Diffusion Improves Graph Learning

Johannes Klicpera, Stefan Weißenberger, Stephan Günnemann
Technical University of Munich
{klicpera,stefan.weissenberger,guennemann}@in.tum.de

Abstract

Graph convolution is the core of most Graph Neural Networks (GNNs) and usually approximated by message passing between direct (one-hop) neighbors. In this work, we remove the restriction of using only the direct neighbors by introducing a powerful, yet spatially localized graph convolution: Graph diffusion convolution (GDC). GDC leverages generalized graph diffusion, examples of which are the heat kernel and personalized PageRank. It alleviates the problem of noisy and often arbitrarily defined edges in real graphs. We show that GDC is closely related to spectral-based models and thus combines the strengths of both spatial (message passing) and spectral methods. We demonstrate that replacing message passing with graph diffusion convolution consistently leads to significant performance improvements across a wide range of models on both supervised and unsupervised tasks and a variety of datasets. Furthermore, GDC is not limited to GNNs but can trivially be combined with any graph-based model or algorithm (e.g. spectral clustering) without requiring any changes to the latter or affecting its computational complexity. Our implementation is available online. [1]

1 Introduction

When people started using graphs for evaluating chess tournaments in the middle of the 19th century they only considered each player's direct opponents, i.e. their first-hop neighbors. Only later was the analysis extended to recursively consider higher-order relationships via $\boldsymbol{A}^2$, $\boldsymbol{A}^3$, etc. and finally generalized to consider all exponents at once, using the adjacency matrix's dominant eigenvector [38, 75]. The field of Graph Neural Networks (GNNs) is currently in a similar state. Graph Convolutional Networks (GCNs) [32], also referred to as Message Passing Neural Networks (MPNNs) [23] are the prevalent approach in this field but they only pass messages between neighboring nodes in each layer. These messages are then aggregated at each node to form the embedding for the next layer. While MPNNs do leverage higher-order neighborhoods in deeper layers, limiting each layer's messages to one-hop neighbors seems arbitrary. Edges in real graphs are often noisy or defined using an arbitrary threshold [70], so we can clearly improve upon this approach.

Since MPNNs only use the immediate neigborhod information, they are often referred to as spatial methods. On the other hand, spectral-based models do not just rely on first-hop neighbors and capture more complex graph properties [16]. However, while being theoretically more elegant, these methods are routinely outperformed by MPNNs on graph-related tasks [32, 74, 81] and do not generalize to previously unseen graphs. This shows that message passing is a powerful framework worth extending upon. To reconcile these two separate approaches and combine their strengths we propose a novel technique of performing message passing inspired by spectral methods: Graph diffusion convolution (GDC). Instead of aggregating information only from the first-hop neighbors, GDC aggregates information from a larger neighborhood. This neighborhood is constructed via a new graph generated by sparsifying a generalized form of graph diffusion. We show how graph diffusion

[1] https://www.daml.in.tum.de/gdc

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

is expressed as an equivalent polynomial filter and how GDC is closely related to spectral-based models while addressing their shortcomings. GDC is spatially localized, scalable, can be combined with message passing, and generalizes to unseen graphs. Furthermore, since GDC generates a new sparse graph it is not limited to MPNNs and can trivially be combined with *any* existing graph-based model or algorithm in a plug-and-play manner, i.e. without requiring changing the model or affecting its computational complexity. We show that GDC consistently improves performance across a wide range of models on both supervised and unsupervised tasks and various homophilic datasets. In summary, this paper's core contributions are:

1. Proposing graph diffusion convolution (GDC), a more powerful and general, yet spatially localized alternative to message passing that uses a sparsified generalized form of graph diffusion. GDC is not limited to GNNs and can be combined with any graph-based model or algorithm.
2. Analyzing the spectral properties of GDC and graph diffusion. We show how graph diffusion is expressed as an equivalent polynomial filter and analyze GDC's effect on the graph spectrum.
3. Comparing and evaluating several specific variants of GDC and demonstrating its wide applicability to supervised and unsupervised learning on graphs.

2 Generalized graph diffusion

We consider an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V}$ and edge set $\mathcal{E}$. We denote with $N = |\mathcal{V}|$ the number of nodes and $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ the adjacency matrix. We define generalized graph diffusion via the diffusion matrix

$$\boldsymbol{S} = \sum_{k=0}^{\infty} \theta_k \boldsymbol{T}^k, \tag{1}$$

with the weighting coefficients θ_k, and the generalized transition matrix $\boldsymbol{T}$. The choice of θ_k and $\boldsymbol{T}^k$ must at least ensure that Eq. 1 converges. In this work we will consider somewhat stricter conditions and require that $\sum_{k=0}^{\infty} \theta_k = 1$, $\theta_k \in [0, 1]$, and that the eigenvalues of $\boldsymbol{T}$ are bounded by $\lambda_i \in [0, 1]$, which together are sufficient to guarantee convergence. Note that regular graph diffusion commonly requires $\boldsymbol{T}$ to be column- or row-stochastic.

Transition matrix. Examples for $\boldsymbol{T}$ in an undirected graph include the random walk transition matrix $\boldsymbol{T}_{\text{rw}} = \boldsymbol{A}\boldsymbol{D}^{-1}$ and the symmetric transition matrix $\boldsymbol{T}_{\text{sym}} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, where the degree matrix $\boldsymbol{D}$ is the diagonal matrix of node degrees, i.e. $D_{ii} = \sum_{j=1}^{N} A_{ij}$. Note that in our definition $\boldsymbol{T}_{\text{rw}}$ is column-stochastic. We furthermore adjust the random walk by adding (weighted) self-loops to the original adjacency matrix, i.e. use $\tilde{\boldsymbol{T}}_{\text{sym}} = (w_{\text{loop}}\boldsymbol{I}_N + \boldsymbol{D})^{-1/2}(w_{\text{loop}}\boldsymbol{I}_N + \boldsymbol{A})(w_{\text{loop}}\boldsymbol{I}_N + \boldsymbol{D})^{-1/2}$, with the self-loop weight $w_{\text{loop}} \in \mathbb{R}^+$. This is equivalent to performing a lazy random walk with a probability of staying at node i of $p_{\text{stay},i} = w_{\text{loop}}/D_i$.

Special cases. Two popular examples of graph diffusion are personalized PageRank (PPR) [57] and the heat kernel [36]. PPR corresponds to choosing $\boldsymbol{T} = \boldsymbol{T}_{\text{rw}}$ and $\theta_k^{\text{PPR}} = \alpha(1-\alpha)^k$, with teleport probability $\alpha \in (0, 1)$ [14]. The heat kernel uses $\boldsymbol{T} = \boldsymbol{T}_{\text{rw}}$ and $\theta_k^{\text{HK}} = e^{-t}\frac{t^k}{k!}$, with the diffusion time t [14]. Another special case of generalized graph diffusion is the approximated graph convolution introduced by Kipf & Welling [32], which translates to $\theta_1 = 1$ and $\theta_k = 0$ for $k \neq 1$ and uses $\boldsymbol{T} = \tilde{\boldsymbol{T}}_{\text{sym}}$ with $w_{\text{loop}} = 1$.

Weighting coefficients. We compute the series defined by Eq. 1 either in closed-form, if possible, or by restricting the sum to a finite number K. Both the coefficients defined by PPR and the heat kernel give a closed-form solution for this series that we found to perform well for the tasks considered. Note that we are not restricted to using $\boldsymbol{T}_{\text{rw}}$ and can use any generalized transition matrix along with the coefficients θ_k^{PPR} or θ_k^{HK} and the series still converges. We can furthermore choose θ_k by repurposing the graph-specific coefficients obtained by methods that optimize coefficients analogous to θ_k as part of their training process. We investigated this approach using label propagation [8, 13] and node embedding models [1]. However, we found that the simple coefficients defined by PPR or the heat kernel perform better than those learned by these models (see Fig. 7 in Sec. 6).

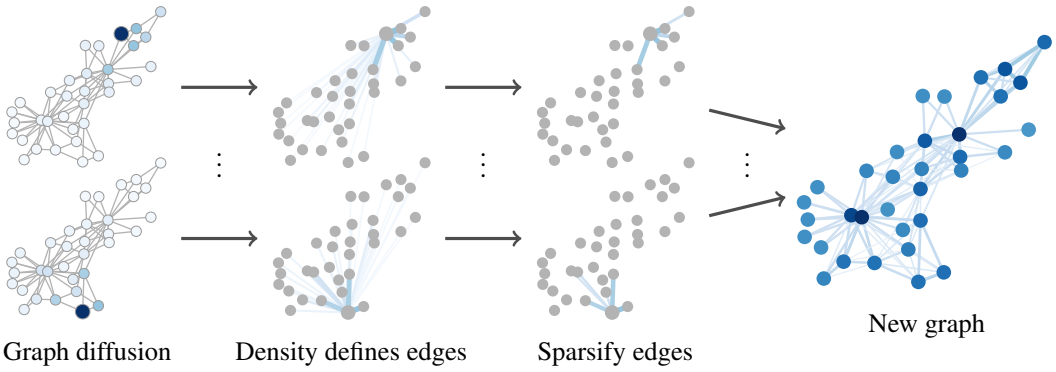


Figure 1: Illustration of graph diffusion convolution (GDC). We transform a graph $\boldsymbol{A}$ via graph diffusion and sparsification into a new graph $\tilde{\boldsymbol{S}}$ and run the given model on this graph instead.

3 Graph diffusion convolution

Essentially, graph diffusion convolution (GDC) exchanges the normal adjacency matrix $\boldsymbol{A}$ with a sparsified version $\tilde{\boldsymbol{S}}$ of the generalized graph diffusion matrix $\boldsymbol{S}$, as illustrated by Fig. 1. This matrix defines a weighted and directed graph, and the model we aim to augment is applied to this graph instead. We found that the calculated edge weights are beneficial for the tasks considered. However, we even found that GDC works when ignoring the weights after sparsification. This enables us to use GDC with models that only support unweighted edges such as the degree-corrected stochastic block model (DCSBM). If required, we make the graph undirected by using $(\tilde{\boldsymbol{S}} + \tilde{\boldsymbol{S}}^T)/2$, e.g. for spectral clustering. With these adjustments GDC is applicable to *any* graph-based model or algorithm.

Intuition. The general intuition behind GDC is that graph diffusion smooths out the neighborhood over the graph, acting as a kind of denoising filter similar to Gaussian filters on images. This helps with graph learning since both features and edges in real graphs are often noisy. Previous works also highlighted the effectiveness of graph denoising. Berberidis & Giannakis [7] showed that PPR is able to reconstruct the underlying probability matrix of a sampled stochastic block model (SBM) graph. Kloumann et al. [35] and Ragain [64] showed that PPR is optimal in recovering the SBM and DCSBM clusters in the space of landing probabilities under the mean field assumption. Li et al. [40] generalized this result by analyzing the convergence of landing probabilities to their mean field values. These results confirm the intuition that graph diffusion-based smoothing indeed recovers meaningful neighborhoods from noisy graphs.

Sparsification. Most graph diffusions result in a dense matrix $\boldsymbol{S}$. This happens even if we do not sum to $k = \infty$ in Eq. 1 due to the "four/six degrees of separation" in real-world graphs [5]. However, the values in $\boldsymbol{S}$ represent the influence between all pairs of nodes, which typically are highly localized [54]. This is a major advantage over spectral-based models since the spectral domain does not provide any notion of locality. Spatial localization allows us to simply truncate small values of $\boldsymbol{S}$ and recover sparsity, resulting in the matrix $\tilde{\boldsymbol{S}}$. In this work we consider two options for sparsification: 1. top-k: Use the k entries with the highest mass per column, 2. Threshold ϵ: Set entries below ϵ to zero. Sparsification would still require calculating a dense matrix $\boldsymbol{S}$ during preprocessing. However, many popular graph diffusions can be approximated efficiently and accurately in linear time and space. Most importantly, there are fast approximations for both PPR [3, 77] and the heat kernel [34], with which GDC achieves a linear runtime $\mathcal{O}(N)$. Furthermore, top-k truncation generates a regular graph, which is amenable to batching methods and solves problems related to widely varying node degrees [15]. Empirically, we even found that sparsification slightly *improves* prediction accuracy (see Fig. 5 in Sec. 6). After sparsification we calculate the (symmetric or random walk) transition matrix on the resulting graph via $\boldsymbol{T}_{\text{sym}}^{\tilde{\boldsymbol{S}}} = \boldsymbol{D}_{\tilde{\boldsymbol{S}}}^{-1/2} \tilde{\boldsymbol{S}} \boldsymbol{D}_{\tilde{\boldsymbol{S}}}^{-1/2}$.

Limitations. GDC is based on the assumption of homophily, i.e. "birds of a feather flock together" [49]. Many methods share this assumption and most common datasets adhere to this principle. However, this is an often overlooked limitation and it seems non-straightforward to overcome. One way of extending GDC to heterophily, i.e. "opposites attract", might be negative edge weights

[17, 44]. Furthermore, we suspect that GDC does not perform well in settings with more complex edges (e.g. knowledge graphs) or graph reconstruction tasks such as link prediction. Preliminary experiments showed that GDC indeed does not improve link prediction performance.

4 Spectral analysis of GDC

Even though GDC is a spatial-based method it can also be interpreted as a graph convolution and analyzed in the graph spectral domain. In this section we show how generalized graph diffusion is expressed as an equivalent polynomial filter and vice versa. Additionally, we perform a spectral analysis of GDC, which highlights the tight connection between GDC and spectral-based models.

Spectral graph theory. To employ the tools of spectral theory to graphs we exchange the regular Laplace operator with either the unnormalized Laplacian $\boldsymbol{L}_{\text{un}} = \boldsymbol{D} - \boldsymbol{A}$, the random-walk normalized $\boldsymbol{L}_{\text{rw}} = \boldsymbol{I}_N - \boldsymbol{T}_{\text{rw}}$, or the symmetric normalized graph Laplacian $\boldsymbol{L}_{\text{sym}} = \boldsymbol{I}_N - \boldsymbol{T}_{\text{sym}}$ [76]. The Laplacian's eigendecomposition is $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$, where both $\boldsymbol{U}$ and $\boldsymbol{\Lambda}$ are real-valued. The graph Fourier transform of a vector $\boldsymbol{x}$ is then defined via $\hat{\boldsymbol{x}} = \boldsymbol{U}^T\boldsymbol{x}$ and its inverse as $\boldsymbol{x} = \boldsymbol{U}\hat{\boldsymbol{x}}$. Using this we define a graph convolution on $\mathcal{G}$ as $\boldsymbol{x} *_{\mathcal{G}} \boldsymbol{y} = \boldsymbol{U}((\boldsymbol{U}^T\boldsymbol{x}) \odot (\boldsymbol{U}^T\boldsymbol{y}))$, where $\odot$ denotes the Hadamard product. Hence, a filter g_ξ with parameters ξ acts on $\boldsymbol{x}$ as $g_\xi(\boldsymbol{L})\boldsymbol{x} = \boldsymbol{U}\hat{\boldsymbol{G}}_\xi(\boldsymbol{\Lambda})\boldsymbol{U}^T\boldsymbol{x}$, where $\hat{\boldsymbol{G}}_\xi(\boldsymbol{\Lambda}) = \text{diag}(\hat{g}_{\xi,1}(\boldsymbol{\Lambda}), \dots, \hat{g}_{\xi,N}(\boldsymbol{\Lambda}))$. A common choice for g_ξ in the literature is a polynomial filter of order J, since it is localized and has a limited number of parameters [16, 27]:

$$g_\xi(\boldsymbol{L}) = \sum_{j=0}^{J} \xi_j \boldsymbol{L}^j = \boldsymbol{U} \left(\sum_{j=0}^{J} \xi_j \boldsymbol{\Lambda}^j \right) \boldsymbol{U}^T. \tag{2}$$

Graph diffusion as a polynomial filter. Comparing Eq. 1 with Eq. 2 shows the close relationship between polynomial filters and generalized graph diffusion since we only need to exchange $\boldsymbol{L}$ by $\boldsymbol{T}$ to go from one to the other. To make this relationship more specific and find a direct correspondence between GDC with θ_k and a polynomial filter with parameters ξ_j we need to find parameters that solve

$$\sum_{j=0}^{J} \xi_j \boldsymbol{L}^j \stackrel{!}{=} \sum_{k=0}^{K} \theta_k \boldsymbol{T}^k. \tag{3}$$

To find these parameters we choose the Laplacian corresponding to $\boldsymbol{L} = \boldsymbol{I}_n - \boldsymbol{T}$, resulting in (see App. A)

$$\xi_j = \sum_{k=j}^{K} \binom{k}{j} (-1)^j \theta_k, \qquad \theta_k = \sum_{j=k}^{J} \binom{j}{k} (-1)^k \xi_j, \tag{4}$$

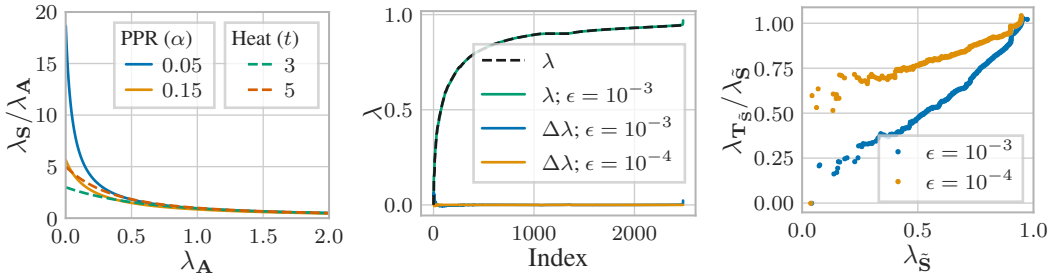
which shows the direct correspondence between graph diffusion and spectral methods. Note that we need to set $J = K$. Solving Eq. 4 for the coefficients corresponding to the heat kernel θ_k^{HK} and PPR θ_k^{PPR} leads to

$$\xi_j^{\text{HK}} = \frac{(-t)^j}{j!}, \qquad \xi_j^{\text{PPR}} = \left(1 - \frac{1}{\alpha}\right)^j, \tag{5}$$

showing how the heat kernel and PPR are expressed as polynomial filters. Note that PPR's corresponding polynomial filter converges only if $\alpha > 0.5$. This is caused by changing the order of summation when deriving ξ_j^{PPR}, which results in an alternating series. However, if the series does converge it gives the exact same transformation as the equivalent graph diffusion.

Spectral properties of GDC. We will now extend the discussion to all parts of GDC and analyze how they transform the graph Laplacian's eigenvalues. GDC consists of four steps: 1. Calculate the transition matrix $\boldsymbol{T}$, 2. take the sum in Eq. 1 to obtain $\boldsymbol{S}$, 3. sparsify the resulting matrix by truncating small values, resulting in $\tilde{\boldsymbol{S}}$, and 4. calculate the transition matrix $\boldsymbol{T}_{\tilde{\boldsymbol{S}}}$.

1. Transition matrix. Calculating the transition matrix $\boldsymbol{T}$ only changes which Laplacian matrix we use for analyzing the graph's spectrum, i.e. we use $\boldsymbol{L}_{\text{sym}}$ or $\boldsymbol{L}_{\text{rw}}$ instead of $\boldsymbol{L}_{\text{un}}$. Adding self-loops to obtain $\tilde{\boldsymbol{T}}$ does not preserve the eigenvectors and its effect therefore cannot be calculated precisely. Wu et al. [78] empirically found that adding self-loops shrinks the graph's eigenvalues.



(a) Graph diffusion as a filter, PPR with α and heat kernel with t. Both act as low-pass filters.

(b) Sparsification with threshold ϵ of PPR ($\alpha = 0.1$) on CORA. Eigenvalues are almost unchanged.

(c) Transition matrix on CORA's sparsified graph $\tilde{\boldsymbol{S}}$. This acts as a weak high-pass filter.

Figure 2: Influence of different parts of GDC on the Laplacian's eigenvalues λ.

2. Sum over $\boldsymbol{T}^k$. Summation does not affect the eigenvectors of the original matrix, since $\boldsymbol{T}^k\boldsymbol{v}_i = \lambda_i\boldsymbol{T}^{k-1}\boldsymbol{v}_i = \lambda_i^k\boldsymbol{v}_i$, for the eigenvector $\boldsymbol{v}_i$ of $\boldsymbol{T}$ with associated eigenvalue λ_i. This also shows that the eigenvalues are transformed as

$$\tilde{\lambda}_i = \sum_{k=0}^{\infty}\theta_k\lambda_i^k. \tag{6}$$

Since the eigenvalues of $\boldsymbol{T}$ are bounded by 1 we can use the geometric series to derive a closed-form expression for PPR, i.e. $\tilde{\lambda}_i = \alpha\sum_{k=0}^{\infty}(1-\alpha)^k\lambda_i^k = \frac{\alpha}{1-(1-\alpha)\lambda_i}$. For the heat kernel we use the exponential series, resulting in $\tilde{\lambda}_i = e^{-t}\sum_{k=0}^{\infty}\frac{t^k}{k!}\lambda_i^k = e^{t(\lambda_i-1)}$. How this transformation affects the corresponding Laplacian's eigenvalues is illustrated in Fig. 2a. Both PPR and the heat kernel act as low-pass filters. Low eigenvalues corresponding to large-scale structure in the graph (e.g. clusters [55]) are amplified, while high eigenvalues corresponding to fine details but also noise are suppressed.

3. Sparsification. Sparsification changes both the eigenvalues and the eigenvectors, which means that there is no direct correspondence between the eigenvalues of $\boldsymbol{S}$ and $\tilde{\boldsymbol{S}}$ and we cannot analyze its effect analytically. However, we can use eigenvalue perturbation theory (Stewart & Sun [69], Corollary 4.13) to derive the upper bound

$$\sqrt{\sum_{i=1}^{N}(\tilde{\lambda}_i - \lambda_i)^2} \leq ||\boldsymbol{E}||_{\mathrm{F}} \leq N||\boldsymbol{E}||_{\max} \leq N\epsilon, \tag{7}$$

with the perturbation matrix $\boldsymbol{E} = \tilde{\boldsymbol{S}} - \boldsymbol{S}$ and the threshold ϵ. This bound significantly overestimates the perturbation since PPR and the heat kernel both exhibit strong localization on real-world graphs and hence the change in eigenvalues empirically does not scale with N (or, rather, $\sqrt{N}$). By ordering the eigenvalues we see that, empirically, the typical thresholds for sparsification have almost no effect on the eigenvalues, as shown in Fig. 2b and in the close-up in Fig. 11 in App. B.2. We find that the small changes caused by sparsification mostly affect the highest and lowest eigenvalues. The former correspond to very large clusters and long-range interactions, which are undesired for local graph smoothing. The latter correspond to spurious oscillations, which are not helpful for graph learning either and most likely affected because of the abrupt cutoff at ϵ.

4. Transition matrix on $\tilde{\boldsymbol{S}}$. As a final step we calculate the transition matrix on the resulting graph $\tilde{\boldsymbol{S}}$. This step does not just change which Laplacian we consider since we have already switched to using the transition matrix in step 1. It furthermore does not preserve the eigenvectors and is thus again best investigated empirically by ordering the eigenvalues. Fig. 2c shows that, empirically, this step slightly dampens low eigenvalues. This may seem counterproductive. However, the main purpose of using the transition matrix is ensuring that sparsification does not cause nodes to be treated differently by losing a different number of adjacent edges. The filtering is only a side-effect.

Limitations of spectral-based models. While there are tight connections between GDC and spectral-based models, GDC is actually spatial-based and therefore does not share their limitations. Similar to polynomial filters, GDC does not compute an expensive eigenvalue decomposition, preserves locality on the graph and is not limited to a single graph after training, i.e. typically the same coefficients θ_k

can be used across graphs. The choice of coefficients θ_k depends on the type of graph at hand and does not change significantly between similar graphs. Moreover, the hyperparameters α of PPR and t of the heat kernel usually fall within a narrow range that is rather insensitive to both the graph and model (see Fig. 8 in Sec. 6).

5 Related work

Graph diffusion and random walks have been extensively studied in classical graph learning [13, 14, 36, 37], especially for clustering [34], semi-supervised classification [12, 22], and recommendation systems [44]. For an overview of existing methods see Masuda et al. [46] and Fouss et al. [22].

The first models similar in structure to current Graph Neural Networks (GNNs) were proposed by Sperduti & Starita [68] and Baskin et al. [6], and the name GNN first appeared in [24, 65]. However, they only became widely adopted in recent years, when they started to outperform classical models in many graph-related tasks [19, 33, 42, 82]. In general, GNNs are classified into spectral-based models [11, 16, 28, 32, 41], which are based on the eigendecomposition of the graph Laplacian, and spatial-based methods [23, 26, 43, 52, 56, 62, 74], which use the graph directly and form new representations by aggregating the representations of a node and its neighbors. However, this distinction is often rather blurry and many models can not be clearly attributed to one type or the other. Deep learning also inspired a variety of unsupervised node embedding methods. Most models use random walks to learn node embeddings in a similar fashion as word2vec [51] [25, 61] and have been shown to implicitly perform a matrix factorization [63]. Other unsupervised models learn Gaussian distributions instead of vectors [10], use an auto-encoder [31], or train an encoder by maximizing the mutual information between local and global embeddings [73].

There have been some isolated efforts of using extended neighborhoods for aggregation in GNNs and graph diffusion for node embeddings. PPNP [33] propagates the node predictions generated by a neural network using personalized PageRank, DCNN [4] extends node features by concatenating features aggregated using the transition matrices of k-hop random walks, GraphHeat [79] uses the heat kernel and PAN [45] the transition matrix of maximal entropy random walks to aggregate over nodes in each layer, PinSage [82] uses random walks for neighborhood aggregation, and MixHop [2] concatenates embeddings aggregated using the transition matrices of k-hop random walks before each layer. VERSE [71] learns node embeddings by minimizing KL-divergence from the PPR matrix to a low-rank approximation. Attention walk [1] uses a similar loss to jointly optimize the node embeddings and diffusion coefficients θ_k. None of these works considered sparsification, generalized graph diffusion, spectral properties, or using preprocessing to generalize across models.

6 Experimental results

Experimental setup. We take extensive measures to prevent any kind of bias in our results. We optimize the hyperparameters of *all* models on *all* datasets with both the unmodified graph and all GDC variants *separately* using a combination of grid and random search on the validation set. Each result is averaged across 100 data splits and random initializations for supervised tasks and 20 random initializations for unsupervised tasks, as suggested by Klicpera et al. [33] and Shchur et al. [67]. We report performance on a test set that was used exactly *once*. We report all results as averages with $95\,\%$ confidence intervals calculated via bootstrapping.

We use the symmetric transition matrix with self-loops $\tilde{\boldsymbol{T}}_{\text{sym}} = (\boldsymbol{I}_N + \boldsymbol{D})^{-1/2}(\boldsymbol{I}_N + \boldsymbol{A})(\boldsymbol{I}_N + \boldsymbol{D})^{-1/2}$ for GDC and the column-stochastic transition matrix $\boldsymbol{T}_{\text{rw}}^{\tilde{\boldsymbol{S}}} = \tilde{\boldsymbol{S}}\boldsymbol{D}_{\tilde{\boldsymbol{S}}}^{-1}$ on $\tilde{\boldsymbol{S}}$. We present two simple and effective choices for the coefficients θ_k: The heat kernel and PPR. The diffusion matrix $\boldsymbol{S}$ is sparsified using either an ϵ-threshold or top-k.

Datasets and models. We evaluate GDC on six datasets: The citation graphs CITESEER [66], CORA [48], and PUBMED [53], the co-author graph COAUTHOR CS [67], and the co-purchase graphs AMAZON COMPUTERS and AMAZON PHOTO [47, 67]. We only use their largest connected components. We show how GDC impacts the performance of 9 models: Graph Convolutional Network (GCN) [32], Graph Attention Network (GAT) [74], jumping knowledge network (JK) [80], Graph Isomorphism Network (GIN) [81], and ARMA [9] are supervised models. The degree-corrected stochastic block model (DCSBM) [30], spectral clustering (using $\boldsymbol{L}_{\text{sym}}$) [55], DeepWalk

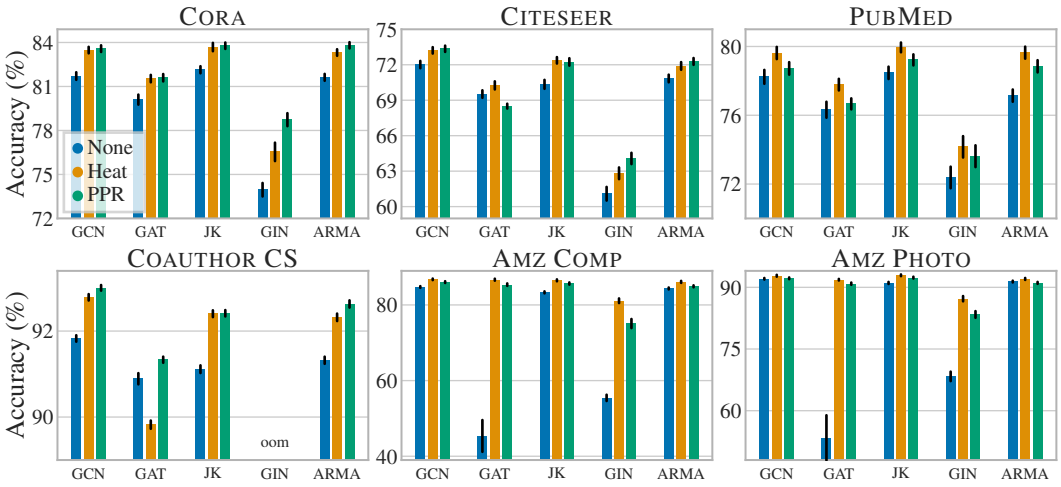


Figure 3: Node classification accuracy of GNNs with and without GDC. GDC consistently improves accuracy across models and datasets. It is able to fix models whose accuracy otherwise breaks down.

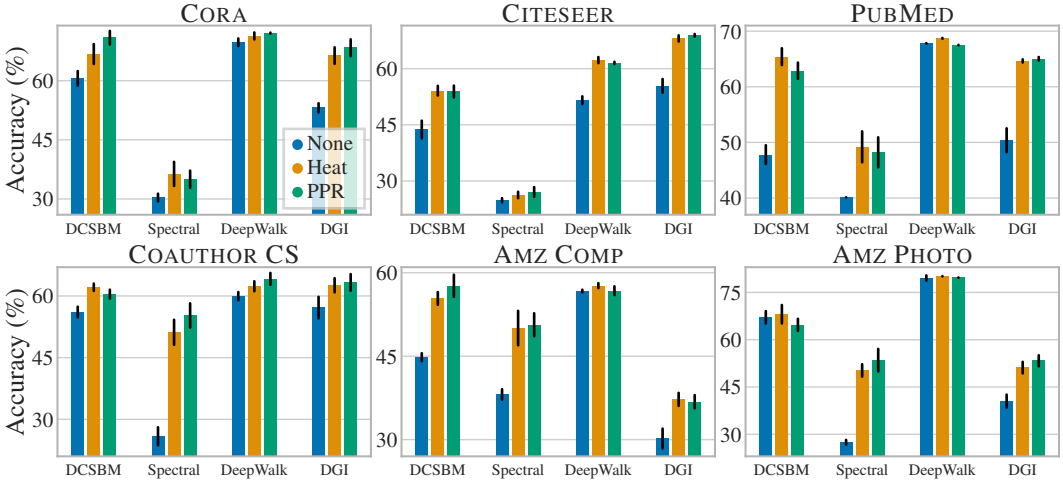


Figure 4: Clustering accuracy with and without GDC. GDC consistently improves the accuracy across a diverse set of models and datasets.

[61], and Deep Graph Infomax (DGI) [73] are unsupervised models. Note that DGI uses node features while other unsupervised models do not. We use k-means clustering to generate clusters from node embeddings. Dataset statistics and hyperparameters are reported in App. B.

Semi-supervised node classification. In this task the goal is to label nodes based on the graph, node features $\boldsymbol{X} \in \mathbb{R}^{N \times F}$ and a subset of labeled nodes $\boldsymbol{y}$. The main goal of GDC is improving the performance of MPNN models. Fig. 3 shows that GDC consistently and significantly improves the accuracy of a wide variety of state-of-the-art models across multiple diverse datasets. Note how GDC is able to fix the performance of GNNs that otherwise break down on some datasets (e.g. GAT). We also surpass or match the previous state of the art on all datasets investigated (see App. B.2).

Clustering. We highlight GDC's ability to be combined with any graph-based model by reporting the performance of a diverse set of models that use a wide range of paradigms. Fig. 4 shows the unsupervised accuracy obtained by matching clusters to ground-truth classes using the Hungarian algorithm. Accuracy consistently and significantly improves for all models and datasets. Note that spectral clustering uses the graph's eigenvectors, which are not affected by the diffusion step itself. Still, its performance improves by up to 30 percentage points. Results in tabular form are presented in App. B.2.

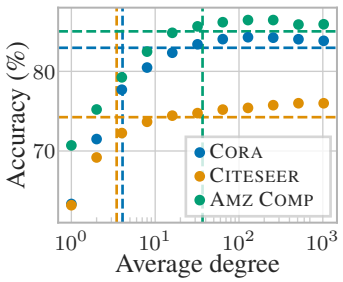


Figure 5: GCN+GDC accuracy (using PPR and top-k). Lines indicate original accuracy and degree. GDC surpasses original accuracy at around the same degree independent of dataset. Sparsification often improves accuracy.

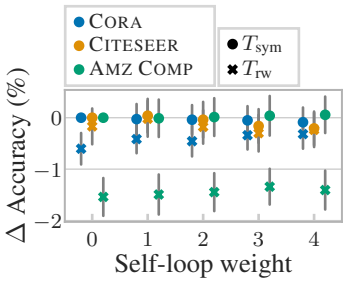


Figure 6: Difference in GCN+GDC accuracy (using PPR and top-k, percentage points) compared to the symmetric $\boldsymbol{T}_{\text{sym}}$ without self-loops. $\boldsymbol{T}_{\text{rw}}$ performs worse and self-loops have no significant effect.

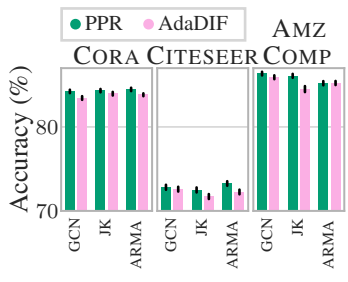


Figure 7: Accuracy of GDC with coefficients θ_k defined by PPR and learned by AdaDIF. Simple PPR coefficients consistently perform better than those obtained by AdaDIF, even with optimized regularization.

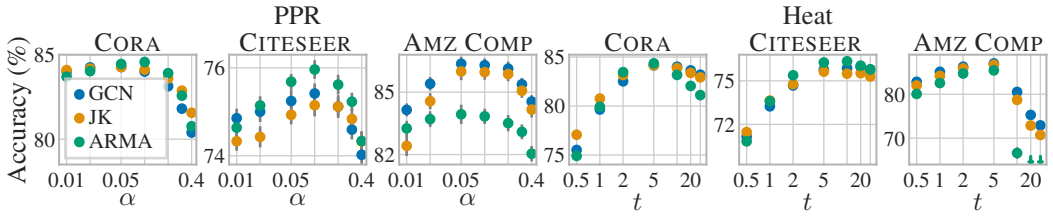


Figure 8: Accuracy achieved by using GDC with varying hyperparameters of PPR (α) and the heat kernel (t). Optimal values fall within a narrow range that is consistent across datasets and models.

In this work we concentrate on node-level prediction tasks in a transductive setting. However, GDC can just as easily be applied to inductive problems or different tasks like graph classification. In our experiments we found promising, yet not as consistent results for graph classification (e.g. 2.5 percentage points with GCN on the DD dataset [18]). We found no improvement for the inductive setting on PPI [50], which is rather unsurprising since the underlying data used for graph construction already includes graph diffusion-like mechanisms (e.g. regulatory interactions, protein complexes, and metabolic enzyme-coupled interactions). We furthermore conducted experiments to answer five important questions:

Does GDC increase graph density? When sparsifying the generalized graph diffusion matrix $\boldsymbol{S}$ we are free to choose the resulting level of sparsity in $\tilde{\boldsymbol{S}}$. Fig. 5 indicates that, surprisingly, GDC requires roughly the same average degree to surpass the performance of the original graph independent of the dataset and its average degree (ϵ-threshold in App. B.2, Fig. 12). This suggests that the sparsification hyperparameter can be obtained from a fixed average degree. Note that CORA and CITESEER are both small graphs with low average degree. Graphs become denser with size [39] and in practice we expect GDC to typically *reduce* the average degree at constant accuracy. Fig. 5 furthermore shows that there is an optimal degree of sparsity above which the accuracy decreases. This indicates that sparsification is not only computationally beneficial but also improves prediction performance.

How to choose the transition matrix $\boldsymbol{T}$? We found $\boldsymbol{T}_{\text{sym}}$ to perform best across datasets. More specifically, Fig. 6 shows that the symmetric version on average outperforms the random walk transition matrix $\boldsymbol{T}_{\text{rw}}$. This figure also shows that GCN accuracy is largely insensitive to self-loops when using $\boldsymbol{T}_{\text{sym}}$ – all changes lie within the estimated uncertainty. However, we did find that other models, e.g. GAT, perform better with self-loops (not shown).

How to choose the coefficients θ_k? We found the coefficients defined by PPR and the heat kernel to be effective choices for θ_k. Fig. 8 shows that their optimal hyperparameters typically fall within a narrow range of $\alpha \in [0.05, 0.2]$ and $t \in [1, 10]$. We also tried obtaining θ_k from models that learn analogous coefficients [1, 8, 13]. However, we found that θ_k obtained by these models tend to converge to a minimal neighborhood, i.e. they converge to $\theta_0 \approx 1$ or $\theta_1 \approx 1$ and all other $\theta_k \approx 0$.

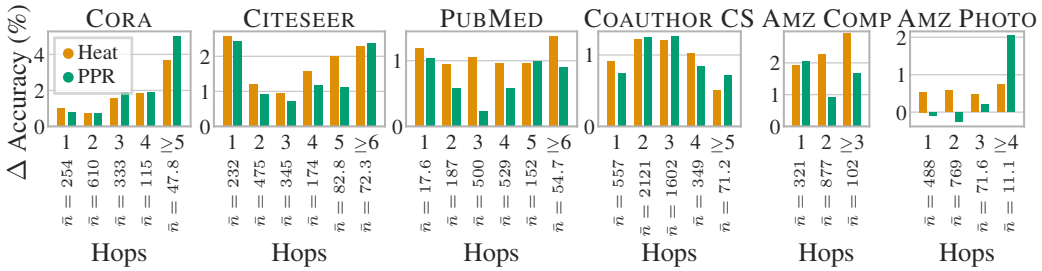


Figure 10: Improvement (percentage points) in GCN accuracy by adding GDC depending on distance (number of hops) from the training set. Nodes further away tend to benefit more from GDC.

This is caused by their training losses almost always decreasing when the considered neighborhood shrinks. We were able to control this overfitting to some degree using strong regularization (specifically, we found L_2 regularization on the difference of neighboring coefficients $\theta_{k+1} - \theta_k$ to perform best). However, this requires hand-tuning the regularization for every dataset, which defeats the purpose of *learning* the coefficients from the graph. Moreover, we found that even with hand-tuned regularization the coefficients defined by PPR and the heat kernel perform better than trained θ_k, as shown in Fig. 7.

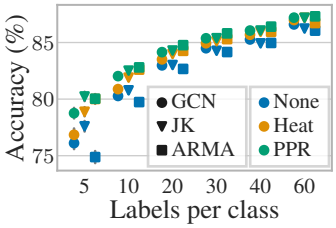


Figure 9: Accuracy on Cora with different label rates. Improvement from GDC increases for sparser label rates.

How does the label rate affect GDC? When varying the label rate from 5 up to 60 labels per class we found that the improvement achieved by GDC increases the sparser the labels are. Still, GDC improves performance even for 60 labels per class, i.e. $17\,\%$ label rate (see Fig. 9). This trend is most likely due to larger neighborhood leveraged by GDC.

Which nodes benefit from GDC? Our experiments showed no correlation of improvement with most common node properties, except for the distance from the training set. Nodes further away from the training set tend to benefit more from GDC, as demonstrated by Fig. 10. Besides smoothing out the neighborhood, GDC also has the effect of increasing the model's range, since it is no longer restricted to only using first-hop neighbors. Hence, nodes further away from the training set influence the training and later benefit from the improved model weights.

7 Conclusion

We propose graph diffusion convolution (GDC), a method based on sparsified generalized graph diffusion. GDC is a more powerful, yet spatially localized extension of message passing in GNNs, but able to enhance any graph-based model. We show the tight connection between GDC and spectral-based models and analyzed GDC's spectral properties. GDC shares many of the strengths of spectral methods and none of their weaknesses. We conduct extensive and rigorous experiments that show that GDC consistently improves the accuracy of a wide range of models on both supervised and unsupervised tasks across various homophilic datasets and requires very little hyperparameter tuning. There are many extensions and applications of GDC that remain to be explored. We expect many graph-based models and tasks to benefit from GDC, e.g. graph classification and regression. Promising extensions include other diffusion coefficients θ_k such as those given by the methods presented in Fouss et al. [22] and more advanced random walks and operators that are not defined by powers of a transition matrix.

Acknowledgments

This research was supported by the German Federal Ministry of Education and Research (BMBF), grant no. 01IS18036B, and by the Deutsche Forschungsgemeinschaft (DFG) through the Emmy Noether grant GU 1409/2-1 and the TUM International Graduate School of Science and Engineering (IGSSE), GSC 81. The authors of this work take full responsibilities for its content.

References

[1] Sami Abu-El-Haija, Bryan Perozzi, Rami Al-Rfou, and Alex Alemi. Watch Your Step: Learning Node Embeddings via Graph Attention. In *NeurIPS*, 2018.

[2] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, Hrayr Harutyunyan, Greg Ver Steeg, and Aram Galstyan. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *ICML*, 2019.

[3] R. Andersen, F. Chung, and K. Lang. Local Graph Partitioning using PageRank Vectors. In *FOCS*, 2006.

[4] James Atwood and Don Towsley. Diffusion-Convolutional Neural Networks. In *NIPS*, 2016.

[5] Lars Backstrom, Paolo Boldi, Marco Rosa, Johan Ugander, and Sebastiano Vigna. Four degrees of separation. In *ACM Web Science Conference*, 2012.

[6] Igor I. Baskin, Vladimir A. Palyulin, and Nikolai S. Zefirov. A Neural Device for Searching Direct Correlations between Structures and Properties of Chemical Compounds. *Journal of Chemical Information and Computer Sciences*, 37(4):715–721, 1997.

[7] Dimitris Berberidis and Georgios B. Giannakis. Node Embedding with Adaptive Similarities for Scalable Learning over Graphs. *CoRR*, 1811.10797, 2018.

[8] Dimitris Berberidis, Athanasios N. Nikolakopoulos, and Georgios B. Giannakis. Adaptive diffusions for scalable learning over graphs. *IEEE Transactions on Signal Processing*, 67(5):1307–1321, 2019.

[9] Filippo Maria Bianchi, Daniele Grattarola, Lorenzo Livi, and Cesare Alippi. Graph Neural Networks with convolutional ARMA filters. *CoRR*, 1901.01343, 2019.

[10] Aleksandar Bojchevski and Stephan Günnemann. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. *ICLR*, 2018.

[11] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral Networks and Deep Locally Connected Networks on Graphs. In *ICLR*, 2014.

[12] Eliav Buchnik and Edith Cohen. Bootstrapped Graph Diffusions: Exposing the Power of Nonlinearity. *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, 2(1):1–19, 2018.

[13] Siheng Chen, Aliaksei Sandryhaila, Jose M. F. Moura, and Jelena Kovacevic. Adaptive graph filtering: Multiresolution classification on graphs. In *IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2013.

[14] F. Chung. The heat kernel as the pagerank of a graph. *Proceedings of the National Academy of Sciences*, 104(50):19735–19740, 2007.

[15] Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. Inference and phase transitions in the detection of modules in sparse networks. *Physical Review Letters*, 107(6):065701, 2011.

[16] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *NIPS*, 2016.

[17] Tyler Derr, Yao Ma, and Jiliang Tang. Signed Graph Convolutional Networks. In *ICDM*, 2018.

[18] Paul D. Dobson and Andrew J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology*, 330(4):771–783, 2003.

[19] David K. Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *NIPS*, 2015.

[20] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *LREC 2010 Workshop on New Challenges for NLP Frameworks*, 2010.

[21] Matthias Fey and Jan E. Lenssen. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR workshop*, 2019.

[22] François Fouss, Kevin Francoisse, Luh Yen, Alain Pirotte, and Marco Saerens. An experimental investigation of kernels on graphs for collaborative recommendation and semisupervised classification. *Neural Networks*, 31:53–72, 2012.

[23] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In *ICML*, 2017.

[24] M. Gori, G. Monfardini, and F. Scarselli. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks*, 2005.

[25] Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *KDD*, 2016.

[26] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs. In *NIPS*, 2017.

[27] David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval. Wavelets on graphs via spectral graph theory. *Applied and Computational Harmonic Analysis*, 30(2):129–150, 2011.

[28] Mikael Henaff, Joan Bruna, and Yann LeCun. Deep Convolutional Networks on Graph-Structured Data. *CoRR*, 1506.05163, 2015.

[29] Eric Jones, Travis Oliphant, Pearu Peterson, and others. *SciPy: Open source scientific tools for Python*. 2001.

[30] Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011.

[31] Thomas N. Kipf and Max Welling. Variational Graph Auto-Encoders. In *NIPS workshop*, 2016.

[32] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*, 2017.

[33] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then Propagate: Graph Neural Networks Meet Personalized PageRank. In *ICLR*, 2019.

[34] Kyle Kloster and David F Gleich. Heat kernel based community detection. In *KDD*, 2014.

[35] Isabel M. Kloumann, Johan Ugander, and Jon Kleinberg. Block models and personalized PageRank. *Proceedings of the National Academy of Sciences*, 114(1):33–38, 2017.

[36] Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete structures. In *ICML*, 2002.

[37] Stéphane Lafon and Ann B. Lee. Diffusion Maps and Coarse-Graining: A Unified Framework for Dimensionality Reduction, Graph Partitioning, and Data Set Parameterization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(9):1393–1403, 2006.

[38] Edmund Landau. Zur relativen Wertbemessung der Turnierresultate. *Deutsches Wochenschach*, 11: 366–369, 1895.

[39] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *KDD*, 2005.

[40] Pan Li, Eli Chien, and Olgica Milenkovic. Optimizing generalized pagerank methods for seed-expansion community detection. In *NeurIPS*, 2019.

[41] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive Graph Convolutional Neural Networks. In *AAAI*, 2018.

[42] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *ICLR*, 2018.

[43] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated Graph Sequence Neural Networks. In *ICLR*, 2016.

[44] Jeremy Ma, Weiyu Huang, Santiago Segarra, and Alejandro Ribeiro. Diffusion filtering of graph signals and its use in recommendation systems. In *ICASSP*, 2016.

[45] Zheng Ma, Ming Li, and Yuguang Wang. PAN: Path Integral Based Convolution for Deep Graph Neural Networks. In *ICML workshop*, 2019.

[46] Naoki Masuda, Mason A Porter, and Renaud Lambiotte. Random walks and diffusion on networks. *Physics reports*, 716:1–58, 2017.

[47] Julian J. McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-Based Recommendations on Styles and Substitutes. In *SIGIR*, 2015.

[48] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.

[49] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

[50] Jörg Menche, Amitabh Sharma, Maksim Kitsak, Susan Ghiassian, Marc Vidal, Joseph Loscalzo, and Albert-László Barabási. Uncovering disease-disease relationships through the incomplete human interactome. *Science*, 347(6224):1257601, 2015.

[51] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*, 2013.

[52] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M. Bronstein. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *CVPR*, 2017.

[53] Galileo Namata, Ben London, Lise Getoor, and Bert Huang. Query-driven Active Surveying for Collective Classification. In *International Workshop on Mining and Learning with Graphs (MLG), KDD*, 2012.

[54] Huda Nassar, Kyle Kloster, and David F. Gleich. Strong Localization in Personalized PageRank Vectors. In *International Workshop on Algorithms and Models for the Web Graph (WAW)*, 2015.

[55] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On Spectral Clustering: Analysis and an algorithm. In *NIPS*, 2002.

[56] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. In *ICML*, 2016.

[57] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Report, Stanford InfoLab, 1998.

[58] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS workshop*, 2017.

[59] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[60] Tiago P. Peixoto. The graph-tool python library. *figshare*, 2014.

[61] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: online learning of social representations. In *KDD*, 2014.

[62] Trang Pham, Truyen Tran, Dinh Q. Phung, and Svetha Venkatesh. Column Networks for Collective Classification. In *AAAI*, 2017.

[63] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *WSDM*, 2018.

[64] Stephen Ragain. Community Detection via Discriminant functions for Random Walks in the degree-corrected Stochastic Block Model. Report, Stanford University, 2017.

[65] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[66] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective Classification in Network Data. *AI Magazine*, 29(3):93–106, 2008.

[67] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of Graph Neural Network Evaluation. In *NIPS workshop*, 2018.

[68] A. Sperduti and A. Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[69] Gilbert Wright Stewart and Ji-guang Sun. *Matrix Perturbation Theory*. Computer Science and Scientific Computing. 1990.

[70] Yu-Hang Tang, Dongkun Zhang, and George Em Karniadakis. An atomistic fingerprint algorithm for learning ab initio molecular force fields. *The Journal of Chemical Physics*, 148(3):034101, 2018.

[71] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. VERSE: Versatile Graph Embeddings from Similarity Measures. In *WWW*, 2018.

[72] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[73] Petar Velickovic, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R. Devon Hjelm. Deep Graph Infomax. In *ICLR*, 2019.

[74] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *ICLR*, 2018.

[75] Sebastiano Vigna. Spectral ranking. *Network Science, CoRR (updated, 0912.0238v15)*, 4(4):433–445, 2016.

[76] Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[77] Zhewei Wei, Xiaodong He, Xiaokui Xiao, Sibo Wang, Shuo Shang, and Ji-Rong Wen. TopPPR: Top-k Personalized PageRank Queries with Precision Guarantees on Large Graphs. In *SIGMOD*, 2018.

[78] Felix Wu, Tianyi Zhang, Amauri Holanda de Souza Jr., Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying Graph Convolutional Networks. In *ICML*, 2019.

[79] Bingbing Xu, Huawei Shen, Qi Cao, Keting Cen, and Xueqi Cheng. Graph Convolutional Networks using Heat Kernel for Semi-supervised Learning. In *IJCAI*, 2019.

[80] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*, 2018.

[81] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *ICLR*, 2019.

[82] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *KDD*, 2018.