbf{A}, \mathbf{B} \rangle = \text{tr}(\mathbf{B}^\top \mathbf{A}) = \sum_{i,j} A_{ij} B_{ij}.
\]

Assuming Cauchy sequence convergence, the vector space endowed with an inner product is a Hilbert space \( \mathcal{H} \) [24]. We focus on a subspace \( \mathcal{H}^+ \subset \mathcal{H} \) that contains PSD matrices, i.e., \( \mathcal{H}^+ = \{ \mathbf{A} \in \mathcal{H} | \mathbf{A} \succeq 0 \} \). It can be easily proven that \( \mathcal{H}^+ \) is a convex cone [25].

C. Graph Convolutional Network

For a given graph \( \mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{W}) \), a GCN [1] associated with \( \mathcal{G} \) is defined as follows. Denote by \( \tilde{\mathcal{W}} = \mathcal{W} + \mathcal{I}_N \) and \( \tilde{\mathcal{D}} = \mathcal{D} + \mathcal{I}_N \) the adjacency and degree matrices augmented with self-loops, respectively. The augmented normalized Laplacian [26] is defined by \( \tilde{\mathcal{L}} = \tilde{\mathcal{D}}^{-1/2} \tilde{\mathcal{W}} \tilde{\mathcal{D}}^{-1/2} \) and we set \( \mathbf{P} = \mathcal{I}_N - \tilde{\mathcal{L}} \). Let \( L, C \in \mathbb{N}_+ \) be the layer and channel sizes, respectively. With weights \( \Theta^{(l)} \in \mathbb{R}^{C \times C}, l \in \{1, \ldots, L\} \), the GCN is defined by \( f_l = f_{l-1} \circ \cdots \circ f_1 \) where \( f_l : \mathbb{R}^{N \times C} \rightarrow \mathbb{R}^{N \times C} \) is defined by \( f_l(\mathbf{X}) = \text{ReLU}(\mathbf{PX}\Theta^{(l)}) \).

D. Convergence to Invariant Subspace

Several studies [4], [5] have reported that the expressive power of GCN does not grow with layer number; node representations become indistinguishable (known as over-smoothing) when many layers are stacked in the GCN, leading to performance degradation. To understand this undesirable phenomenon, [7] provided theoretical conditions under which the output of GCN exponentially converges to the invariant subspace, which is the eigenspace corresponding to the lowest frequency of the graph Laplacian matrix and has "no information" other than connected components and node degrees.

Given definitions in Sec. II-C, let \( \lambda_1 \leq \cdots \leq \lambda_N \) be the eigenvalues of \( \mathbf{P} \). Suppose \( \mathcal{G} \) has \( M \) connected components, define \( \lambda = \max_{n=1, \ldots, N-M} |\lambda_n| < 1 \) and \( \lambda_{N-M+1} = \cdots = \lambda_N = 1 \). With initial value \( \mathbf{X}^0 \), the distance between the output of \( l \)-th layer \( \mathbf{X}^{(l)} \) and the invariant subspace \( \mathcal{M} \) satisfies

\[
d_M(\mathbf{X}^{(l)}) \leq (s\lambda)^l d_M(\mathbf{X}^{(0)}),
\]

where \( s = \sup_{t=1, \ldots, L} s_t \) and \( s_t \) is the maximum singular value of \( \Theta^{(t)} \). In particular, \( d_M(\mathbf{X}^{(0)}) \) exponentially converges to 0 if \( s\lambda < 1 \). Thus, the gap between \( \lambda \) and \( \lambda_N \) should be minimized to slow down convergence to the invariant subspace and avoid oversmoothing.

III. SPECTRAL GRAPH LEARNING

We propose a spectral graph learning scheme to enhance GCN’s expressiveness and optimize its performance. Given an empirical covariance matrix \( \mathbf{C} \) estimated from data, we compute matrix \( \mathbf{C} \) that has a desired eigen-gap and a pre-specified last eigenvector \( \mathbf{u} \). \( \mathbf{u} \) is also the first eigenvector of to-be-constructed graph Laplacian \( \mathbf{L} = \mathbf{C}^{-1} \), and is set to the mean signal of the training data—the most likely signal assuming a Gaussian Markov random field (GMRF) generation model [27]. We first describe a projection operator \( \text{Proj}(\mathbf{C}) \) projecting \( \mathbf{C} \) onto a convex cone \( \mathcal{H}^+_{\mathbf{C}} \) of real symmetric matrices sharing the same last eigenvector \( \mathbf{u} \) with a desired eigen-gap \( \kappa \). We then use \( \text{Proj}(\cdot) \) in a modified GLASSO formulation [14] to derive an iterative optimization algorithm.

A. Computing Last Eigen-Pair \((\lambda_N, \mathbf{u})\)

Given an empirical covariance matrix \( \mathbf{C} \), we first project \( \mathbf{C} \) onto the rank-1 eigen-component \( \mathbf{U} = \mathbf{u}\mathbf{u}^\top \) to maximize the inner product \( \langle \mathbf{C}, \mathbf{U} \rangle \). The resulting residual signal \( \mathbf{E}_{N-1} \) is

\[
\mathbf{E}_{N-1} = \mathbf{C} - \langle \mathbf{C}, \mathbf{U} \rangle \mathbf{U}.
\]

It can be easily proven that \( \langle \mathbf{C}, \mathbf{U} \rangle \geq 0 \) [25]. Thus, the last eigen-pair of \( \mathbf{C} \) is \( (\lambda_N, \mathbf{v}_N) = (\langle \mathbf{C}, \mathbf{U} \rangle, \mathbf{U}) \).

B. Computing Next Eigen-Pair \((\lambda_{N-1}, \mathbf{v}_{N-1})\)

Next, we compute the \((N-1)\)-th eigen-pair \((\lambda_{N-1}, \mathbf{v}_{N-1})\). We seek a vector \( \mathbf{v}_{N-1} \) that maximizes the following inner product:

\[
v_{N-1} = \arg \max_{\mathbf{v}} \langle \mathbf{E}_{N-1}, \mathbf{vv}^\top \rangle, \quad \text{s.t.} \quad \begin{cases} \mathbf{u}^\top \mathbf{v} = 0 \\ \|\mathbf{v}\|_2 = 1 \end{cases}.
\]

The constraints require eigenvectors of a real symmetric matrix \( \mathbf{C} \) to be orthonormal. Objective in (4) is equivalent to \( \text{tr}(\mathbf{v}^\top \mathbf{E}_{N-1} \mathbf{v}) \), which is quadratic and convex, given \( \mathbf{E}_{N-1} \) can be proven to be PSD [25]. Thus, the maximization (4) is non-convex and NP-hard.

1) SDP Relaxation: Instead of solving (4) directly, we relax it via semi-definite programming relaxation (SDR) [23] as follows. Define first \( \mathbf{V} = \mathbf{vv}^\top \). Adding constraint \( \text{tr}(\mathbf{V}) = 1 \) would imply \( \|\mathbf{v}\|_2 = 1 \). Rank-1 constraint \( \mathbf{V} = \mathbf{vv}^\top \) implies \( \mathbf{V} \succeq 0 \) but not vice versa. Thus, by relaxing \( \mathbf{V} = \mathbf{vv}^\top \) to the less stringent \( \mathbf{V} \succeq 0 \), we get the following SDR formulation:

\[
\max_{\mathbf{V}} \langle \mathbf{E}_{N-1}, \mathbf{V} \rangle, \quad \text{s.t.} \quad \begin{cases} \langle \mathbf{U}, \mathbf{V} \rangle = 0 \\ \text{tr}(\mathbf{V}) = 1 \\ \mathbf{V} \succeq 0 \end{cases}.
\]
(5) has a linear objective and a set of linear constraints plus a PSD cone constraint, and thus is an semi-definite programming (SDP) problem. SDR is commonly used to approximate quadratically constrained quadratic programs (QCQP) [28], where (4) is a special case. SDP can be solved in polynomial time using off-the-shelf solvers [23]. Given computed solution \( V \) to (5), we compute its rank-1 approximation \( vv^\top \), where \( v \) is the last eigenvector of \( V \) corresponding to the largest eigenvalue. \( v \) is then the next eigenvector \( v_{N-1} \) for \( C \).

2) Fast Approximation: Because matrix variable \( V \) in (5) has size \( N \times N \), for large graphs this is still expensive. Thus, we perform a fast approximation when \( N \) is large. We first approximate \( E_{N-1} \) with its rank-1 approximation \( ee^\top \), where \( e \) is the last eigenvector of \( E_{N-1} \). We then formulate the following problem:

\[
\max_v e^\top v, \quad \text{s.t. } \begin{cases} 
  u^\top v = 0 \\
  \|v\|_2^2 \leq 1 
\end{cases}.
\]

(6)

The first constraint is the same as the first constraint is (4). The second constraint is a relaxation of \( \|v\|_2 = 1 \) in (4), so that the feasible space is a convex set.

Then, we rewrite the constrained problem (6) into the corresponding unconstrained version:

\[
\min_v -e^\top v + \gamma v^\top uu^\top v + \Phi_1(v)
\]

(7)

where \( \gamma > 0 \) is a penalty weight parameter, and \( \Phi_1(v) \) is a convex function in \( v \) defined as

\[
\Phi_1(v) = \begin{cases} 
  0 & \text{if } \|v\|_2^2 \leq 1 \\
  \infty & \text{o.w.}
\end{cases}.
\]

(8)

Denote by \( \Theta(v) = -e^\top v + \gamma v^\top uu^\top v \). Thus, the objective in (7) is composed of two functions: i) \( \Theta(v) \) is convex and differentiable w.r.t. \( v \) with gradient \( \nabla \Theta(v) = -e + 2\gamma uu^\top v \), and ii) \( \Phi_1(v) \) is convex and non-differentiable. One can thus optimize (7) using proximal gradient (PG) [30]. For initialization, we set the first solution to be \( v^0 = e/\|e\|_2 \).

3) Compute Eigenvalue \( \lambda_{N-1} \): Given \( v_{N-1} \), we compute the corresponding eigenvalue \( \lambda_{N-1} \) as

\[
\lambda_{N-1} = \max \left( \lambda_N - \kappa, (E_{N-1}, v_{N-1}v_{N-1}^\top) \right)
\]

(9)

where \( \kappa > 0 \) is a chosen parameter so that there is a desired eigen-gap between the last two eigenvalues \( \lambda_N \) and \( \lambda_{N-1} \). Residual signal is now \( E_{N-2} = E_{N-1} - \lambda_{N-1}v_{N-1}v_{N-1}^\top \). 

C. Computing Remaining Eigen-Pairs \( (\lambda_i, v_i) \): Computing remaining eigen-pairs \( (\lambda_i, v_i) \) for \( i \in \{N - 2, \ldots, 1\} \) is a straightforward extension. Given residual signal \( E_i = E_{i+1} - \lambda_{i+1}v_{i+1}v_{i+1}^\top \), we first compute its rank-1 approximation \( E_i \approx ee^\top \), where \( e \) is the last eigenvector of \( E_i \), again computed in linear time using LOBPCG. We formulate a similar optimization to (7) as

\[
\min_v -e^\top v + \gamma v^\top YY^\top v + \Phi_1(v)
\]

(10)

where \( Y \in \mathbb{R}^{N \times (N-1)} \) is composed of last eigenvector \( u \) and previously computed eigenvectors \( \{v_j\}_{j=i+1}^{N-1} \) as columns, i.e.,

\[
Y = \left[ v_{N-1} \ldots v_{i+1} \right].
\]

(11)

Optimal solution \( v_i \) to (10) can be computed using PG again. The corresponding eigenvalue \( \lambda_i \) is

\[
\lambda_i = \min \left( \lambda_{i+1}, (E_i, v_iv_i^\top) \right)
\]

(12)

(12) ensures that the computed eigenvalues satisfy \( \lambda_i \leq \lambda_{i+1} \).

D. GLASSO Formulation

The previous eigen-component computation constitutes a projection \( C = \text{Proj}(\tilde{C}) \)—one can prove this operator is indeed idempotent, i.e., \( \text{Proj}(\text{Proj}(C)) = \text{Proj}(C) \) [25]. Next, we formulate the following GLASSO-like optimization problem to estimate a graph Laplacian matrix \( L \) [31]:

\[
\min_{L^{-1} \in H_{++}^N} \text{Tr}(L\tilde{C}) - \log \det L + \rho \|L\|_1
\]

(13)

where \( \rho > 0 \) is a shrinkage parameter for the \( l_1 \)-norm. The only difference from GLASSO is that (13) has an additional constraint \( L^{-1} \in H_{++}^N \). \( H_{++}^N \) is a set of matrices with the last eigenvector being \( u \) and the desirable eigen-gap \( \kappa \).

We solve (13) iteratively using our projection operator \( \text{Proj}(\cdot) \) and a variant of the block Coordinate descent (BCD) algorithm in [32]. Specifically, similarly done in [22], we solve the dual of GLASSO as follows. Note first that the \( l_1 \)-norm in (13) can be written as

\[
\|L\|_1 = \max_{U \leq 1} \text{Tr}(LU)
\]

(14)

where \( \|U\|_\infty \) is the maximum absolute value element of the symmetric matrix \( U \). Hence, the dual problem of GLASSO that seeks an estimated covariance matrix \( C = L^{-1} \) is

\[
\min_{C \in H_{++}^N} -\log \det C, \quad \text{s.t. } \|C - \tilde{C}\|_{\infty} \leq \rho
\]

(15)

where \( C = \tilde{C} + U \) implies that the primal and dual variables are related via \( L = (C + U)^{-1} \) [15]. To solve (15), we update one row-column pair of \( C \) in (15) in each iteration following optimization procedure in [15].

Our algorithm to solve (13) is thus as follows. We minimize the GLASSO terms in (13) by solving its dual (15) — iteratively updating one row / column of \( C \). We then project \( C \) to \( H_{++}^N \) using our projection operator. We repeat these two steps till convergence. Note that both steps are computed using covariance \( C \) directly, and thus inversion to graph Laplacian \( L = C^{-1} \) is not necessary until convergence, when we output a solution.

IV. GRAPH CONVOLUTIONAL NETWORKS TRAINING

We employ the learned generalized graph Laplacian \( L \) as an underlying topology to train GCNs [1]. The graph convolution at the \( l \)-th layer outputs the following:

\[
X_l = \text{ReLU}(PX_l^{(l)}G^{(l)}),
\]

(16)
where $\mathbf{P}$ is the normalized and self-loop-augmented adjacency matrix, $\mathbf{X}^{(l)}$ and $\mathbf{\Theta}^{(l)}$ are the input feature and network parameters at the $l$-th layer.

To compute $\mathbf{P}$, we need to normalize the given generalized graph Laplacian $\mathbf{L}$ learned in Section III, so that the convergence theorem in [7] is applicable. However, the normalization adopted in GCN [11] is not applicable since it changes the order of engineered eigen-gaps. Instead, we spectrally normalize $\mathbf{L}$ via

$$\mathbf{P} = \mathbf{I}_N - \mathbf{L}, \quad \mathbf{L} = \frac{2}{\mu_{\max}}(\mathbf{L} - 1/\lambda_N \mathbf{I}_N), \quad (17)$$

where $1/\lambda_N$ is the smallest eigenvalue of $\mathbf{L}$, so that the smallest eigenvalue of $\mathbf{P}$ equals to 0 and the largest eigenvalue of $\mathbf{P}$ equals to 1, then $d_{\mathcal{M}}(\mathbf{X})$ does not vanish after graph convolution. $\mu_{\max}$ is the maximum eigenvalue for all $\mathbf{L}$ with different eigen-gaps, so that the eigenvalues of $\mathbf{P}$ is normalized into range $[0,2]$, and the order of the eigen-gap is preserved.

V. EXPERIMENTS

We conducted experiments using the METR-LA [33] dataset to validate our proposed scheme that relieves over-smoothing and increases the optimal GCN layer number with smaller eigen-gaps. Further, by comparing against recent proposal DropEdge [8], our proposed scheme achieved larger optimal layer numbers and lower prediction errors.

A. Dataset and Evaluation Metric

**Dataset.** The METR-LA dataset [33] used in the experiments contains traffic speed data in four months (from March 1st 2012 to June 30th 2012) from 207 sensors in the Los Angeles County. The sensors sampled the speed data every 5 minutes. Our task is to predict the current traffic speed using historical speed data from 50 minutes ago to 5 minutes ago as the input feature. In our experiments, we randomly sampled 70% data as training dataset, 20% as validation dataset and 10% as test dataset.

**Evaluation Metric.** We use Mean Squared Error (MSE) as our evaluation metric defined as:

$$\text{MSE}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2 \quad (18)$$

where $\mathbf{x} = [x_1, \ldots, x_N]$ are the ground truth values, and $\hat{\mathbf{x}} = [\hat{x}_1, \ldots, \hat{x}_N]$ are the predicted values.

B. Experimental Settings

For graph learning, we computed the empirical covariance matrix $\mathbf{C}$ with all the observations in the training data, and used the mean signal in the training data as the first eigenvector of Laplacian $\mathbf{L}$ (last eigenvector $\mathbf{u}$ of computed covariance $\mathbf{C}$). Sparsity parameter $\rho$ in (13) was set to $10^{-4}$. The GCN training was implemented in PyTorch and was optimized by Adam [34]. Our GCN model was consisted of $L$ GCN blocks ($L = [1,9]$) and two linear layers. Each GCN block contained a graph convolutional layer, a BatchNorm layer and an activation layer (leaky ReLU).

C. Comparison among Different Eigen-gaps

For graph learning, we set eigen-gap as $\kappa \in \{1.0, 3.0, 5.0, 7.0, 8.0\}$ and compared against the solution produced without eigen-gap modification, which had eigen-gap 8.5083. Fig. 1 shows the results for GCN training using graph Laplacian matrices learned with eigen-gap $\kappa$ and the original eigen-gap. Results with 1 layer were omitted since the values were too large. As shown in Fig. 1, smaller eigen-gap relieved over-smoothing during the GCN training and achieved larger optimal number of layers. For example, for eigen-gap 1.0, the optimal result was achieved with layer 6, while for larger eigen-gaps, the optimal results were reached at layer 1. The MSE results are also shown in Table I, where the optimal results are highlighted in bold font.

D. Comparison with State-of-the-art Methods

We compared our method with recent DropEdge [8]. Same experimental settings were used for DropEdge with drop rate $p = \{0,0.3,0.5\}$. The MSE results are shown in Fig. 2. We observe that our method had better performance in terms of slowing down over-smoothing and achieving lower prediction error. Specifically, DropEdge achieved the optimal results at the second layer with MSE 0.0107, while our method increased the optimal layer number to 6 with gap 1.0, and achieved lower MSE 0.0094.

| TABLE I | MSE RESULTS WITH DIFFERENT LAYER SIZE OF GCN MODELS USING LAPLACIAN MATRICES LEARNED WITH DIFFERENT Eigen-gaps. |
|---|---|---|---|---|---|---|---|---|
| layers | gap = 1 | gap = 3 | gap = 5 | gap = 7 | gap = 8 | gap = 8.5083 |
| 1 | 0.00953 | 0.00948 | 0.00954 | 0.00946 | 0.0096 | 0.00946 |
| 2 | 0.00949 | 0.00955 | 0.00961 | 0.00968 | 0.00981 | 0.01046 |
| 3 | 0.00948 | 0.0096 | 0.00992 | 0.00983 | 0.01032 | 0.01156 |
| 4 | 0.0095 | 0.00982 | 0.00956 | 0.00981 | 0.01086 | 0.01205 |
| 5 | 0.00954 | 0.00976 | 0.00991 | 0.00993 | 0.01156 | 0.01247 |
| 6 | 0.00994 | 0.00965 | 0.00988 | 0.01054 | 0.01142 | 0.01268 |
| 7 | 0.0097 | 0.00993 | 0.00983 | 0.01079 | 0.01209 | 0.01317 |
| 8 | 0.00944 | 0.00967 | 0.01054 | 0.01084 | 0.01177 | 0.0125 |
| 9 | 0.00969 | 0.01001 | 0.01012 | 0.0116 | 0.01347 | 0.01322 |

Fig. 1. MSE results with different layer sizes of GCN models using Laplacian matrices learned with different eigen-gaps.

\[ \text{MSE}(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{x}_i)^2, \quad (18) \]
VI. CONCLUSION

To improve the expressiveness of a graph convolutional network (GCN), we propose to learn a graph Laplacian matrix from data in the spectral domain, such that a target eigen-gap (the difference between the Laplacian’s two distinct dominant eigenvalues) can be directly engineered. Specifically, we compute each eigen-pair greedily via a fast approximation of a semi-definite programming (SDP) formulation. The eigen-gap slows down the inevitable convergence to the subspace spanned by the first eigenvector of the Laplacian, and leads to more optimal depth layers (more expressiveness) and lower loss function values during supervised GCN training. Compared to a recent work [8] that drops edges randomly to improve graph sparsity, our proposal produced deeper GCNs.

REFERENCES

[1] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” Proc. Int. Conf. Learn. Rep., April 2017.
[2] F. Hu, Y. Zhu, S. Wu, L. Wang, and T. Tan, “Hierarchical graph convolutional networks for semi-supervised node classification,” in Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence. (IJCAI), 2019.
[3] T. H. Do, D. M. Nguyen, and N. Deligiannis, “Graph auto-encoder for graph signal denoising,” in ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2020, pp. 3322–3326.
[4] Q. Li, Z. Han, and X.-M. Wu, “Deeper insights into graph convolutional networks for semi-supervised learning,” in Thirty-Second AAAI conference on artificial intelligence, 2018.
[5] G. Li, M. Muller, A. Thabet, and B. Ghanem, “Deepgcns: Can gcn go as deep as cnns?” in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 9267–9276.
[6] T. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in International Conference on Learning Representations, Toulon, France, April 2017.
[7] K. Oono and T. Suzuki, “Graph neural networks exponentially lose expressive power for node classification,” in International Conference on Learning Representations, 2020.
[8] Y. Rong, W. Huang, T. Xu, and J. Huang, “Dropedge: Towards deep graph convolutional networks on node classification,” in International Conference on Learning Representations, 2020.
[9] C. Meyer, Matrix Analysis and Applied Linear Algebra. SIAM, 2010.
[10] G. Golub and C. F. V. Loan, Matrix Computations (Johns Hopkins Studies in the Mathematical Sciences). Johns Hopkins University Press, 2012.
[11] A. Ortega, P. Frossard, J. Kovacevic, J. M. F. Moura, and P. Vandergheynst, “Graph signal processing: Overview, challenges, and applications,” in Proceedings of the IEEE, vol. 106, no.5, May 2018, pp. 808–828.
[12] G. Cheung, E. Magli, Y. Tanaka, and M. Ng, “Graph spectral image processing,” in Proceedings of the IEEE, vol. 106, no.5, May 2018, pp. 907–930.
[13] F. Chung, Spectral Graph Theory. American Mathematical Society, 1996.
[14] J. Friedman, T. Hastie, and R. Tibshirani, “Sparse inverse covariance estimation with the graphical lasso,” Biostatistics (Oxford, England), vol. 9, pp. 432–41, 08 2008.
[15] O. Banerjee and L. Ghaoui, “Model selection through sparsity max likelihood estimation,” Journal of Machine Learning Research, vol. 9, pp. 489–514, 2008.
[16] H. Eglmez, E. Pavez, and A. Ortega, “Graph learning from data under Laplacian and structural constraints,” in IEEE Journal of Selected Topics in Signal Processing, vol. 11, no.6, July 2017, pp. 825–841.
[17] W. Hu, G. Cheung, A. Ortega, and O. Au, “Multi-resolution graph Fourier transforms for compression of piecewise smooth images,” in IEEE Transactions on Image Processing, vol. 24, no.1, January 2015, pp. 419–433.
[18] W. Hu, G. Cheung, and A. Ortega, “Intra-prediction and generalized graph Fourier transform for image coding,” in IEEE Signal Processing Letters, vol. 22, no.11, November 2015, pp. 1913–1917.
[19] J. Pang and G. Cheung, “Graph Laplacian regularization for inverse imaging: Analysis in the continuous domain,” in IEEE Transactions on Image Processing, vol. 26, no.4, April 2017, pp. 1770–1785.
[20] C. Dinesh, G. Cheung, and I. V. Bajic, “Point cloud denoising via feature graph laplacian regularization,” IEEE Transactions on Image Processing, vol. 29, pp. 4143–4158, 2020.
[21] F. Chen, G. Cheung, and X. Zhang, “Fast amp: robust image interpolation using gradient graph laplacian regularizer,” in 2021 IEEE International Conference on Image Processing (ICIP), 2021, pp. 1964–1968.
[22] S. Bagheri, G. Cheung, A. Ortega, and F. Wang, “Learning sparse graph Laplacian with K eigenvector prior via iterative gasso and projection,” in ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021, pp. 5365–5369.
[23] B. Gartner and J. Matousek, Approximation Algorithms and Semidefinite Programming. Springer, 2012.
[24] M. Vetterli, J. Kovacevic, and V. Goyal, Foundations of Signal Processing. Cambridge University Press, 2014. [Online]. Available: https://books.google.ca/books?id=5WAtsnQAACAJ
[25] S. Bagheri, “Learning sparse graph Laplacian with K eigenvector prior via iterative gasso and projection,” in Workshop Library, 2021. [Online]. Available: http://hdl.handle.net/10315/38692
[26] F. Wu, A. Souza, T. Zhang, C. Fiby, T. Yu, and K. Weinberger, “Simplifying graph convolutional networks,” in Proceedings of the 36th International Conference on Machine Learning, PMLR, 2019, pp. 6861–6871.
[27] H. Rue and L. Held, Gaussian Markov random fields: theory and applications. CRC Press, 2005.
[28] Z.-q. Luo, W.-k. Ma, A. M.-c. So, Y. Ye, and S. Zhang, “Semidefinite relaxation of quadratic optimization problems,” IEEE Signal Processing Magazine, vol. 27, no. 3, pp. 20–34, 2010.
[29] A. Knyazev, “Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method,” SIAM journal on scientific computing, vol. 23, no. 2, pp. 517–541, 2001.
[30] N. Parikh and S. Boyd, “Proximal algorithms,” in Foundations and Trends in Optimization, vol. 1, no.3, 2013, pp. 123–231.
[31] R. Mazumder and T. Hastie, “The graphical lasso: New insights and applications,” Electron. J. Statist., vol. 6, pp. 2125–2149, 2012. [Online]. Available: https://doi.org/10.1214/12-EJS740
[32] S. Wright, “Coordinate descent algorithms,” Math. Program., vol. 151, no.1, pp. 3–34, 2015. [Online]. Available: https://doi.org/10.1007/s10107-015-0892-3
[33] Y. Li, R. Yu, C. Shahabi, and Y. Liu, “Diffusion convolutional recurrent neural network: Data-driven traffic forecasting,” in International Conference on Learning Representations (ICLR ’18), 2018.
[34] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in International Conference on Learning Representations, 2015.