Learning Adaptive Heat Kernel Convolution on Graphs and Its Approximations

Anonymous CVPR submission

Paper ID 11206

Abstract

Various Graph Neural Networks (GNN) have been successful in analyzing features represented in non-Euclidean spaces, however, it has also shown many limitations such as oversmoothing, i.e., information becomes too averaged with an increase in number of hidden layers. The issue stems from the intrinsic formulation of Graph Convolution where the features are aggregated only from a direct neighborhood per layer uniformly across the entire nodes in the graph. As setting different number of hidden layers per node is infeasible, recent works leverage a diffusion kernel to redefine the graph structure and incorporate information from farther nodes. However, such approaches require eigendecomposition of a graph Laplacian which is computationally challenging especially when dealing with a large graph. In this regards, we propose a diffusion learning framework where the range of feature aggregation is defined by a convolution operation with a heat kernel controlled by scale. To make the framework light and efficient, we leverage various types of polynomial approximations of the convolution and derive derivatives of the expansion coefficients with respect to the scale in a closed-form. With a downstream classifier, the entire framework is made trainable in an end-to-end manner, and it is tested on with various independent datasets for two different tasks (i.e., node classification and graph classification) to demonstrate feasibility of our ideas.

1. Introduction

Graph Neural Network (GNN) has been heavily recognized in machine learning and computer vision with various practical applications such as text classification [16], neural machine translation [2], 3D shape analysis [35, 36], semantic segmentation [30, 39], social information systems [23] and speech recognition [25]. At the heart of these GNN models lies the graph convolution, which develops useful representation of each node with a filtering operation on the graph. Given a graph comprised of a set of nodes/edges and signal defined on its nodes, in [19], a convolutional layer was introduced as a linear combination of the signal within a direct neighborhood of each node using the topology of the graph, and its equivalence with spectral filtering [13] has been shown. Stacking these convolutional layers (together with non-linear activations) constitutes the fundamental Graph Convolutional Network (GCN) [19], and testing a newly developed GCN on classifying node-wise labels has become a standard benchmark to validate the GCN model [3, 4, 37, 40, 41].

Notice that there is an intrinsic issue with the architecture of conventional GCNs: each convolution layer gathers information within a direct neighborhood *uniformly* across all nodes. When information aggregation from direct neighborhood is not sufficient, the convolution layers are stacked to seek for useful information from a larger neighborhood. Such an architecture with several convolution layers broadens the range of neighborhood for information aggregation *uniformly*, again, across the entire nodes in the graph. Eventually, when the same information is shared across all the nodes, it leads to "oversmoothed" representation of each node not being able to characterize one from another. This behavior can be easily interpreted from the spectral perspective, as a filtering operation in the spectral space will uniformly affect all nodes in the graph space.

Perhaps the most intuitive solution to the problem above would be to use different ranges of neighborhood per node. However, it is difficult to design such a model with conventional graph convolution, as it would require different number of convolution layers for each node which is highly impractical. Several recent works tried to overcome this locality issue of the graph convolution. Methods in [18, 34] leverage attention mechanism to capture long-range relationships among nodes, [12, 38] develop pooling scheme to compress an overall graph, and [4] improve upon the vanila GCN with skip connection of residuals as in ResNet [14].

Notably, GraphHeat [40] used a diffusion kernel , i.e., heat-kernel, to simply define distances between nodes as a heat diffusion process along the graph structure, without any complicated modification of the vanilla graph convolution. It redefined connections between nodes according to a specific "scale" as a probability of a random walker traveling from one node to another within some time denoted as the scale. It easily connects many local nodes within

a range (i.e., scale) even though they are not directly connected. Such an approach is quite effective when the homophily condition is reasonably held even if the given edges in a graph may be noisy. Still, the scale was defined as a user parameter and the same range was arranged across the entire graph. Later, a framework that train on the scale according to a target task was introduced with the gradient of a loss with respect to the scale for each node was defined in a closed-form. While shown to be quite effective, such an approach is computationally inefficient with heavy diagonalization of graph Laplacian, especially when dealing with a large or a population of graphs.

In this regime, we propose an efficient framework that learns adaptive scales for each node of a graph with approximations, i.e., **A**daptive **S**cale via **AP**proximation (ASAP). The key idea is to train on the node-wise *range of neighborhood* instead of stacking layers of convolution per node. For this, we first show that the formulation in [40] can be redefined as a heat kernel convolution, which can be approximated with various polynomials, i.e., Chebyshev, Hermite and Lagurre, as in [17]. Then, we derive the derivatives of the expansion coefficients of these polynomials in the scale space, and use it to define a task-specific gradient to train the scale within the approximation. While the existing graph convolution with the first order Chebyshev polynomial approximation is capable of dealing with only 1-hop distance range of a given graph structure, our model let each node achieve better representation leveraging node features from farther nodes within a trained "range" defined by the heat kernel. The ideas above lead to the following contributions:

- We propose a novel diffusion convolution layer with polynomial approximation that aggregates feature within a range of neighbors instead of hop distances,
- We derive closed-form derivatives of the polynomial coefficients with respect to the range (i.e., scale) so that the range can be trained via gradient-descent,
- We develop a unified graph analysis framework efficiently trainable in an end-to-end manner and validate it on two separate tasks on independent datasets.

ASAP demonstrates superior results on the node classification task in a semi-supervised learning setting [32], as well as on a graph classification task performed on a population of brain networks to predict diagnostic labels for Alzheimer's Disease (AD). Especially on the brain network experiment, the trained scales delineates specific regions highly responsible for the prediction of AD, providing interpretability of the training process.

2. Related Work

GNNs typically learn novel representation of individual nodes by aggregating information from its neighbors via message passing. The vanila GCN [19] and Variant GNNs utilize graph convolution to perform feature aggregation from neighbors. Simplifying Graph Convolution (SGC) [37] captures high-order information in the graph with the K-th power of the graph convolution matrix, GCNII [4] extends a vanilla GCN model residual connection and identity mapping [14], and Graph Diffusion Convolution (GDC) [21] introduces spatially localized graph convolution to aggregate information of indirect nodes. Graph Attention Network (GAT) [34] introduced attention mechanism on graphs to assign relationships to different nodes.

To overcome oversmoothing issue from convolution operation, Personalized Propagation of Neural Prediction and its Approximation (APPNP) [20] improved message propagation scheme based on personalized PageRank [29], Graph Random Neural Network (GRAND) [10] developed graph data augmentation, Deep Adaptive Graph Neural Network (DAGNN) [24] disentangled representation transform and message propagation to construct a deep model.

While the methods introduced above were mainly evaluated on semi-supervised learning for node classification, notice that many of these methods can be adopted for graph classification as well [21, 34] with an additional layer transforming node embeddings to a graph embedding. Other literature, although not fully discussed in this section, will be introduced and used as baselines to compare the performances with our proposed model for both node classification and graph classification tasks in Section 5.

3. Preliminaries

We review convolution on graphs with heat kernel and its approximation as they set cornerstones of ASAP .

Graph Notation. $G = \{V, E\}$ denotes an undirected graph, where V is a set of nodes with $|V| = N$ and E is an edge set. A graph G is often represented as a symmetric adjacency matrix A of which individual elements a_{pq} encodes connectivity information between node p and q. A graph Laplacian is defined as $L = D - A$ where D is a degree matrix, i.e., a diagonal matrix with $D_{pp} = \sum_q A_{pq}$. A normalized graph Laplacian normalized on the degree is defined as $\hat{L} = I_n - D^{-1/2}AD^{-1/2}$ where I_n is an identity matrix. Since $\hat{L}$ is real symmetric, it has a complete set of orthonormal basis $U = [u_1|u_2|...|u_n]$ known as Laplacian eigenvectors and corresponding real and non-negative eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_N$.

Heat Kernel Convolution. To capture smooth transition of node characteristics among different nodes as a diffusion process, a heat kernel is designed as

$$f(\lambda_i) = e^{-s\lambda_i} \tag{1}$$

where λ_i is the i-th eigenvalue of the graph Laplacian and s controls the time/scale of heat diffusion. In [6], the heat kernel between nodes p and q of a graph G is defined in the

spectral graph domain spanned by U as

$$h_s(p,q) = \sum_{i=1}^{N} e^{-s\lambda_i} u_i(p) u_i(q) \tag{2}$$

where u_i is i-th eigenvector of the graph Laplacian. Using convolutional theorem [28], graph Fourier transform, i.e., $\hat{x} = U^T x$, offers a way to define the graph convolution operation $*$ of a signal $x(p)$ with a filter h_s. Using Eq. (2), heat kernel convolution with h_s as a low-pass filter is defined as

$$h_s * x(p) = \sum_{i=1}^{N} e^{-s\lambda_i} \hat{x}(i) u_i(p) \tag{3}$$

whose band-width is controlled by s.

Polynomial Approximation of Heat Kernel Convolution. The exact computation of Eq. (3) requires diagonalization of a graph Laplacian which is computationally challenging. Existing literature use Chebyshev polynomial as a basis to approximate the convolution operation as a linear transform [8, 15]. In [17], approximation of heat kernel convolution was introduced using several orthogonal polynomials [5, 27] such as Chebyshev, Hermite and Laguerre. The analytic solutions to the polynomial coefficients $c_{s,n}$ for scale s and degree n were derived for Chebyshev polynomial P_n^T, Hermite polynomial P_n^H and Laguerre polynomial as P_n^L where n denotes the degree of each polynomial.

A polynomial $P_n \in \{P_n^T, P_n^H, P_n^L\}$ is often defined by a second order recurrence as

$$P_{n+1}(\lambda) = (\alpha_n \lambda + \beta_n) P_n(\lambda) + \gamma_n P_{n-1}(\lambda) \tag{4}$$

where initial conditions $P_{-1}(\lambda) = 0$ and $P_0(\lambda) = 1$ for $n \geq 0$ and parameters α_n, β_n and γ_n determine the type of polynomial. Then, the heat kernel $e^{-s\lambda}$ and polynomial expansion coefficients $c_{s,n}$ can be defined with P_n as

$$e^{-s\lambda} = \sum_{n=0}^{\infty} c_{s,n} P_n(\lambda), \tag{5}$$

$$c_{s,n} = \int_a^b e^{-s\lambda} P_n(\lambda) w(\lambda) d\lambda \tag{6}$$

where $w(\lambda)$ differs across polynomials to make them orthonormal within $[a, b]$, i.e., $\int_a^b P_n(\lambda) P_k(\lambda) w(\lambda) d\lambda = \delta_{nk}$.

Now, the solution to the heat diffusion (3) can be expressed in terms of P_n and $c_{s,n}$ via Eq. (5) and (6) as

$$h_s * x(p) = \sum_{n=0}^{\infty} c_{s,n} \sum_{j=0}^{\infty} P_n(\lambda_j) \hat{x}(j) u_j(p). \tag{7}$$

Since $\hat{L} u_j = \lambda_j u_j$, the heat diffusion in Eq. (7) can be further written as

$$h_s * x(p) = \sum_{n=0}^{\infty} c_{s,n} P_n(\hat{L}) x(p) \tag{8}$$

where initial conditions $P_{-1}(\hat{L})x(p) = 0$ and $P_0(\hat{L})x(p) = x(p)$ from the second order recurrence. Notice that Eq. (8) represents the convolution operation as a simple linear combination of $c_{s,n}$ and P_n without u_j, and it is approximated at the order of m for practical purposes.

4. Learning Adaptive Scales via Polynomial Approximation

We introduce our model, i.e., ASAP, that "learn" the optimal range of neighborhood for individual graph nodes based on a specific task. The derivatives to train s via backpropagation for two separate tasks, i.e., node classification and graph classification, are derived in their closed-forms.

4.1. Model Architecture

In most of existing GCN frameworks [19, 41], the convolution operation at the k-th layer is given as

$$H_k = \sigma_k(\tilde{A} H_{k-1} W_k) \tag{9}$$

where $\tilde{A}$ is a normalized adjacency matrix, W_k is a trainable weight matrix and σ_k is a non-linear activation function, and it takes an input H_{k-1} and outputs a new representation H_k. Each convolution operation takes features from a direct neighbor of each node according to $\tilde{A}$ to design the new node representation. As more convolution layers are stacked, the range of aggregation is uniformly increased for all nodes causing the infamous "oversmoothing" issue. To alleviate this issue, we propose to utilize a diffusion kernel, i.e., a heat kernel, on the graph to define the individual ranges of neighborhood for the nodes, so that the node-wise range is adaptively defined to avoid oversmoothing.

The architecture of ASAP is given in Fig. 1. The overall components are similar to the original GCN [19], however, ASAP redefines the convolution with approximated heat kernel operation. Consider a graph G given as a Laplacian $\hat{L}$, feature X defined on its nodes and either node-wise or graph-wise label Y. Our framework takes $\hat{L}$ and X as inputs, performs approximated heat kernel convolutions and outputs a prediction $\hat{Y}$. The $\hat{Y}$ is then compared with the ground truth Y, and the error is backpropagated to update model parameters including the scale s.

Convolution Layer. Based on Eq. (8), Eq. (9) is redefined by replacing the normalized adjacency matrix $\tilde{A}$ with the heat kernel with polynomial approximation as

$$H_k = \sigma_k([\sum_{n=0}^{m} c_{s,n} P_n(\hat{L})] H_{k-1} W_k). \tag{10}$$

While $\tilde{A}$ let the model combine information from nodes within 1-hop distance only, the formulation in Eq. (10) let it aggregate information within a "range" of each node defined by s embedded in $c_{s,n}$. Approximating the convolution with the polynomials $P_n(\hat{L})$ let us avoid heavy diagonalization of exact heat kernel convolution which involves eigendecomposition of the $\hat{L}$. The Eq. (10) is an operation in a single convolution layer, and we can stack K of them to achieve better representation of the original X.

Output Layer. The output layer using softmax function yields $\hat{Y}$ which is a prediction of Y. Depending on the task, it may include a simple Multi-layer Perceptron (MLP) that can be trained. A loss L_{err} is computed at this layer which

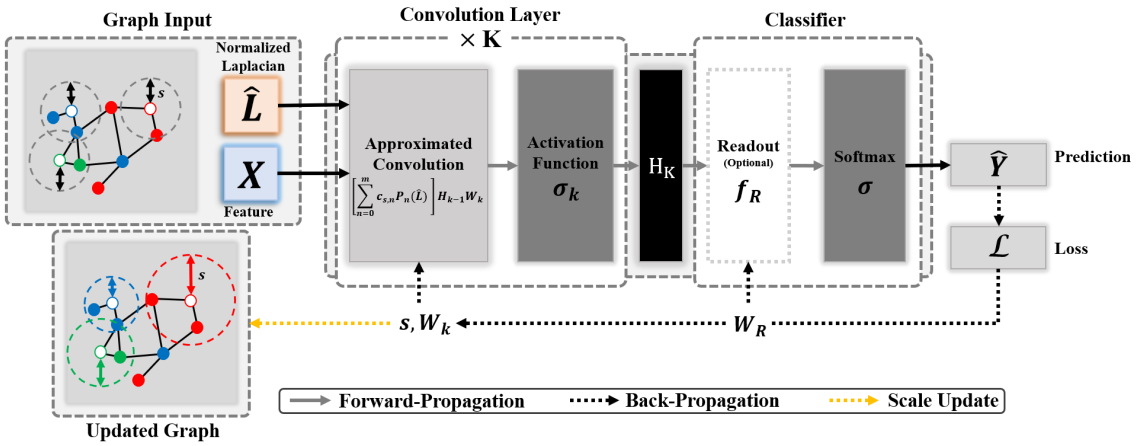

Figure 1. Illustration of ASAP. A graph (as normalized Laplacian $\hat{L}$) and node features X are inputted to the convolution layer via approximation. The output H_K from convolution layers is inputted to a downstream classifier which yields prediction $\hat{Y}$. The loss from $\hat{Y}$ is backpropagated to update the classifier and convolution approximation with s to adaptively adjust the scale of each node.

quantifies the error between $\hat{Y}$ and Y. The loss $L_{\mathbf{err}}$ is defined according to a target task, e.g., node classification or graph classification. More details are given when we describe different graph analysis tasks in Section 4.3 and 4.4.

Model Update. The loss $L_{\mathbf{err}}$ is back-propagated to update the model parameters, i.e., W_k and s. To prevent the value of scale from becoming negative, a regularization $\mathcal{L}_s$ is imposed to define an overall objective function $\mathcal{L}$ as

$$\mathcal{L} = \mathcal{L}_{\mathbf{err}} + \alpha \mathcal{L}_s \tag{11}$$

where α is a hyperparameter. The weight W_k can be easily trained with backpropagation. If there exist a gradient on scale s, a multi-variate $\mathbf{s}$ across all nodes can be also trained with the backpropagation as

$$\mathbf{s} \leftarrow \mathbf{s} - \beta_s \frac{\partial \mathcal{L}}{\partial \mathbf{s}} \tag{12}$$

where β_s is a learning rate. It requires $\frac{\partial \mathcal{L}}{\partial \mathbf{s}}$ to make the framework trainable in an end-to-end manner. In the following section, we derive this gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{s}}$ in a closed-form to approximate the heat kernel convolution in Eq. (10).

4.2. Gradients of Polynomial Coefficients with Scale

We denote expansion coefficient as $c^T_{s,n}$, $c^H_{s,n}$ and $c^L_{s,n}$ that correspond to P^T_n, P^H_n and P^L_n. As introduced in [17], one can obtain a solution to the heat diffusion by obtaining expansion coefficient with each of the polynomials in P_n. In order to design a gradient-based "learning" framework of node-wise range (i.e., scale) based on these expansions, we derived gradients of loss $\frac{\partial \mathcal{L}}{\partial s}$ in a closed-form. This is an essential component of ASAP as it let the model efficiently train without diagonalization of $\hat{L}$. The $\frac{\partial \mathcal{L}}{\partial s}$ can be achieved using the chain rule in a traditional way, and to obtain $\frac{\partial \mathcal{L}}{\partial s}$ in terms of the H_k, we derive $\frac{\partial c_{s,n}}{\partial s}$ as follows.

Chebyshev Polynomial. The recurrence relation and expansion coefficient for Chebyshev polynomial are given as

$$\begin{aligned} P^T_{n+1}(\hat{L}) &= (2-\delta_{n0})\hat{L}P^T_n(\hat{L}) - P^T_{n-1}(\hat{L}), \\ c^T_{s,n} &= (2-\delta_{n0})(-1)^n e^{-\frac{sb}{2}} I_n\left(\frac{sb}{2}\right) \end{aligned} \tag{13}$$

where $b > 0$ is a hyper-parameter, and I_n is the *modified Bessel function of the first kind* [27]. The derivative of $c^T_{s,n}$ with respect to s is obtained as

$$\frac{\partial c^T_{s,n}}{\partial s} = (-1)^n b e^{\frac{-sb}{2}} \left(I_{n-1}\left(\frac{sb}{2}\right) - \left(\frac{2n}{sb}+1\right) I_n\left(\frac{sb}{2}\right)\right). \tag{14}$$

Hermite Polynomial. The recurrence relation and expansion coefficient for Hermite polynomial are written as

$$\begin{aligned} P^H_{n+1}(\hat{L}) &= 2\hat{L}P^H_n(\hat{L}) - 2nP^H_{n-1}(\hat{L}), \\ c^H_{s,n} &= \frac{1}{n!}\left(\frac{-s}{2}\right)^n e^{\frac{s^2}{4}}, \end{aligned} \tag{15}$$

and derivative of $c^H_{s,n}$ with respect to s is given as

$$\frac{\partial c^H_{s,n}}{\partial s} = \frac{s e^{\frac{s^2}{4}}}{2n!}\left(\frac{-s}{2}\right)^n \left(\frac{2n}{s^2}+1\right). \tag{16}$$

Laguerre Polynomial. For Laguerre polynomial, the recurrence relation and expansion coefficient are

$$\begin{aligned} P^L_{n+1}(\hat{L}) &= \frac{(2n+1-\hat{L})P^L_n(\hat{L}) - nP^L_{n-1}(\hat{L})}{n+1}, \\ c^L_{s,n} &= \frac{s^n}{(s+1)^{n+1}}, \end{aligned} \tag{17}$$

and the derivative of $c^L_{s,n}$ with respect to s can be derived as

$$\frac{\partial c^L_{s,n}}{\partial s} = \frac{s^{n-1}(n-s)}{(s+1)^{n+2}}. \tag{18}$$

The $\frac{\partial c_{s,n}}{\partial s}$ derived above will be used to define $\frac{\partial \mathcal{L}}{\partial s}$ which differs depending on the formulation of $\mathcal{L}$. In the following sections, we introduce node classification and graph classification tasks and show how $\frac{\partial \mathcal{L}}{\partial s}$ can be defined using $\frac{\partial c_{s,n}}{\partial s}$ when the $\mathcal{L}$ is given as a typical cross-entropy.

4.3. Node Classification

The goal of node classification is to predict labels of unlabeled nodes based on information from other nodes. The output layer (after K-convolution layers) of ASAP produces a prediction $\hat{Y} = \sigma(H_K)$, and the training should be performed to reduce the error between $\hat{Y}$ and the true Y. The error is often defined as cross-entropy, and the loss $\mathcal{L}_{\textbf{err}}$ across all labeled nodes V^L is defined as

$$\mathcal{L}_{\textbf{err}} = -\frac{1}{N}\sum_{i\in V^L}\sum_{j=1}^{J} Y_{ij}\ln\hat{Y}_{ij} \tag{19}$$

where N is the number of nodes and J is the number of labels. Y_{ij} is 1 if node $v_i \in V^L$ has a label, otherwise 0, and it is compared with the prediction $\hat{Y}_{ij}$.

To find a direction in the scale space to reduce $\mathcal{L}_{\textbf{err}}$, we calculate the derivative of $\mathcal{L}$ in terms of scale s of $c_{s,n}$ as

$$\frac{\partial\mathcal{L}_{\textbf{err}}}{\partial s} = \frac{\partial\mathcal{L}_{\textbf{err}}}{\partial H_K}\frac{\partial H_K}{\partial c_{s,n}}\frac{\partial c_{s,n}}{\partial s}. \tag{20}$$

For $\frac{\partial\mathcal{L}_{\textbf{err}}}{\partial s}$, as we use softmax to produce pseudo-probability, the derivative of $\mathcal{L}_{\textbf{err}}$ with respect to H_K is derived as

$$\frac{\partial\mathcal{L}_{\textbf{err}}}{\partial H_K} = \hat{Y} - Y. \tag{21}$$

As the $c_{s,n}$ is embedded in every approximated convolution layer, the derivative of H_k with respect to $c_{s,n}$ becomes

$$\begin{aligned}\frac{\partial H_K}{\partial c_{s,n}} =& \sigma_k^{'}([\sum_{n=0}^{m} c_{s,n}P_n(\hat{L})]H_{k-1}W_k) \times W_k^{\mathsf{T}} \\ &\times (\sum_{n=0}^{m} P_n(\hat{L})H_{k-1}^{\mathsf{T}} + [\sum_{n=0}^{m} c_{s,n}P_n(\hat{L})]\frac{\partial H_{k-1}}{\partial c_{s,n}})\end{aligned} \tag{22}$$

where the derivative of H_k is recursively defined with H_{k-1} along the hidden layer.

Using Eq. (21), (22) and $\frac{\partial c_{s,n}}{\partial s}$ that we defined in Section 4.2, $\frac{\partial\mathcal{L}_{\textbf{err}}}{\partial s}$ can be obtained via chain rule. The $\frac{\partial c_{s,n}}{\partial s}$ depends on the type of polynomial (Eq. (14), (16) or (18)). Finally, the gradient on $\mathcal{L}_{\textbf{err}}$ for node classification is written as

$$\begin{aligned}\frac{\partial\mathcal{L}_{\textbf{err}}}{\partial s} =& (\hat{Y} - Y) \times \sigma_k^{'}([\sum_{n=0}^{m} c_{s,n}P_n(\hat{L})]H_{k-1}W_k) \times W_k^{\mathsf{T}} \\ &\times (\sum_{n=0}^{m} P_n(\hat{L})H_{k-1}^{\mathsf{T}} + [\sum_{n=0}^{m} c_{s,n}P_n(\hat{L})]\frac{\partial H_{k-1}}{\partial c_{s,n}}) \times \frac{\partial c_{s,n}}{\partial s}.\end{aligned} \tag{23}$$

ASAP back-propagates the error to update scale s with Eq. (23) to obtain the optimal range of scale for each node.

4.4. Graph Classification

Consider a population of graphs $\{G_t\}_{t=1}^{T}$ with corresponding labels $\{Y_t\}_{t=1}^{T}$, and learning a graph classification model finds a function $f(G_t) = Y_t$. For this, the output layer of ASAP consists of a readout layer $f_R(\cdot)$ (i.e., MLP with ReLU) and a softmax $\sigma(\cdot)$ at the end to construct pseudo-probability for each class. The $f_R(\cdot)$ with weights W_R takes the output H_K from the convolution layers as an input and returns h_G as

$$h_G = f_R(H_K; W_R) \tag{24}$$

and $\hat{Y}$ is computed as $\hat{Y} = \sigma(h_G)$.

As in Section 4.3, the loss $\mathcal{L}_{\textbf{err}}$ for graph classification is defined with cross-entropy as

$$\mathcal{L}_{\textbf{err}} = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j\in J} Y_{tj}\ln\hat{Y}_{tj} \tag{25}$$

where T is sample size, J is the set of class labels. Y_{tj} is one-hot encoded ground truth and compared with the $\hat{Y}_{tj}$.

As in Section 4.3, the gradient of $\mathcal{L}_{\textbf{err}}$ along s is computed via chain rule. As we use an additional MLP for graph classification, i.e., Eq. (24), the gradient of $\mathcal{L}_{\textbf{err}}$ along h_G must be computed and plugged into Eq. (23) as

$$\frac{\partial\mathcal{L}_{\textbf{err}}}{\partial s} = \frac{\partial\mathcal{L}_{\textbf{err}}}{\partial h_G}\frac{\partial h_G}{\partial H_K}\frac{\partial H_K}{\partial c_{s,n}}\frac{\partial c_{s,n}}{\partial s}. \tag{26}$$

Since the activation function of output layer is a softmax, the derivative of $\mathcal{L}_{\textbf{err}}$ with respect to h_G is same as Eq. (21), and the derivative of H_K with respect to $c_{s,n}$ is also the same as Eq. (22). The derivative of h_G with respect to H_k varies depending on the choice of h_G. We have already derived the derivatives of $c_{s,n}$ with respect to s as in Eq. (14), (16), (18) for different types of polynomials. Thus, the final gradient on $\mathcal{L}_{\textbf{err}}$ for graph classification is computed as a product of these partial gradients as

$$\begin{aligned}\frac{\partial\mathcal{L}_{\textbf{err}}}{\partial s} =& (\hat{Y} - Y) \times h_G^{'} \times \sigma_k^{'}([\sum_{n=0}^{m} c_{s,n}P_n(\hat{L})]H_{k-1}W_k) \times W_k^{\mathsf{T}} \\ &\times (\sum_{n=0}^{m} P_n(\hat{L})H_{k-1}^{\mathsf{T}} + [\sum_{n=0}^{m} c_{s,n}P_n(\hat{L})]\frac{\partial H_{k-1}}{\partial c_{s,n}}) \times \frac{\partial c_{s,n}}{\partial s}\end{aligned} \tag{27}$$

where $h_G^{'}$ denotes the derivative of h_G with respect to H_K, and the $h_G^{'}$ for the MLP used within our framework is given in the supplementary. Based on Eq. (27), ASAP updates the scale s adaptively for each node in a graph using Eq. (12).

4.5. Regularization on the Scale

In this section, we impose a regularization on the scale s in Eq. (11). To maintain $s > 0$, the regularization $\mathcal{L}_s$ is given as an absolute value on s as

$$\mathcal{L}_s = |s|[s < 0] \tag{28}$$

which manipulates negative s to have positive value.

To obtain the overall gradient of $\mathcal{L}$ with respect to s, the derivative of $\mathcal{L}_s$ is derived as

$$\frac{\partial\mathcal{L}_s}{\partial s} = -1[s < 0]. \tag{29}$$

Depending on the task, we can compute $\frac{\partial\mathcal{L}}{\partial s}$ as a combination of Eq. (29) with Eq. (23) or (27). Now, ASAP becomes trainable in an end-to-end manner to obtain the optimal model parameters including s.

5. Experiments

In this section, we compare the performances of our model and baselines on node classification and graph classification. **ASAP-C**, **ASAP-H** and **ASAP-L** corresponds to

approximations with P_n^T, P_n^H and P_n^L, and the model with exact computation of the heat kernel convolution is referred as **Exact**. For the node classification, we performed experiments on the conventional benchmarks for semi-supervised learning task [32], and as for the graph classification, we investigate brain network data from Alzheimer's Disease Neuroimaging Initiative (ADNI) to classify different stages towards Alzheimer's Disease (AD). These two experiments on separate tasks will validate the feasibility of ASAP on both efficacy and efficiency.

5.1. Implementation Details

We stacked K=2 heat kernel convolution layers with rectified linear unit (ReLU) as the activation function and softmax function at the output to predict the node or graph classes. The initial scale for every node was set to 2. The n=20 for ASAP-C and ASAP-L, and n=30 for ASAP-H as they empirically demonstrated the best convergence. The same number of hidden nodes in W_k was used within each the dataset, drop-out rate was 0.5, and other hyper-parameters such as learning rate for W_k and s were properly tuned to get the best results (given in the supplementary). For the node classification, we used early stopping to stop training when the validation accuracy stopped increasing. For graph classification, the readout function $f_R(\cdot)$ was defined as MLP with 2 layers with 16 hidden nodes (with ReLU), and 5-fold cross validation (CV) was used.

5.2. Node Classification

Datasets. We conducted experiments on various real-world node classification datasets (in Table 1) that provide connected and undirected graphs. Cora [31], Citeseer [31] and Pubmed [31] are constructed as citation networks. The nodes are papers, the edges are citations of one paper to another, the features are bag-of-words representation of papers, and the labels are specified as an academic topic.

Amazon Computer [32] and Amazon Photo [32] define co-purchase networks. The nodes are goods, the edges indicate whether two goods are frequently purchased together, the node features are bag-of-words representation of product reviews, and the labels are categories of goods. Coauthor CS [32] is a co-authorship network. The nodes are authors, the edges indicate whether two authors co-authored a paper, the features are the keywords from papers, and the labels are each author's field of study.

Setup. We used the accuracy of node classification as the evaluation metric. For Cora, Citseer and Pubmed data,

Table 1. Summary of node classification datasets.

Dataset	Nodes	Edges	Classes	Features
Cora	2,708	5,429	7	1,433
Citeseer	3,327	4,732	6	3,703
Pubmed	19,717	44,338	3	500
Amazon Computers	13,752	245,861	10	767
Amazon Photo	7,650	119,081	8	745
Coauthor CS	18,333	81,894	15	6,805

Table 2. Node classification accuracy (%) on Cora, Citeseer, and Pubmed. ASAP yields better performances over existing baselines (in **bold**) similar to Exact achieving the best results (underline).

Model	Cora	Citeseer	Pubmed
GCN [19]	81.50	70.30	78.60
GAT [34]	83.00	72.50	79.00
APPNP [20]	83.30	71.80	80.10
GDC [21]	82.20	71.80	79.10
SGC [37]	81.70	71.30	78.90
Shoestring [22]	81.90	69.50	79.70
GraphHeat [40]	83.70	72.50	80.50
g-U-Nets [12]	84.40	73.20	79.60
GCNII [4]	85.50	73.40	80.30
GRAND [10]	85.40	75.40	82.70
DAGNN [24]	84.40	73.30	80.50
SelfSAGCN [41]	83.80	73.50	80.70
SuperGAT [18]	84.30	72.60	81.70
SEP-N [38]	84.80	72.90	80.20
ASAP-C	**87.90**	**76.50**	83.30
ASAP-H	85.00	76.10	82.60
ASAP-L	85.90	75.90	**84.10**
Exact	88.20	78.10	85.30

fourteen different baselines were used to compare the results for the node classification task as listed in Table 2. These standard benchmarks are provided with fixed split of 20 nodes per class for training, 500 nodes for validation and 1000 nodes for testing as in other literature [18, 38, 41].

For Amazon and Coauthor CS datasets, six baselines are used as in Table 3. For the MLP with 3-layers, GCN and 3ference, results are obtained from [26]. For others, the experiments were performed by randomly splitting the data as 60%/20%/20% for training/validation/testing datasets as in [26] and replicating it 10 times to obtain mean and standard deviation of the evaluation metric.

Results. Table 2 and 3 show the node classification performance comparisons between ASAP and baseline models. As shown in Table 2, on the node classification benchmarks, learning node-wise adaptive scale performs the best in both Exact and its approximations. ASAP showed improved performance over existing models; exceeding previous best baseline performances by 2.4% (on Cora), 1.1% (on Citeseer) and 1.4% (on Pubmed). The performance of

Table 3. Node classification accuracy (%) comparison on Amazon Computers, Amazon Photo, and Coauthor CS. We ran 10 replicated experiments, and mean and s.d. are reported along with and the maximum accuracy of each experiment in parentheses. The best mean accuracy is in **bold**, and the best maximum accuracy among replications are underlined.

Model	Amazon Computer	Amazon Photo	Coauthor CS
MLP (3-layers)	84.63	91.96	95.63
GCN	90.49	93.91	93.32
3ference [26]	90.74	95.05	95.99
GAT [34]	91.18 ± 0.74 (92.51)	94.49 ± 0.54 (95.16)	93.42 ± 0.31 (93.70)
GDC [21]	86.03 ± 2.26 (92.00)	93.28 ± 1.03 (94.57)	92.68 ± 0.53 (93.45)
GraphHeat [40]	89.59 ± 3.15 (92.62)	94.04 ± 0.75 (94.77)	92.93 ± 0.20 (93.29)
ASAP-C	**94.43 ± 1.16** (95.53)	95.96 ± 1.65 (97.25)	**94.81 ± 0.55** (96.37)
ASAP-H	93.64 ± 0.86 (96.01)	**96.65 ± 0.67** (97.38)	93.52 ± 0.97 (96.37)
ASAP-L	92.76 ± 0.48 (93.67)	95.35 ± 0.85 (96.66)	93.58 ± 0.82 (95.91)
Exact	93.52 ± 0.65 (95.34)	96.41 ± 1.54 (98.17)	93.71 ± 1.16 (97.14)

ASAP were quite similar to that of Exact demonstrating feasible approximation. Despite the slight decrease, the time spent for learning adaptive scales using ASAP was much faster, which will be discussed later in Section 5.4.

For additional datasets in Table 3, the performance of ASAP outperformed the baselines. The results for MLP (3-layers), GCN and 3ference were adopted from [26], which reported the best performance out of 10 replicated experiments. We ran the same experiments (i.e., 10 replicates) for GAT, GDC, GraphHeat, Exact and ASAP, and the mean and standard deviation of metrics are given as well as the best ones in parentheses. ASAP shows significant improvements on the Amazon Computer (94.43%, ASAP-C) and Amazon Photo (96.65%, ASAP-H). On the CS Coauther, we also achieve the highest accuracy (94.81%, ASAP-C) in mean as well as the best model performance in parentheses.

5.3. Graph Classification

Datasets. The ADNI study aims for prevention and treatment of AD. Using the images from the ADNI data, each brain was partitioned into 148 cortical surface regions and 12 sub-cortical regions using Destrieux atlas [9], and tractography on diffusion-weighted imaging (DWI) was applied to calculate the white matter fibers connecting the brain regions to construct 160×160 structural network (i.e., graph). On the same parcellation, region-wise imaging features such as Standard Uptake Value Ratio (SUVR) of metabolism level from FDG-PET and cortical thickness from MRI were measured. For the SUVR normalization, Cerebellum was used as the reference. Total of N=1584 subjects were used for the experiment.

Table 4. Demographics of the ADNI dataset.

Category	CN	SMC	EMCI	LMCI	AD
# of subjects	434	187	500	298	165
Gender (Male / Female)	224 / 210	66 / 121	286 / 214	203 / 95	105 / 60
Age (Mean±Std)	73.02±5.86	71.72±5.24	70.96±7.71	71.44±7.30	74.82±8.73

Setup. 5-way classification experiment was designed to classify the 5 different groups in Table 4. 5-fold cross validation was used to obtain robust results, and accuracy, precision and recall in their mean were used as evaluation metrics. As the baseline, we adopted Linear Support Vector Machine (SVM), Multi-Layer Perceptron (MLP) with 2 layers, GCN [19], GAT [34], GDC [21] and GraphHeat [40].

Results. The performance including accuracy, precision and recall between ASAP and six baselines on the ADNI dataset is shown in Table 5. We used two features, i.e., cortical thickness and FDG which are well-known as useful biomarkers for AD, to classify AD specific labels. As written in Table 5, ASAP showed accuracy around 86% on cortical thickness and 90% on FDG in classifying the 5 diagnostic stages of AD, with precision and recall around 0.86 and 0.91, respectively. These numbers approximately the same with results from Exact and low standard deviation of numbers from ASAP demonstrate feasibility of the approximation. ASAP performed the best surpassing the second best methods by 4.61%p and 2.89%p in accuracy for cortical thickness and FDG experiments, respectively.

Table 5. Classification performance comparison (CN / SMC / EMCI / LMCI / AD) on ADNI dataset.

Feature	Model	Classification (ADNI)		
		Accuracy (%)	Precision	Recall
Cortical Thickness	SVM (Linear)	82.39 ± 2.73	0.8223 ± 0.0333	0.8521 ± 0.0250
	MLP (2-layers)	78.76 ± 2.21	0.7916 ± 0.0365	0.7988 ± 0.0261
	GCN [19]	61.37 ± 3.09	0.5975 ± 0.0257	0.6257 ± 0.0441
	GAT [34]	64.17 ± 5.46	62.69 ± 0.0677	0.6676 ± 0.0461
	GDC [21]	77.10 ± 4.25	0.7686 ± 0.0508	0.7848 ± 0.0440
	GraphHeat [40]	70.90 ± 3.17	0.7028 ± 0.0303	0.7177 ± 0.0255
	ASAP-C	**87.00 ± 2.16**	**0.8683 ± 0.0273**	**0.8850 ± 0.0269**
	ASAP-H	85.41 ± 2.32	0.8587 ± 0.0309	0.8668 ± 0.0295
	ASAP-L	85.64 ± 1.86	0.8593 ± 0.0220	0.8659 ± 0.0219
	Exact	86.24 ± 1.96	0.8655 ± 0.0167	0.8673 ± 0.0225
FDG	SVM (Linear)	85.27 ± 2.09	0.8569 ± 0.0268	0.8691 ± 0.0214
	MLP (2-layers)	87.51 ± 1.62	0.8819 ± 0.0239	0.8824 ± 0.0138
	GCN [19]	68.81 ± 1.95	0.6768 ± 0.0280	0.6965 ± 0.0250
	GAT [34]	69.24 ± 7.13	0.6701 ± 0.1015	0.7362 ± 0.0356
	GDC [21]	86.21 ± 3.24	0.8665 ± 0.0327	0.8698 ± 0.0287
	GraphHeat [40]	76.97 ± 2.42	0.7747 ± 0.0349	0.7733 ± 0.0104
	ASAP-C	89.24 ± 2.23	0.8951 ± 0.0216	0.9036 ± 0.0227
	ASAP-H	90.11 ± 2.44	0.9031 ± 0.0270	0.9099 ± 0.0223
	ASAP-L	**90.40 ± 1.38**	**0.9085 ± 0.0182**	**0.9143 ± 0.0153**
	Exact	90.18 ± 2.67	0.9072 ± 0.0276	0.9071 ± 0.0276

5.4. Model Behavior Analysis

Computation Time with Kernel Convolution. Fig. 2 compares averaged empirical time (in *ms*) spent for one epoch of training process of Exact and ASAP on node classification task (on Cora, Citesser and Pubmed) and graph classification task (on ADNI) with 10 replicates. The colors denote the type of methods, and as seen in Fig. 2, ASAP takes far less time than Exact. Notice that the computation of heat kernel convolution takes the majority of time (in black bar), and the approximations make this process efficient. For the node classification, approximation on Pubmed showed the best efficiency as its graph had the largest number of nodes (19717) compared to Cora (2708)

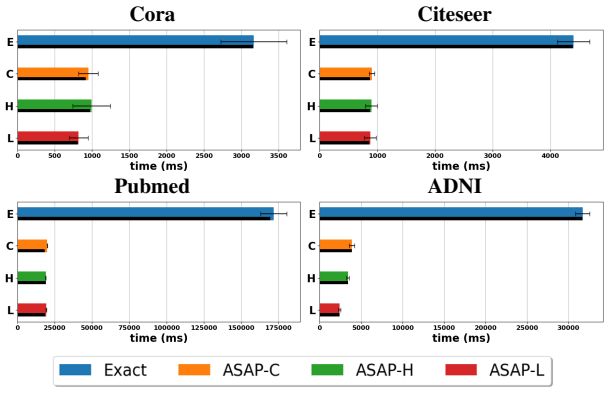

Figure 2. Comparisons of computation time (in *ms*) for one epoch (Forward and back propagation). Within the epoch (in color), computation for heat kernel convolution is given in black bar. Results were obtained from three node classification and a graph classification experiments with 10 repetitions. ASAP saves majority of the computation with approximation.

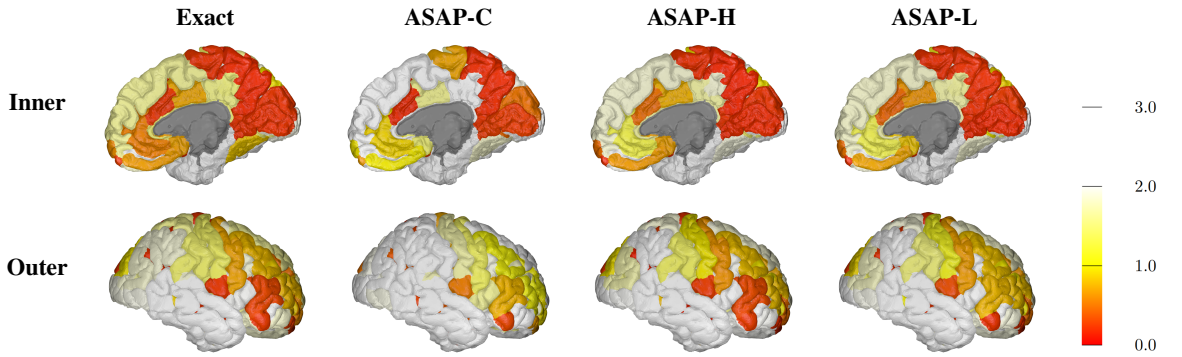

Figure 3. Visualization of the learned scales on the cortical regions of a brain. This visualization shows the scale of each ROI from the classification result using FDG feature. Top: Inner part of right hemisphere, Bottom: Outer part of right hemisphere.

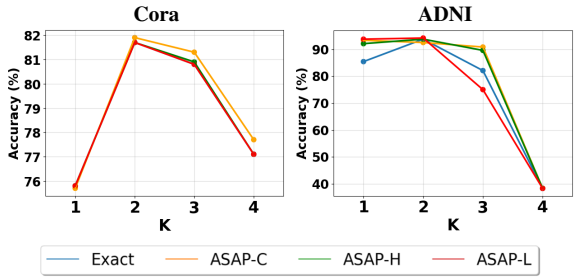

Figure 4. Effect of number of layers K on model performance. Left: accuracy of node classification on Cora, Right: accuracy of graph classification on ADNI. Approximations with ASAP follow the same accuracy patterns with exact convolution computation.

and Citeseer (3327). For the graph classification, approximations were even more efficient as different graphs are given per subject, and eigendecomposition of $\tilde{L}$ from all 1584 subjects had to be performed for Exact. Comparing the computation time of a single epoch on Exact and ASAP-L, the computation time is saved by $\sim$93%.

Interpretation of Scales on Graph Classification. In Section 5.3, we performed classification of graphs (i.e., brain networks) to distinguish different diagnostic labels of AD. The trained classification model yields node-wise optimized scale where the node correspond to specific region of interest (ROI) in the brain. The trained scales denote the optimal ranges of neighborhood for each node. Therefore, if the trained scale is small for a specific node, it means that the node does not have to look far to contribute to the classification. On the other hand, the nodes with large scales need to aggregate information from far distances to constitute an effective embedding as it is not very effective on its own. The trained scales on the brain network classification task with Exact and ASAP are visualized in Fig. 3 conveying two important perspectives. First, the scales delineate which of the ROIs are independently behaving to classify AD-specific labels. Second, the trained scales with ASAP are quite similar to the result from Exact meaning that the approximation is feasibly accurate for practical use.

In Table 6, 10 ROIs with the smallest scales that appear in common across Exact and ASAPs are listed. *Inferior frontal orbital gyrus* on both left and right hemisphere are captured, and several *temporal/orbital regions, precuneous*, and *left putamen* are shown to yield small scales. These ROIs are already known as AD-specific by various literature [1,7,11,33], which are highly affected by AD. At the bottom of Table 6, the RMSE between the estimated scales and the exact ones across all the ROIs are given, and ASAP-L and ASAP-H showed far better estimation than ASAP-C.

Table 6. 10 ROIs with the smallest trained scales for AD classification. RMSE on scales between Exact and ASAP are given at the bottom. (L) and (R) denote left and right hemisphere.

ROI [9]	Exact	ASAP-C	ASAP-H	ASAP-L
(L) G&S.paracentral	0.034	0.052	0.049	0.066
(L) G.front.inf.Orbital	0.036	0.071	0.060	0.043
(R) G.precuneus	0.041	0.044	0.034	0.051
(R) S.ortibal.med.olfact	0.047	0.078	0.054	0.059
(R) G.cingul.Post.ventral	0.055	0.056	0.055	0.051
(R) S.oc.temp.lat	0.055	0.065	0.045	0.063
(R) G.oc.temp.med.Lingual	0.055	0.076	0.043	0.040
(L) Sub.put	0.058	0.077	0.047	0.060
(L) S.postcentral	0.061	0.069	0.060	0.018
(R) G.front.inf.Orbital	0.063	0.093	0.069	0.050
RMSE for all ROIs	-	**0.5827**	**0.2904**	**0.2899**

Effect of K. We examined the performance of ASAP with respect to the number of convolution layers K on Cora and ADNI experiments. When K was varied from 1 to 4 under the same setting, K=2 showed the best performance in both experiments. The performance decrease when K=3 and 4 may be due to lack of training samples as the model sizes are drastically increased. We observed the same pattern across Exact and ASAP which demonstrate, again, that ASAP with approximation is able to train on s properly.

6. Conclusions

In this work, we proposed efficient trainable methods to bypass exact computation of spectral kernel convolution that define adaptive ranges of neighbor for each node. We have derived closed-form derivatives on polynomial coefficients to train the range, i.e., scale, with conventional back-propagation, and the developed framework ASAP demonstrate *state-of-the-art* performance on node classification and brain network classification. The brain network analysis provides neuroscientifically interpretable results corroborated by previous AD literature.

References

[1] Matthieu Bailly, Christophe Destrieux, Caroline Hommet, Karl Mondon, Jean-Philippe Cottier, Emilie Beaufils, Emilie Vierron, Johnny Vercouillie, Méziane Ibazizene, Thierry Voisin, et al. Precuneus and cingulate cortex atrophy and hypometabolism in patients with alzheimer's disease and mild cognitive impairment: Mri and 18f-fdg pet quantitative analysis using freesurfer. *BioMed research international*, 2015, 2015. 8

[2] Jasmijn Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. Graph convolutional encoders for syntax-aware neural machine translation. *arXiv preprint arXiv:1704.04675*, 2017. 1

[3] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018. 1

[4] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pages 1725–1735. PMLR, 2020. 1, 2, 6

[5] Theodore S Chihara. *An introduction to orthogonal polynomials*. Courier Corporation, 2011. 3

[6] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997. 2

[7] Laura W de Jong, Karin van der Hiele, Ilya M Veer, JJ Houwing, RGJ Westendorp, ELEM Bollen, Paul W de Bruin, HAM Middelkoop, Mark A van Buchem, and Jeroen van der Grond. Strongly reduced volumes of putamen and thalamus in alzheimer's disease: an mri study. *Brain*, 131(12):3277–3285, 2008. 8

[8] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016. 3

[9] Christophe Destrieux, Bruce Fischl, Anders Dale, and Eric Halgren. Automatic parcellation of human cortical gyri and sulci using standard anatomical nomenclature. *Neuroimage*, 53(1):1–15, 2010. 7, 8

[10] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. *Advances in neural information processing systems*, 33:22092–22103, 2020. 2, 6

[11] Clare J Galton, Karalyn Patterson, K Graham, Matthew A Lambon-Ralph, G Williams, N Antoun, BJ Sahakian, and JR Hodges. Differing patterns of temporal atrophy in alzheimer's disease and semantic dementia. *Neurology*, 57(2):216–225, 2001. 8

[12] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019. 1, 6

[13] 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. 1

[14] 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. 1, 2

[15] Mingguo He, Zhewei Wei, and Ji-Rong Wen. Convolutional neural networks on graphs with chebyshev approximation, revisited. *arXiv preprint arXiv:2202.03580*, 2022. 3

[16] Lianzhe Huang, Dehong Ma, Sujian Li, Xiaodong Zhang, and Houfeng Wang. Text level graph neural network for text classification. *arXiv preprint arXiv:1910.02356*, 2019. 1

[17] Shih-Gu Huang, Ilwoo Lyu, Anqi Qiu, and Moo K Chung. Fast polynomial approximation of heat kernel convolution on manifolds and its application to brain sulcal and gyral graph pattern analysis. *IEEE transactions on medical imaging*, 39(6):2201–2212, 2020. 2, 3, 4

[18] Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. *arXiv preprint arXiv:2204.04879*, 2022. 1, 6

[19] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 1, 2, 3, 6, 7

[20] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997*, 2018. 2, 6

[21] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. Diffusion improves graph learning. *arXiv preprint arXiv:1911.05485*, 2019. 2, 6, 7

[22] Wanyu Lin, Zhaolin Gao, and Baochun Li. Shoestring: Graph-based semi-supervised classification with severely limited labeled data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4174–4182, 2020. 6

[23] Wanyu Lin and Baochun Li. Medley: Predicting social trust in time-varying online social networks. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021. 1

[24] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 338–348, 2020. 2, 6

[25] Yuzong Liu and Katrin Kirchhoff. Graph-based semisupervised learning for acoustic modeling in automatic speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(11):1946–1956, 2016. 1

[26] Yi Luo, Guangchun Luo, Ke Yan, and Aiguo Chen. Inferring from references with differences for semi-supervised node classification on graphs. *Mathematics*, 10(8):1262, 2022. 6, 7

[27] Frank WJ Olver, Daniel W Lozier, Ronald F Boisvert, and Charles W Clark. *NIST handbook of mathematical functions hardback and CD-ROM*. Cambridge university press, 2010. 3, 4

[28] Alan V Oppenheim, John Buck, Michael Daniel, Alan S Willsky, Syed Hamid Nawab, and Andrew Singer. *Signals & systems*. Pearson Educación, 1997. 3

[29] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. 2

[30] Xiaojuan Qi, Renjie Liao, Jiaya Jia, Sanja Fidler, and Raquel Urtasun. 3d graph neural networks for rgbd semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5199–5208, 2017. 1

[31] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008. 6

[32] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018. 2, 6

[33] Gary W Van Hoesen, Josef Parvizi, and Ching-Chiang Chu. Orbitofrontal cortex pathology in alzheimer's disease. *Cerebral Cortex*, 10(3):243–251, 2000. 8

[34] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 1, 2, 6, 7

[35] Nitika Verma, Edmond Boyer, and Jakob Verbeek. Feastnet: Feature-steered graph convolutions for 3d shape analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2598–2606, 2018. 1

[36] Xin Wei, Ruixuan Yu, and Jian Sun. View-gcn: View-based graph convolutional network for 3d shape analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1850–1859, 2020. 1

[37] 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. 1, 2, 6

[38] Junran Wu, Xueyuan Chen, Ke Xu, and Shangzhe Li. Structural entropy guided graph hierarchical pooling. In *International Conference on Machine Learning*, pages 24017–24030. PMLR, 2022. 1, 6

[39] Guo-Sen Xie, Jie Liu, Huan Xiong, and Ling Shao. Scale-aware graph neural network for few-shot semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5475–5484, 2021. 1

[40] Bingbing Xu, Huawei Shen, Qi Cao, Keting Cen, and Xueqi Cheng. Graph convolutional networks using heat kernel for semi-supervised learning. *arXiv preprint arXiv:2007.16002*, 2020. 1, 2, 6, 7

[41] Xu Yang, Cheng Deng, Zhiyuan Dang, Kun Wei, and Junchi Yan. Selfsagcn: self-supervised semantic alignment for graph convolution network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16775–16784, 2021. 1, 3, 6