Perturbing Eigenvalues with Residual Learning in Graph Convolutional Neural Networks

Shibo Yao EPSOYAO@GMAIL.COM
Dantong Yu DTYU@NJIT.EDU
Martin Tuchman School of Management, New Jersey Institute of Technology, Newark, NJ 07102
Xiangmin Jiao XIANGMIN.JIAO@STONYBROOK.EDU
Department of Applied Mathematics & Statistics, Stony Brook University, Stony Brook, NY 11794

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

Network structured data is ubiquitous in natural and social science applications. Graph Convolutional Neural Network (GCN) has attracted significant attention recently due to its success in representing, modeling, and predicting large-scale network data. Various types of graph convolutional filters were proposed to process graph signals to boost the performance of graph-based semi-supervised learning. This paper introduces a novel spectral learning technique called *EigLearn*, which uses residual learning to perturb the eigenvalues of the graph filter matrix to optimize its capability. EigLearn is relatively easy to implement, yet thorough experimental studies reveal that it is more effective and efficient than prior works on the specific issue, such as LanczosNet and FisherGCN. EigLearn only perturbs a small number of eigenvalues and does not require a complete eigendecomposition. Our investigation shows that EigLearn reaches the maximal performance improvement by perturbing about 30 to 40 eigenvalues, and the EigLearn-based GCN has comparable efficiency as the standard GCN. Furthermore, EigLearn bears a clear explanation in the spectral domain of the graph filter and shows aggregation effects in performance improvement when coupled with different graph filters. Hence, we anticipate that EigLearn may serve as a useful neural unit in various graph-involved neural net architectures.

Keywords: Graph convolution; Eigenvalue perturbation; Spectral learning; Semi-supervised learning

1. Introduction

We address the problem of using Graph Convolutional Neural Network (GCN) for semi-supervised classification on a graph that describes pairwise relationship among the samples (or nodes), assuming that we have the full feature set for both training and testing samples and the label set for training samples. The primary goal of this work is to improve the performance of GCN by learning a more effective graph convolution filter matrix.

Convolutional Neural Network architecture on graph-structured data dates back to (Bruna et al., 2013) and (Henaff et al., 2015), where given a graph $\mathcal{G}$, the convolution operation is defined based on the graph adjacency matrix A or the graph Laplacian $L = D - A$, where D is the degree matrix. Specifically, the low-frequency eigenvectors of the graph Laplacian, along with a set of learnable parameters, jointly define a graph operator in filtering the input signal. The essence of such an operation is to transform the input signal

into the spectral domain (Sandryhaila and Moura, 2013), manipulate the coefficients on the low-frequency band, and then transform the signal back, i.e.

$$\tilde{x} = V_k \Theta_k V_k^T x = \sum_{i=1}^{k} \theta_i v_i v_i^T x, \tag{1}$$

where V_k is composed of the first k low-frequency eigenvectors $\{v_i \mid i = 1, 2, \ldots, k\}$ of the graph Laplacian, and $\Theta_k = \mathrm{diag}(\theta_1, \theta_2, ..., \theta_k)$ is composed of the learnable parameters of the graph filter. A disadvantage of a low-rank filter as in (1) is that it throws away the high-frequency signals and results in information loss.

As an alternative to computing the eigenvectors, Defferrard et al. (2016) proposed to use a truncated expansion of Chebyshev polynomials to construct a graph filter in graph convolution. Kipf and Welling (2016) further simplified the filter using a first-order approximation. Generally speaking, these graph filter matrices can be viewed as graph shift operators (Gavili and Zhang, 2017) as studied in (Dasoulas et al., 2021). Our proposed approach improves the performance of the filters, such as that in (Kipf and Welling, 2016), by learning some perturbations in the spectral domain. Another recent work (Franceschi et al., 2019) proposed a generative approach with the reparameterization trick (Kingma and Welling, 2013) to learn the graph, where each edge is modeled as an independent Bernoulli distribution. The drawback of such an approach is its quadratic complexity and the difficulty in spectral interpretation.

The recently developed LanczosNet (Liao et al., 2019) is one of the closely relevant approaches to ours. It replaces the eigenvectors in (1) using the Ritz vectors from Lanczos (or Arnoldi) iterations, and then learns the parameters for an ensemble of the Ritz vectors and the polynomials of the graph Laplacian. The use of Lanczos iterations reduces the cost of computing the eigenvalues and eigenvectors. However, LanczosNet appears to utilize the Ritz vectors indiscriminately by treating those corresponding to the largest and smallest eigenvalues with equal importance. In this work, we leverage more advanced variants of Lanczos iterations with implicit restart (Lehoucq et al., 1998) and utilize only the Ritz vectors corresponding to the dominant eigenvalues.

Another related work to our approach is the FisherGCN (Sun et al., 2020), which uses a minimax formulation based on the Fisher information metric (FIM) to learn a spectral perturbation. Motivated by the adversarial attack model based on FIM in (Zhao et al., 2019), the minimax formulation in FisherGCN tries to mitigate the worst-case scenario to improve adversarial robustness while improving generalization. However, as pointed out in (Tsipras et al., 2018), there may exist an inherent tension between the goal of adversarial robustness and that of standard generalization. Hence, FisherGCN may also suffer from such a trade-off and in turn yield a suboptimal performance, as evident in our experimental study. In addition, their minimax formulation leads to significantly higher training costs with FisherGCN compared to the regular GCN (Kipf and Welling, 2016). In contrast, we formulate the objective as a pure minimization problem to improve the performance and generalization, with minimal overhead compared to the regular GCN (Kipf and Welling, 2016). Although this work does not consider adversarial robustness, our proposed approach demonstrates superior robustness in terms of reducing the standard deviation of testing accuracy.

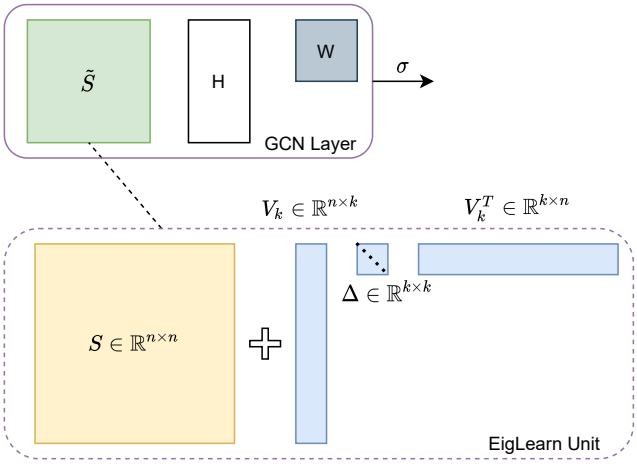

Figure 1: Schematic of one-layer of GCN with EigLearn. H denotes the feature matrix, W is the linear mapping matrix and $\tilde{S} = S + V_k \Delta V_k^T$ is the perturbed graph filter matrix with S being the original filter matrix. In the perturbation unit, V_k is composed of k significant eigenvectors of S. The learnable parameters in Δ represents the eigenvalue perturbation and is realized as a residual unit in the neural network.

At a high level, the core idea of our proposed approach, EigLearn, is to overcome the shortcomings of the aforementioned approaches using a novel combination of some variants of spectral modification (Koutis and Le, 2019) and residual learning (He et al., 2016; Srivastava et al., 2015). As illustrated in Fig. 1, EigLearn learns some perturbations to the low-frequency signals while preserving high-frequency signals by including a residual unit. Overall, the contributions of this work are as follows:

- We introduce a new formulation of eigenvalue perturbation as a residual learning problem, called EigLearn,[1] which hybridizes the ideas of optimal low-rank approximations and residual learning.
- EigLearn can be easily applied to existing graph filter matrices to improve their performances with moderate computational overhead.
- EigLearn is robust in terms of hyperparameters, and empirical results show that a set of default parameters yields near-optimal improvements.

The remainder of the paper is organized as follows. Section 2 briefly reviews semi-supervised learning with GCN and lays out the theoretical foundation of our approach. Section 3 describes the logic and details of our proposed method, EigLearn. Section 4 compares EigLearn with LanczosNet and FisherGCN and presents the empirical study of the sensitivity of its hyperparameters. Finally, we conclude the paper with some discussions on future works.

1. https://github.com/ShiboYao/EigLearn

2. Preliminaries

2.1. Semi-supervised Classification with GCN

In semi-supervised learning, let $\{x_u\}$ and $\{x_l\}$ denote the sets of feature vectors of unlabeled and labeled samples, respectively, and $\{y_l\}$ denote the set of labels of training samples typically amongst $\{x_l\}$. The objective is to infer the labels $\{y_u\}$ for the testing samples. Graph-based learning involves a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that describes the pairwise relations of the samples (i.e., the nodes in $\mathcal{V}$). GCN belongs to the family of graph-based learning, which takes advantage of both $\mathcal{G}$ and the rich capacity of the neural network to improve model performance. A typical GCN layer is formulated recursively as follows:

$$H^{l+1} = \sigma(SH^lW) \tag{2}$$

where $H^l \in \mathbb{R}^{n\times d}$ is the feature-map matrix for all the nodes in the lth layer, $S \in \mathbb{R}^{n\times n}$ is a graph filter matrix, $W \in \mathbb{R}^{d\times d'}$ corresponds to a weight matrix of a single (dense) layer perceptron, and σ is an activation function for nonlinearity. The classification result is usually obtained with a softmax function in the final output layer.

Typically, the graph filter matrix S is obtained from the adjacency matrix A or the graph Laplacian L. A commonly used filter is the symmetrically normalized adjacency (SNA) matrix with added self-loops (Kipf and Welling, 2016), i.e.

$$S_{\text{SNA}} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}, \tag{3}$$

where $\tilde{A} = A+I$ and $\tilde{D} = \text{diag}(\tilde{A}\mathbf{1})$. Other examples include higher-order polynomials with some normalization (Gavili and Zhang, 2017). Sun et al. (2020) introduced a more sophisticated filter matrix by preprocessing the graph adjacency matrix A based on DeepWalk similarities (Perozzi et al., 2014) and termed the resulting Graph Convolutional Networks GCN^T. We denote the graph filter matrix in (Sun et al., 2020) as S_{DW}. In this work, we use GCN based on S_{SNA} in (Kipf and Welling, 2016) and GCN^T based on S_{DW} in (Sun et al., 2020) as the baselines for perturbation and performance improvements. For conciseness, we will omit the subscripts and use S to denote the graph filter matrix for generality.

A commonality among the existing filters is that they can all be interpreted as some manipulation of the eigenvalues of the given graph adjacency or Laplacian. Taking a higher-order polynomial filter as an example, when the graph is undirected, it is easy to show the polynomial construction of graph filter is equivalent to the polynomial of eigenvalues, in that given $M = V\Lambda V^T$,

$$\sum_{i=0}^{k}\theta_i M^i = \sum_{i=0}^{k}\theta_i (V\Lambda V^T)^i = \sum_{i=0}^{k}\theta_i V\Lambda^i V^T = V\left(\sum_{i=0}^{k}\theta_i\Lambda^i\right)V^T.$$

Another example is the inverse shifted Laplacian (Jiang et al., 2019)

$$\tilde{L}^{-1} = (\tilde{D}^{-\frac{1}{2}}(\tilde{D}-\tilde{A})\tilde{D}^{-\frac{1}{2}})^{-1},$$

where $\tilde{A} = A + \theta I$ and $\tilde{D} = \text{diag}(\tilde{A}\mathbf{1})$. Given $\tilde{L} = V\Lambda V^T$, the inverse matrix is calculated as follows:

$$\tilde{L}^{-1} = (V\Lambda V^T)^{-1} = (V^T)^{-1}\Lambda^{-1}V^{-1} = V\Lambda^{-1}V^T,$$

where $\Lambda_{ii}^{-1} = 1/\lambda_i$. Hence, the inverse shifted Laplacian as a graph filter matrix can also be interpreted as a manipulation of the eigenvalues. This observation motivates us to formulate our approach also as manipulation of the eigenvalues. Instead of working on the full spectrum, we propose to manipulate the eigenvalues of a graph filter matrix selectively to improve its performance with low computational cost.

2.2. Optimal Low-Rank Approximations and Minimal Perturbation

We develop our approach by starting with an approximation of the graph filter matrix S. Our point of departure is the following well-known fact regarding the optimal low-rank approximations.

Lemma 1 *Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$ with an eigendecomposition $A = V\Lambda V^T$, where the eigenvalues in Λ are in descending order in magnitude, i.e., $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$, the matrix $A_k = \sum_{i=1}^{k} \lambda_i v_i v_i^T$ satisfies*

$$A_k = \arg \min_{B\in\mathbb{R}^{n\times n} \wedge rank(B)\leq k} \|A - B\|. \tag{4}$$

The lemma directly follows from the Eckart-Young theorem (Golub and Van Loan, 2013, p. 79). Lemma 1 is more relevant to the classical graph shift operators as in (Bruna et al., 2013) and (Henaff et al., 2015), but it is not directly applicable in our setting, since we perturb the eigenvalues of the filter matrix. The following theorem shows that the optimal low-rank approximation is indeed a good choice for perturbing the filter.

Theorem 2 *Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$, its optimal rank-k approximation A_k in Lemma 1 also minimizes the maximum perturbation in $A - B$ for all $B \in \mathbb{R}^{n\times n}$ of rank k or less and all perturbations $\delta \in \mathbb{R}^{n\times n}$ with $\|\delta\| = c$ for some constant $c > 0$, i.e.,*

$$A_k = \sum_{i=1}^{k} \lambda_i v_i v_i^T = \arg \min_{\substack{B\in\mathbb{R}^{n\times n} \\ rank(B)\leq k}} \max_{\substack{\delta\in\mathbb{R}^{n\times n} \\ \|\delta\|=c}} \|A - B + \delta\|. \tag{5}$$

Proof Given any $B \in \mathbb{R}^{n\times n}$, let $\hat{u}_1 \in \mathbb{R}^n$ and $\hat{v}_1 \in R^n$ denote the left and right singular vectors corresponding to the largest singular value of $A - B$. Due to the Cauchy-Schwartz inequality,

$$\|A - B + \delta\| \leq \|A - B\| + \|\delta\|,$$

where the inequality is an equality when $\delta = c\hat{u}_1\hat{v}_1^T$. Hence,

$$\min_{\substack{B\in\mathbb{R}^{n\times n} \\ \text{rank}(B)\leq k}} \max_{\substack{\delta\in\mathbb{R}^{n\times n} \\ \|\delta\|=c}} \|A - B + \delta\| = \min_{\substack{B\in\mathbb{R}^{n\times n} \\ \text{rank}(B)\leq k}} \|A - B\| + c,$$

which is minimized iff $\|A - B\|$ is minimized, i.e., $B = A_k$ due to Lemma 1. ■

In plain English, Theorem 2 states that the worst-case deviation from the graph filter S is minimized when the filter is constructed from the optimal low-rank approximation. It

is worth noting that Lemma 1 also holds in the Frobenius norm, so does Theorem 2. We omit their proofs.

In practice, however, the graph filter matrix S is typically asymmetric, even in the case of SNA. In general, we could apply the Eckart-Young theorem in place of Lemma 1 and use a truncated singular value decomposition (TSVD) of an asymmetric S to construct the graph filter. However, if S is not too far from symmetry, it is more efficient (in terms of both computational cost and memory requirement) to use the eigenvalue decomposition of $(S+S^T)/2$ to construct a "near-optimal" low-rank approximation. In this work, we use the latter approach for its better efficiency. We found it to be effective for applications such as citation networks.

2.3. Connection of Eigenvalue Perturbation with Residual Learning

One limitation of using a low-rank approximation as described in Section 2.2 is that such low-rank approximations only contain the low-frequency signals, and a filter based on low-rank approximation alone may miss some important information in the high-frequency signals. To overcome this limitation, one can introduce a "residual unit" into the neural network, analogous to the residual learning in HighwayNet (Srivastava et al., 2015) and ResNet (He et al., 2016). The key idea of residual learning is to allow the original signal flow freely in the neural net and focus on learning the signal residual, and therefore generate better sample representations. In this work, we propose a similar yet different idea. Our objective is different from the aforementioned residual learning, in that we design a novel residual unit in the frequency domain, to better learn the low-frequency signals while preserving the original signals in both low-frequency and high-frequency bands, and in turn improve the performance. We accomplish the residual learning in EigLearn by training a constant number of free parameters that serve as the perturbation to the graph filter matrix's low-frequency eigenvalues. One benefit of our residual learning is that it does not require a complete eigendecomposition and incurs no steep computation cost. In other words, the residual unit in EigLearn is composed of a collection of actual neurons that resembles the biases, as we will explain in detail in Section 3.

3. Learning Eigenvalue Perturbation

As discussed in Section 2.1, our approach is motivated by the fact that many GCN models exploit certain types of eigenvalue manipulation on the adjacency matrix A or the Laplacian matrix L and construct an effective graph filter matrix. The examples include but are not limited to, subspace constructed from the graph Laplacian with learnable coefficients (Bruna et al., 2013; Henaff et al., 2015), graph shift operator constructions (Dasoulas et al., 2021; Defferrard et al., 2016) and inverse Laplacian (Jiang et al., 2019). We develop our approach by first raising the following questions:

- Given a graph filter matrix, either hand-crafted or parameterized, can we further perturb it using a data-driven approach and make it more effective in learning graph signals?
- The polynomial graph filter matrices are based on manipulation on all eigenvalues, which can be a waste of computation and a waste of model capacity if the polynomial

is parameterized. Can we selectively perturb a subset of eigenvalues efficiently and effectively?

- Can we do this without increasing the overall complexity of GCN?

These questions also serve as the design principles of our method and guide the investigation throughout the paper.

3.1. Perturbing the Eigenvalues

Without loss of generality, let us assume the graph filter matrix $S \in \mathbb{R}^{n \times n}$ is symmetric, since we have addressed the general treatment of asymmetric matrices in Section 2.2. Let $S = V \Lambda V^T$ denote the eigendecomposition of S. We hypothesize that there is a sizable room for performance improvement even in those well-designed graph filter matrices for GCN, if we introduce some modifications to the graph filter matrix S. One might be tempted to directly treat all entries in S as free parameters, but such an approach would incur at least quadratic complexity and would not reveal insights on the spectral properties.

Instead, we propose to modify the eigenvalues of S and optimize the filter in the spectral domain. Because the modification is often small, conceptually it corresponds to a perturbation process, i.e.

$$\tilde{\Lambda} = \Lambda + \Delta = \text{diag}(\lambda_1 + \delta_1, ..., \lambda_n + \delta_n), \tag{6}$$

where most of the δ_i should be zeros for efficiency. We address the complexity with k significant eigenvectors and residual learning in the following section.

3.2. Analysis of Complexity

One potential challenge of the proposed approach is the complexity of the eigendecomposition in the first place. A straightforward implementation based on eigendecomposition introduces cubical complexity and hinders scalability. Instead, we can improve the GCN performance with the proposed approach with only additional linear complexity w.r.t. the number of nodes and a constant number of learnable parameters. This is done by learning the perturbation on a constant number, k, of significant eigenvalues. Note that most of the graphs in real-world applications are sparse. Hence, extracting a significant eigenvector in the sparse system has linear-time complexity in the number of edges (and equivalently the number of nodes) per Lanczos iteration. Since it typically suffices to use a low-precision approximation to the leading eigenvectors, we expect the number of Lanczos iterations can be limited to a small constant. Hence, solving for k significant eigenvectors in a sparse system is approximately $O(kn)$. Given $\Delta = \text{diag}(\delta_1, ..., \delta_k)$,

$$\tilde{\Lambda} = \Lambda + \Delta = \text{diag}(\lambda_1 + \delta_1, ..., \lambda_k + \delta_k, \lambda_{k+1}, ..., \lambda_n). \tag{7}$$

Note that it is unnecessary to have a full matrix decomposition. The above equation is the same as follows:

$$\tilde{S} = S + V_k \Delta V_k^T \tag{8}$$

where V_k are pre-computed and stored. It constitutes residual learning where S is the original matrix and the perturbation is the residual component. To generalize the approach

to dense matrices S without suffering from quadratic complexity when solving for V_k, one can employ the promising matrix sparsification (Achlioptas and McSherry, 2007) and graph sparsification (Spielman and Teng, 2011; Koutis et al., 2015) methods that preserve the spectral properties within a provable error bound. Although the sparsification on a dense system takes quadratic complexity w.r.t. the number of nodes, it is inexpensive in practice and can be done reasonably fast for very large-scale situations.

One of the most important hyperparameters in EigLearn is k, which should be large enough to introduce sufficient capacity to the model and yet small enough to prevent over-fitting and retain the complexity. Practically, one could apply grid search to identify an optimal k in the same way as finding the optimal number of neurons. In our experimental study, we found that a small k (30–40) achieves optimal performance and the EigLearn is robust within a reasonable range of k (20-150).

When computing V_k, we observe linear complexity w.r.t. $|\mathcal{E}|$, i.e., $O(kr|\mathcal{E}|)$, where k is constant and not proportional to $|\mathcal{V}|$, r is the number of Arnoldi iterations and constant with low precision, and $|\mathcal{E}|$ is the number of edges in the graph. In most applications, the graph is sparse and $|\mathcal{E}|$ is linear to $|\mathcal{V}|$, and therefore computing V_k is $O(kr|\mathcal{V}|)$, i.e. linear in $|\mathcal{V}|$. For dense graphs, the evaluation of $A \cdot X$ or $S \cdot X$ is already quadratic, and computing V_k is also quadratic. Hence, EigLearn does not change the asymptotic complexity of the GCN model in either dense or sparse cases. It's worth mentioning that with a relatively larger error tolerance in the eigen solver, EigLearn still works well.

3.3. Forward-Back Propagation, Gradient Descent and Neural Architecture Setup

Let us first use one GCN layer as an illustration and omit the bias term for simplicity. The forward propagation is:

$$H^{l+1} = \mathrm{ReLU}(\tilde{S}H^l W). \tag{9}$$

Substituting $\tilde{S} = S + V_k \Delta V_k^T$, we have

$$H^{l+1} = \mathrm{ReLU}((S + V_k \Delta V_k^T)H^l W). \tag{10}$$

We follow a two-stage training procedure. Firstly Δ is initialized as 0 and fixed. Stage-I is equivalent to a regular GCN training and

$$H^{l+1} = \mathrm{ReLU}(SH^l W).$$

The parameters in W (and the bias term if there is any) are updated in stage-I. In stage-II, we keep W fixed and update Δ.

Assume we employ empirical risk minimization learning schema given some predefined loss function and let $\mathcal{L}$ denote the empirical loss back-propagated to this GCN layer. The gradient descent on W in stage-I training is

$$W \leftarrow W - \eta \cdot \nabla_W \mathcal{L} \tag{11}$$

and the gradient descent on Δ in stage-II training is

$$\Delta \leftarrow \Delta - \eta \cdot \nabla_\Delta \mathcal{L} \tag{12}$$

where η is the learning rate.

In our experimental study, we use a two-layer GCN for illustration. To further reduce the degree of freedom and prevent potential overfitting, we share Δ (of which the k diagonal entries are trainable) in both layers, i.e.,

$$\tilde{S}^{(1)} = \tilde{S}^{(0)} = \tilde{S} = S + V_k \Delta V_k^T. \tag{13}$$

Hence, the GCN output is

$$\hat{Y} = \text{softmax}(\tilde{S}(\text{ReLU}(\tilde{S}XW^{(0)}))W^{(1)}). \tag{14}$$

4. Empirical Study

We have implemented EigLearn in PyTorch (Paszke et al., 2019). Our implementation presently supports the graph filter matrices S_{SNA} and S_{DW} as mentioned in Section 2.1. We refer to the resulting GCN models as EigLearnGCN and EigLearnGCN^T, respectively. In terms of the eigenvalue solver, we used eigsh in SciPy(Virtanen et al., 2020), a wrapper of ARPACK (Lehoucq et al., 1998). It is worth mentioning that EigLearn only requires moderate accuracy of the computed eigenvalues and eigenvectors. Throughout our empirical study, we used a relative tolerance of $1e{-}8$ for the eigenvalues, leading to a good balance between accuracy and efficiency. To assess the effectiveness of the proposed approach against other competitors, we conducted semi-supervised node classification on three benchmark datasets, namely Cora, CiteSeer and PubMed, and compared our approach with LanczosNet (Liao et al., 2019) and FisherGCN (Sun et al., 2020) based on their corresponding experimental setups.

We used a similar configuration as in (Sun et al., 2020). In particular, we used a two-layer GCN with 64 hidden units. During stage-I training on the regular GCN, we minimized the loss function (i.e., negative log-likelihood) using the ADAM optimizer with 200 epochs, 0.01 learning rate, $2e{-}4$ weight decay, and 0.5 dropout for the sparse feature input. In stage-II training for the perturbations to the dominant eigenvalues, we set the number of dominant eigenvalues to be perturbed to 40. We used a smaller learning rate of 0.002 and a larger weight decay of $4e{-}3$, in order to encourage smaller perturbations and penalize large perturbation.

4.1. Comparison with LanczosNet

We first compare EigLearn with LanczosNet (Liao et al., 2019), since it is probably the most similar to our approach in that it also has a learnable filter based on approximate eigenvectors. The experimental study in (Liao et al., 2019) had a focus on multi-scale molecule regression besides semi-supervised node classification. Since the publicly available implementation of LanczosNet does not include the semi-supervised classification, we adapted the problem setup of EigLearn based on that in (Liao et al., 2019). In particular, we adopted the public fixed split in this comparison. In addition, according to the publicly available implementation, LanczosNet did not employ sparse dropout. For a fair comparison, we implemented EigLearn both with and without sparse dropout.

Table 1 compares EigLearnGCN with and without sparse dropouts with LanczosNet and AdaLanczosNet in (Liao et al., 2019). Since the testing loss was not given in (Liao

Table 1: Performance comparison of EigLearnGCN with and without sparse dropout versus LanczosNet and AdaLanczosNet (without dropout). Leaders are in boldface.

	Testing Accuracy		
model	Cora	CiteSeer	PubMed
LanczosNet	79.5 ± 1.8	66.2 ± 1.9	78.3 ± 0.3
AdaLanczosNet	80.4 ± 1.1	68.7 ± 1.0	78.1 ± 0.4
EigLearnGCN (w/ dropout)	$\mathbf{82.1} \pm 0.2$	70.3 ± 0.8	79.2 ± 0.5
EigLearnGCN (w/o dropout)	81.8 ± 0.3	$\mathbf{70.7} \pm 0.7$	$\mathbf{79.3} \pm 0.2$

et al., 2019), we only report testing accuracy in Table 1. EigLearnGCN consistently delivered better performance than LanczosNet regardless of whether sparse dropout is utilized. It is worth noting that the results in (Liao et al., 2019) were obtained using the fine-tuned hyperparameters for different cases, but EigLearnGCN used the default parameters for all the cases. These results show that EigLearn is not only more accurate but also more robust than LanczosNet in terms of parameter tuning. The superiority of EigLearn is probably because EigLearn only perturbs the dominant eigenvalues to construct minimal perturbations based on Theorem 2, whereas LanczosNet perturbs all the approximate extreme eigenvalues (i.e., the Ritz values). The additional parameters associated with the smallest eigenvalues in LanczosNet are unlikely to improve accuracy, and their presence may cause LanczosNet to be more prone to overfitting and hence lower performance. We also noticed that the baseline performance of GCN in (Liao et al., 2019) was worse than that in (Kipf and Welling, 2016), but no explanation was provided in (Liao et al., 2019).

4.2. Comparison with FisherGCN

The second assessment focuses on the comparison between EigLearn and FisherGCN (Sun et al., 2020) that is arguably more sophisticated than LanczosNet. In this comparison, we adopted the same settings for data splitting and training as in (Sun et al., 2020). In particular, we split the data into a training set with 20 samples per class, a validation set with 500 samples, and a testing set with 1000 samples. We ran experiments with 20 random splits and for each data split there were 10 different initializations per split. As thoroughly studied in (Shchur et al., 2018), a random split is a less biased evaluation setting for GCN performance, and hence is preferable over the public fixed split. The other parameters remained the same in EigLearn as in Section 4.1, which is consistent with those in (Sun et al., 2020).

Table 2 reports a comparison among the baseline GCN, FisherGCN, and EigLearn using the graph filter matrices based on S_{SNA} and S_{DW}, respectively. We used both classification accuracy and loss on testing set as the evaluation metrics. For the baselines, we report the performances of both TensorFlow-based implementations of GCN and GCN^T in (Sun et al., 2020) and our PyTorch-based implementation.[2] Table 2 shows that EigLearn consistently improved the baseline performances and reduced testing losses. Furthermore,

2. There are some subtle differences between the implementations in TensorFlow and PyTorch that lead to the performance discrepancies in different baseline implementations. For example, TensorFlow real-

Table 2: Comparison of model performances in terms of average values and standard deviations. PT denotes PyTorch, and EL denotes EigLearn. Leaders are in boldface.

	Testing Accuracy			Testing Loss		
model	Cora	CiteSeer	PubMed	Cora	CiteSeer	PubMed
GCN	80.5 ± 2.3	69.6 ± 2.0	78.2 ± 2.4	1.07 ± 0.04	1.36 ± 0.03	0.75 ± 0.04
FisherGCN	80.7 ± 2.2	69.8 ± 2.0	78.4 ± 2.4	1.06 ± 0.04	1.35 ± 0.03	0.74 ± 0.04
GCN (PT)	79.8 ± 2.1	69.7 ± 1.9	78.3 ± 2.2	0.66 ± 0.04	0.96 ± 0.04	0.58 ± 0.05
ELGCN	$\mathbf{81.4} \pm 1.5$	$\mathbf{70.1} \pm 1.8$	$\mathbf{78.9} \pm 2.1$	$\mathbf{0.59} \pm 0.04$	$\mathbf{0.94} \pm 0.04$	$\mathbf{0.57} \pm 0.05$
GCN^T	81.2 ± 2.3	70.3 ± 1.9	79.0 ± 2.6	1.04 ± 0.04	1.33 ± 0.03	0.70 ± 0.05
FisherGCN^T	81.5 ± 2.2	70.5 ± 1.7	79.3 ± 2.7	1.03 ± 0.03	1.32 ± 0.03	0.69 ± 0.04
GCN^T (PT)	81.1 ± 1.8	70.5 ± 1.7	79.3 ± 2.0	0.62 ± 0.04	0.92 ± 0.04	0.54 ± 0.04
ELGCN^T	$\mathbf{82.2} \pm 1.4$	$\mathbf{70.6} \pm 1.6$	$\mathbf{79.8} \pm 1.8$	$\mathbf{0.56} \pm 0.04$	$\mathbf{0.90} \pm 0.04$	$\mathbf{0.53} \pm 0.04$

EigLearnGCN and EigLearnGCN^T noticeably outperformed FisherGCN and FisherGCN^T correspondingly. EigLearn also yielded larger improvements to the baselines than both FisherGCN and FisherGCN^T. These larger improvements are significant, since compared to FisherGCN, EigLearn is easier to implement and fits more naturally into the neural network architectures. In addition, EigLearnGCN is significantly faster than FisherGCN in that the additional cost associated with EigLearn is only a small fraction of the cost for training a regular GCN, while FisherGCN is several times slower than GCN as reported in (Sun et al., 2020). It is also worth noting that EigLearn reduced the standard deviations of testing accuracy consistently, while FisherGCN^T increased the standard deviation for PubMed. Hence, these experiment results confirm that EigLearn is more accurate and robust than FisherGCN in terms of standard generalization.

4.3. Apply EigLearn on ChebyNet and SGCN

To testify the applicability of the EigLearn unit, we also inject it into ChebyNet (Defferrard et al., 2016) and Simplified-GCN (Wu et al., 2019), and observe performance improvement on SGC and ChebyNet with the EigLearn unit as shown in table 3 using the same settings in the previous section. The ChebyNet model is based on an order-2 Chebyshev polynomial of the shifted and rescaled Laplacian. The SGC model is based on a power-2 graph filter matrix.[3]

4.4. Comparison with TruncateTrain

In the methodology section we argued that the residual formulation to learn the eigenvalue perturbation is necessary. To validate it, we also conducted experiments to compare

izes the weight decay with L_2 regularization and includes the penalty in the total loss, while PyTorch implements the weight decay in the ADAM optimizer and excludes the penalty from the total loss.

3. The SGC performance in the original paper is based on the public fix split. We use the available implementation (https://github.com/Tiiiger/SGC) and perform random splits, and observe a performance degradation. The performance degradation on random split also happens to ChebyNet.

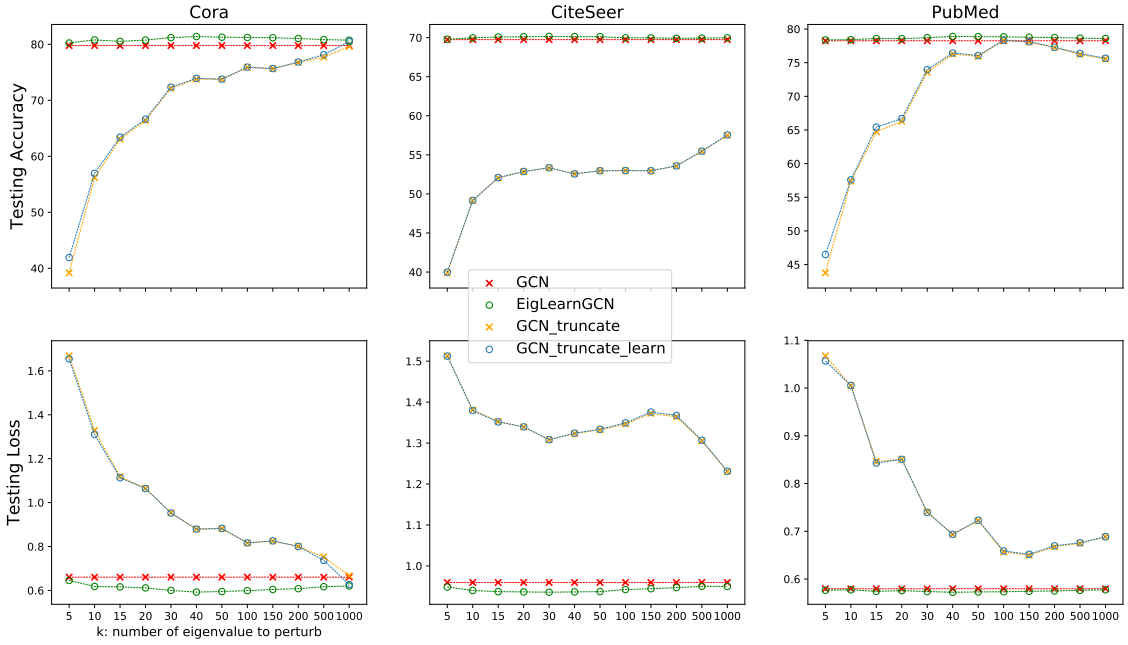

Figure 2: EigLearn v.s. TruncateTrain

Table 3: Comparison of model performances in terms of average values and standard deviations. EL denotes EigLearn. Leaders are in boldface.

	Testing Accuracy			Testing Loss		
model	Cora	CiteSeer	PubMed	Cora	CiteSeer	PubMed
ChebyNet	76.5 ± 2.4	68.5 ± 2.0	75.9 ± 2.6	0.74 ± 0.05	0.98 ± 0.04	0.65 ± 0.06
EL-ChebyNet	$\mathbf{78.5} \pm 2.0$	$\mathbf{69.1} \pm 2.0$	$\mathbf{76.3} \pm 2.5$	$\mathbf{0.68} \pm 0.05$	$\mathbf{0.96} \pm 0.04$	$\mathbf{0.65} \pm 0.06$
SGC	78.1 ± 2.7	69.2 ± 2.1	77.6 ± 2.5	0.81 ± 0.03	1.08 ± 0.03	0.58 ± 0.03
EL-SGC	$\mathbf{81.0} \pm 1.5$	$\mathbf{69.6} \pm 2.0$	$\mathbf{79.2} \pm 2.3$	$\mathbf{0.61} \pm 0.04$	$\mathbf{0.97} \pm 0.03$	$\mathbf{0.55} \pm 0.04$

EigLearn and the direct eigenvalue perturbation learning based on a truncated eigendecomposition of the graph filter matrix (denote it as TruncateTrain). As shown in Fig. 2, two observations are apparent. Firstly, only when we used a large number of eigenvectors from the graph filter matrix, could we preserve the model performance. This observation invalidates the linear complexity assumption in TruncateTrain. Secondly, directly learning the eigenvalue perturbation could not boost the model performance as effectively as EigLearn utilizing the residual formulation. Although we do not provide theoretical analyses on this, the straightforward explanations are: the residual formulation anchors the base level of the eigenvalues and performance; on top of that, the perturbations are learned on individual eigenvalues without changing the meaningful attenuation pattern of the spectrum.

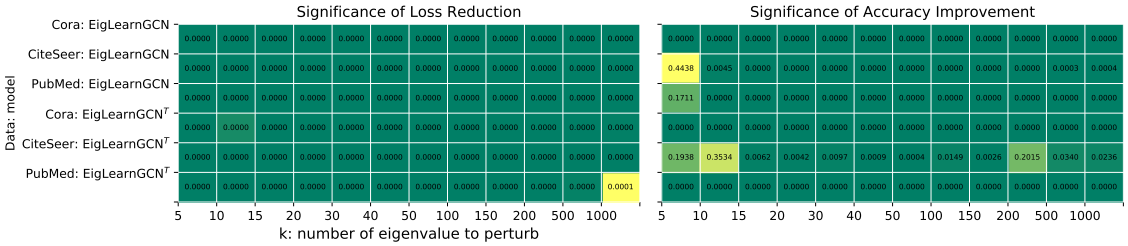

Figure 3: Pair-Sample t-test on EigLearn Performance Improvement

4.5. Experiment on Large Dataset

To demonstrate the scalability and generalization of EigLearn, we also run experiments on a significantly larger dataset, the arxiv network, which is roughly 10 times larger than PubMed in the number of nodes with more than 1 million edges, from the Open Graph Benchmark (Hu et al., 2021). We follow the detailed GCN settings provided in the official implementation from the OGB project. We train the eigenvalue perturbation with a learning rate of 0.002 for 50 epochs without regularization or early stopping. For computer memory issue, we use an order-3 filter matrix in GCN^T. We aggregate the results from ten runs (for both data random split and trainable parameter random initialization) in Table 4. EigLearn consistently improves on GCN with both filter matrices and across different numbers of eigenvalues. In term of algorithm efficiency, we compare the overall training time of GCN and EL-GCN (including computing V_k) and do not observe a significant time increase, i.e. (10 times average measured in seconds), 3.06 v.s. 4.18 on Cora, 3.44 v.s. 4.16 on CiteSeer, 6.44 v.s. 7.83 on Pubmed and 213.9 v.s. 273.6 on the large dataset ogbn-arxiv.

Table 4: Performance Improvement by EigLearn on ogbn-arxiv Dataset

	Training Accuracy			Testing Accuracy		
model	k=20	k=50	k=100	k=20	k=50	k=100
GCN	78.84 ± 0.57	79.17 ± 0.45	78.94 ± 0.59	73.86 ± 0.11	73.60 ± 0.10	73.74 ± 0.09
ELGCN	$\mathbf{80.27} \pm 0.06$	$\mathbf{80.45} \pm 0.07$	$\mathbf{80.27} \pm 0.12$	$\mathbf{74.14} \pm 0.08$	$\mathbf{73.86} \pm 0.09$	$\mathbf{73.98} \pm 0.11$
GCN^T	79.80 ± 0.42	79.42 ± 0.41	79.73 ± 0.49	73.67 ± 0.10	73.61 ± 0.11	73.62 ± 0.16
ELGCN^T	$\mathbf{81.45} \pm 0.07$	$\mathbf{81.43} \pm 0.10$	$\mathbf{81.13} \pm 0.12$	$\mathbf{74.08} \pm 0.11$	$\mathbf{74.02} \pm 0.07$	$\mathbf{73.99} \pm 0.07$

4.6. Other Experiment Results

To further demonstrate that the improvement induced by EigLearn is systematic, we conducted pair-sample t-tests across different datasets, graph filter matrices (GCN means symmetrically normalized adjacency and GCN^T is a higher-order polynomial) with perturbation on different k eigenvalues. As shown in Fig. 3, the t-tests reveal that the majority of experiment settings are statistically significant except for a few corner cases (e.g., when k is small).

Next, we show the sensitivity analysis on the EigLearn in Fig. 4. Essentially, EigLearn extracts the perturbation in the subspace spanned by k dominant eigenvectors of the graph

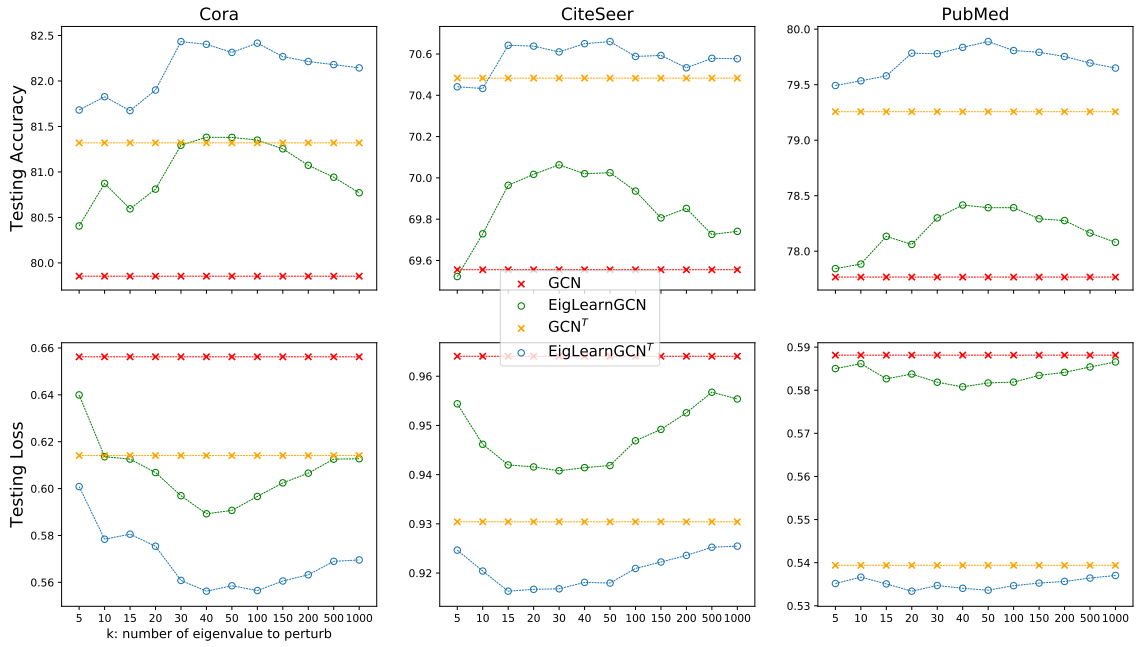

Figure 4: EigLearn Sensitivity on k

filter matrix. One observation from Fig. 4 is that the performance induced by EigLearn is relatively consistent within a reasonably wide band of k value (e.g., 20 to 150). It also confirms that we do not need a large number of eigenvectors to make EigLearn achieve high effectiveness. On the contrary, when k is large, the performance improvement diminishes. It is most probably due to overfitting when many trainable parameters are introduced to the model. When k is too small, the extra capacity added to the model is not sufficient for EigLearn to make sizable improvement. Besides, as a common practice, we conduct perturbation in the subspace spanned by the dominant eigenvectors instead of the least significant eigenvectors. We also ran experiments with the least significant eigenvectors. These eigenvectors barely made any performance improvement. This provides some experimental justification of using the significant eigenvectors and supports our argument that polynomial-based approaches may waste computation and model capacity on a large number of insignificant eigenvalues when the implicit perturbation is applied to the full spectrum.

We also checked the sensitivity on learning rate and regularization (weight decay) in the residual unit. In general, EigLearn is quite robust toward these two hyperparameters. Experimental studies exposed that a smaller learning rate was indeed effective in learning the perturbation and confirmed our assumption that the perturbations are small and should be learned with a moderate learning rate. As for regularization, it turned out that a small weight decay or even no weight decay led to slightly improved results, although it does not make a big difference. Additional experiment results can be found in the provided anonymous code URL.

5. Discussions and Conclusions

In this work, we introduced *EigLearn*, which perturbs the given graph filter matrix in its spectral domain for improving GCN performance. We formulated the eigenvalue perturbation of the graph filter matrix as a residual learning problem and realized it with a residual unit in neural network architecture without increasing the model complexity. Experimental results show that EigLearn makes more significant performance improvements than the previous works, including LanczosNet and FisherGCN. We also studied the detailed behaviors of EigLearn to demonstrate its robustness and provided a spectral explanation for the perturbation process. EigLearn is easy to implement and naturally fits in the neural-network configuration. We anticipate the future work resides in generalizing EigLearn to GCN for graph classification and regression and other graph learning problems. We also plan to investigate the integration of EigLearn with adversarial learning to improve adversarial robustness.

References

Dimitris Achlioptas and Frank McSherry. Fast computation of low-rank matrix approximations. *Journal of the ACM (JACM)*, 54(2):9–es, 2007.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

George Dasoulas, Johannes Lutzeyer, and Michalis Vazirgiannis. Learning parametrised graph shift operators. *arXiv preprint arXiv:2101.10050*, 2021.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *arXiv preprint arXiv:1606.09375*, 2016.

Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *International Conference on Machine Learning*, pages 1972–1982. PMLR, 2019.

Adnan Gavili and Xiao-Ping Zhang. On the shift operator, graph frequency, and optimal filtering in graph signal processing. *IEEE Transactions on Signal Processing*, 65(23):6303–6318, 2017.

Gene H Golub and Charles F Van Loan. *Matrix Computations*, volume 3. Johns Hopkins University Press, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Mikael Henaff, Joan Bruna, and Yann LeCun. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs, 2021.

Bo Jiang, Doudou Lin, Jin Tang, and Bin Luo. Data representation and learning with graph diffusion-embedding networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10414–10423, 2019.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Ioannis Koutis and Huong Le. Spectral modification of graphs for improved spectral clustering. In *NeurIPS*, pages 4859–4868, 2019.

Ioannis Koutis, Alex Levin, and Richard Peng. Faster spectral sparsification and numerical algorithms for SDD matrices. *ACM Transactions on Algorithms (TALG)*, 12(2):1–16, 2015.

Richard B Lehoucq, Danny C Sorensen, and Chao Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. SIAM, 1998.

Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. In *ICLR*, 2019.

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. *arXiv preprint arXiv:1912.01703*, 2019.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710, 2014.

Aliaksei Sandryhaila and José MF Moura. Discrete signal processing on graphs: Graph Fourier transform. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6167–6170. IEEE, 2013.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Daniel A Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *SIAM Journal on Computing*, 40(4):981–1025, 2011.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Ke Sun, Piotr Koniusz, and Zhen Wang. Fisher-Bures adversary graph convolutional networks. In *Uncertainty in Artificial Intelligence*, pages 465–475. PMLR, 2020.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, 2020.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, pages 6861–6871. PMLR, 2019.

Chenxiao Zhao, P Thomas Fletcher, Mixue Yu, Yaxin Peng, Guixu Zhang, and Chaomin Shen. The adversarial attack and detection under the fisher information metric. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5869–5876, 2019.