Higher-order Graph Convolutional Networks

John Boaz Lee[1], Ryan A. Rossi[2], Xiangnan Kong[1], Sungchul Kim[2], Eunyee Koh[2], and Anup Rao[2]
[1]Worcester Polytechnic Institute, MA, USA
[2]Adobe Research, CA, USA
{jtlee, xkong}@wpi.edu, {rrossi, sukim, eunyee, anuprao}@adobe.com

Abstract

Following the success of deep convolutional networks in various vision and speech related tasks, researchers have started investigating generalizations of the well-known technique for graph-structured data. A recently-proposed method called Graph Convolutional Networks has been able to achieve state-of-the-art results in the task of node classification. However, since the proposed method relies on localized first-order approximations of spectral graph convolutions, it is unable to capture higher-order interactions between nodes in the graph. In this work, we propose a motif-based graph attention model, called Motif Convolutional Networks, which generalizes past approaches by using weighted multi-hop motif adjacency matrices to capture higher-order neighborhoods. A novel attention mechanism is used to allow each individual node to select the most relevant neighborhood to apply its filter. Experiments show that our proposed method is able to achieve state-of-the-art results on the semi-supervised node classification task.

Introduction

In recent years, deep learning has made a significant impact on the field of computer vision. Various deep learning models have achieved state-of-the-art results on a number of vision-related benchmarks. In most cases, the preferred architecture is a Convolutional Neural Network (CNN). CNN models have been applied successfully to the tasks of image classification (Krizhevsky, Sutskever, and Hinton 2012), image super-resolution (Kim, Lee, and Lee 2016), and video action recognition (Feichtenhofer, Pinz, and Zisserman 2016), among many others.

CNNs, however, are designed to work for data that can be represented as grids (*e.g.*, videos, images, or audio clips) and do not generalize to graphs – which have more irregular structure. Due to this limitation, it cannot be applied directly to many real-world problems whose data come in the form of graphs – social networks (Perozzi, Al-Rfou, and Skiena 2014) or citation networks (Lu and Getoor 2003) in social network analysis, for instance.

A recent deep learning architecture, called Graph Convolutional Networks (GCN) (Kipf and Welling 2017) approximates the spectral convolution operation on graphs by defining a layer-wise propagation that is based on the one-hop neighborhood of nodes. The first-order filters used by GCNs were found to be useful and have allowed the model to beat many established baselines in the semi-supervised node classification task.

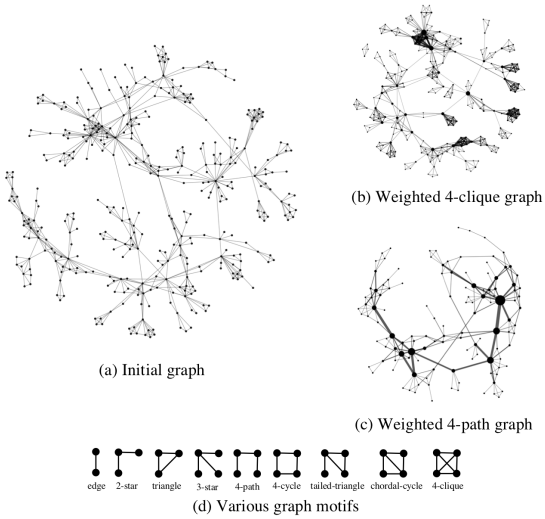

Figure 1: Graphs defined by different motif adjacencies can differ significantly. Size (weight) of nodes and edges in (b) and (c) correspond to the frequency of 4-node cliques and 4-node paths between nodes, respectively.

However, in many cases, it has been shown that it may be beneficial to consider the higher-order structure in graphs (Yang et al. 2018; Rossi, Ahmed, and Koh 2018; Milo et al. 2002; Rossi, Zhou, and Ahmed 2018b). In this work, we introduce a general class of graph convolution networks which utilize weighted multi-hop *motif* adjacency matrices (Rossi, Ahmed, and Koh 2018) to capture higher-order neighborhoods in the graph. The weighted adjacency matrices are computed using various network motifs (Rossi, Ahmed, and Koh 2018). Fig. 1 shows an example of the node neighborhoods that are induced when we consider two different kinds of motifs, showing that the choice of motif can significantly alter the neighborhood structure of nodes.

Our proposed method, which we call Motif Convolutional Networks (MCN), uses a novel attention mechanism to allow each node to select the most relevant motif-induced neighborhood to integrate information from. Intuitively, this allows a node to select its one-hop neighborhood (as in classical GCN) when its immediate neighborhood contains

enough information for the model to classify the node correctly but gives it the flexibility to select an alternative neighborhood (defined by higher-order structures) when the information in its immediate vicinity is too sparse and/or noisy.

The aforementioned attention mechanism is trained using reinforcement learning which rewards choices (*i.e*, actions) that consistently result in a correct classification. Our main contributions can be summarized as follows:

- We propose a model that generalizes GCNs by introducing multiple weighted motif-induced adjacencies that capture various higher-order neighborhoods.
- We introduce a novel attention mechanism that allows the model to choose the best neighborhood to integrate information from.
- We demonstrate the superiority of the proposed method by comparing against strong baselines on three established graph benchmarks. We also show that it works comparatively well on graphs exhibiting heterophily.
- We demonstrate the usefulness of attention by showing how different nodes prioritize different neighborhoods.

Related Literature

Neural Networks for Graphs Initial attempts to adapt neural network models to work with graph-structured data started with recursive models that treated the data as directed acyclic graphs (Sperduti and Starita 1997; Frasconi, Gori, and Sperduti 1998). Later on, more generalized models called Graph Neural Networks (GNN) were introduced to process arbitrary graph-structured data (Gori, Monfardini, and Scarselli 2005; Scarselli et al. 2009).

Recently, with the rise of deep learning and the success of models such as recursive neural networks (RNN) (Hausknecht and Stone 2015; Zhou et al. 2016) for sequential data and CNNs for grid-shaped data, there has been a renewed interest in adapting some of these approaches to arbitrary graph-structured data.

Some work introduced architectures tailored for more specific problem domains (Li et al. 2016; Duvenaud et al. 2015) – like NeuralFPS (Duvenaud et al. 2015) which is an end-to-end differentiable deep architecture which generalizes the well-known Weisfeiler-Lehman algorithm for molecular graphs – while others defined graph convolutions based on spectral graph theory (Henaff, Bruna, and LeCun 2015). Another group of methods attempt to substitute principled-yet-expensive graph convolutions using spectral approaches by using approximations of such. Defferrard, Bresson, and Vandergheynst (2016) used Chebyshev polynomials to approximate a smooth filter in the spectral domain while GCNs (Kipf and Welling 2017) further simplified the process by using first-order filters.

The model introduced by Kipf and Welling (2017) has been shown to work well on a variety of graph-based tasks (Nguyen and Grishman 2018; Yan, Xiong, and Lin 2018; Kipf and Welling 2017) and have spawned variants including (Velickovic et al. 2018; Abu-El-Haija et al. 2018). We introduce a generalization of GCN (Kipf and Welling 2017) in this work but we differ from past approaches in two main points: first, we use weighted motif-induced adjacencies to expand the possible kinds of node neighborhoods available to nodes, and secondly, we introduce a novel attention mechanism that allows each node to select the most relevant neighborhood to diffuse (or integrate) information.

Higher-order Structures with Network Motifs Network motifs (Milo et al. 2002) are fundamental building blocks of complex networks; investigation of such patterns usually lead to the discovery of crucial information about the structure and the function of many complex systems that are represented as graphs. Prill, Iglesias, and Levchenko (2005) studied motifs in biological networks showing that the dynamical property of robustness to perturbations correlated highly to the appearance of certain motif patterns while Paranjape, Benson, and Leskovec (2017) looked at motifs in temporal networks showing that graphs from different domains tend to exhibit very different organizational structures as evidenced by the type of motifs present.

Multiple work have demonstrated that it is useful to account for higher-order structures in different graph-based ML tasks (Rossi, Ahmed, and Koh 2018; Yang et al. 2018; Ahmed et al. 2018). DeepGL (Rossi, Zhou, and Ahmed 2018a) uses motifs as a basis to learn deep inductive relational functions that represent compositions of relational operators applied to a base graph function such as triangle counts. Rossi, Ahmed, and Koh (2018) proposed the notion of *higher-order network embeddings* and demonstrated that one can learn better embeddings when various motif-based matrix formulations are considered. Yang et al. (2018) defined a hierarchical motif convolution for the task of subgraph identification for graph classification. In contrast, we propose a new class of higher-order network embedding methods based on graph convolutions that uses a novel motif-based attention for the task of semi-supervised node classification.

Attention Models Attention was popularized in the deep learning community as a way for models to attend to important parts of the data (Mnih et al. 2014; Bahdanau, Cho, and Bengio 2015). The technique has been successfully adopted by models solving a variety of tasks. For instance, it was used by Mnih et al. (2014) to take glimpses of relevant parts of an input image for image classification; on the other hand, Xu et al. (2015) used attention to focus on task-relevant parts of an image for the image captioning task. Meanwhile, Bahdanau, Cho, and Bengio (2015) utilized attention for the task of machine translation by fixing the model attention on specific parts of the input when generating the corresponding output words.

There has also been a surge in interest at applying attention to deep learning models for graphs. The work of Velickovic et al. (2018) used a node self-attention mechanism to allow each node to focus on features in its neighborhood that were more relevant while Lee, Rossi, and Kong (2018) used attention to guide a walk in the graph to learn an embedding for the graph. More specialized methods of graph attention models include (Choi et al. 2017; Han, Liu, and Sun 2018) with Choi et al. (2017) using attention on a medical ontology graph for medical diagnosis and Han, Liu, and Sun (2018) using attention on a knowledge graph for the task of entity

link prediction. Our approach differs significantly, however, from previous approach in that we use attention to allow our model to select task relevant neighborhoods.

Approach

We begin this section by introducing the foundational layer that is used to construct arbitrarily deep motif convolutional networks. When certain constraints are imposed on our model's architecture, the model degenerates into a GAT (Velickovic et al. 2018) which, in turn, generalizes a GCN (Kipf and Welling 2017). Because of this, we briefly introduce a few necessary concepts from (Velickovic et al. 2018; Kipf and Welling 2017) before defining the actual neural architecture we employ – including the reinforcement learning strategy we use to train our attention mechanism.

Graph Self-Attention Layer

A multi-layer GCN (Kipf and Welling 2017) is constructed using the following layer-wise propagation:

$$\mathbf{H}^{(l+1)} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}). \tag{1}$$

Here, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the modified adjacency matrix of the input graph with added self-loops – $\mathbf{A}$ is the original adjacency matrix of the input undirected graph with N nodes while $\mathbf{I}_N$ represents an identity matrix of size N. The matrix $\tilde{\mathbf{D}}$, on the other hand, is the diagonal degree matrix of $\tilde{\mathbf{A}}$ (*i.e.*, $\tilde{D}_{i,i} = \sum_j \tilde{A}_{i,j}$). Finally, $\mathbf{H}^{(l)}$ is the matrix of node features inputted to layer l while $\mathbf{W}^{(l)}$ is a trainable embedding matrix used to embed the given inputs (typically to a lower dimension) and σ is a non-linearity.

The term $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ in Eq. 1 produces a symmetric normalized matrix which update's each nodes representation via a weighted sum of the features in a node's one-hop neighborhood (the added self-loop allows the model to include the node's own features). Each link's strength (*i.e.*, weight) is normalized by considering the degrees of the corresponding nodes. Formally, at each layer l, node i integrates neighboring features to obtain a new feature/embedding via

$$\vec{\mathbf{h}}_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}_i^{(\tilde{\mathbf{A}})}} \alpha_{i,j}\vec{\mathbf{h}}_j^{(l)}\mathbf{W}^{(l)}\right), \tag{2}$$

where $\vec{\mathbf{h}}_i^{(l)}$ is the feature of node i at layer l, with fixed weights $\alpha_{i,j} = \frac{1}{\sqrt{\deg(i)\deg(j)}}$, and $\mathcal{N}_i^{(\tilde{\mathbf{A}})}$ is the set of i's neighbors defined by the matrix $\tilde{\mathbf{A}}$ – which includes itself.

In GAT (Velickovic et al. 2018), Eq. 2 is modified with weights α that are differentiable or trainable and this can be viewed as follows,

$$\alpha_{i,j} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}[\vec{\mathbf{h}}_i\mathbf{W}\ \vec{\mathbf{h}}_j\mathbf{W}]\right)\right)}{\sum_{k \in \mathcal{N}_i^{(\tilde{\mathbf{A}})}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}[\vec{\mathbf{h}}_i\mathbf{W}\ \vec{\mathbf{h}}_k\mathbf{W}]\right)\right)}. \tag{3}$$

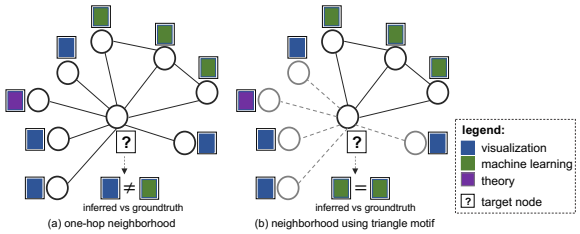

Figure 2: A researcher (target node) may have collaborated on various projects in visualization and theory. However, his main research focus is ML and he collaborates closely with lab members who also work among themselves. (a) If we simply use the target node's one-hop neighborhood, we may incorrectly infer his research area; however, (b) when we limit his neighborhood using the triangle motif, we reveal neighbors connected via stronger bonds giving us a better chance at inferring the correct research area. This observation is empirically shown in our experimental results.

The attention vector $\mathbf{a}$ in Eq. 3 is a trainable weight vector that assigns importance to the different neighbors of i allowing the model to highlight particular neighboring node features that are more task-relevant.

Using the formulation in Eq. 3 with Eqs. 1 and 2, we can now define multiple layers which can be stacked together to form a deep GCN (with self-attention) that is end-to-end differentiable. The initial input to the model can be set as $\mathbf{H}^{(1)} = \mathbf{X}$, where $\mathbf{X} \in \mathbb{R}^{N \times D}$ is the initial node attribute matrix with D attributes. The final layer's weight matrix can also be set accordingly to output node embeddings at the desired output dimensions.

Convolutional Layer with Motif Attention

We observe that both GCN and GAT rely on the one-hop neighborhood of nodes (*i.e.*, $\tilde{\mathbf{A}}$ in Eq. 1) to propagate information. However, it may not always be suitable to apply a single uniform definition of node neighborhood for all nodes. For instance, we show an example in Fig. 2 where a node can benefit from using a neighborhood defined using triangle motifs to keep only neighbors connected via a stronger bond which is a well-known concept from social theory allowing us to distinguish between weaker ties and strong ones via the triadic closure (Friggeri, Chelius, and Fleury 2011).

Weighted Motif-Induced Adjacencies Given a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $N = |\mathcal{V}|$ nodes, $M = |\mathcal{E}|$ edges, as well as a set of T network motifs[1] $\mathcal{H} = \{H_1, \cdots, H_T\}$, we can construct T different motif-induced adjacency matrices $\mathcal{A} = \{\mathbf{A}_1, \cdots, \mathbf{A}_T\}$ with:

$$(\mathbf{A}_t)_{i,j} = \#\text{ of motifs of type } H_t \text{ which contain } (i,j) \in \mathcal{E}.$$

As shown in Fig. 1, neighborhoods defined by different motifs can vary significantly. Furthermore, the weights in a motif-induced adjacency $\mathbf{A}_t$ can also vary as motifs can appear in varying degrees of frequency between different pairs of nodes.

[1]We use the term motifs loosely here and it can also be used to mean graphlets or orbits (Ahmed et al. 2017).

Motif Matrix Functions Each of the calculated motif adjacencies $\mathbf{A}_t \in \mathcal{A}$ can now be potentially used to define motif-induced neighborhoods $\mathcal{N}_i^{(\mathbf{A}_t)}$ with respect to a node i. While Eq. 3 defines self-attention weights over a node's neighborhood, the initial weights in $\mathbf{A}_t$ can still be used as reasonable initial estimates of each neighbor's "importance."

Hence, we introduce a *motif-based matrix formulation* as a function $\Psi : \mathbb{R}^{N\times N} \rightarrow \mathbb{R}^{N\times N}$ over a motif adjacency $\mathbf{A}_t \in \mathcal{A}$ similar to (Rossi, Ahmed, and Koh 2018). Given a function Ψ, we can obtain *motif-based matrices* $\tilde{\mathbf{A}}_t = \Psi(\mathbf{A}_t)$, for $t = 1, \cdots, T$. Below, we summarize the different variants of Ψ that we chose to investigate.

• **Unweighted Motif Adjacency w/ Self-loops:** In the simplest case, we can construct $\tilde{\mathbf{A}}$ (here on, we omit the subscripts t for brevity) from $\mathbf{A}$ by simply ignoring the weights:

$$\tilde{\mathbf{A}}_{i,j} = \begin{cases} 1 & i = j \\ 1 & \mathbf{A}_{i,j} > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

But, as mentioned above, we lose the initial benefit of leveraging the weights in the motif-induced adjacency $\mathbf{A}$.

• **Weighted Motif Adjacency w/ Row-wise Max:** We can also choose to retain the weighted motif adjacency $\mathbf{A}$ without modification save for added row-wise maximum self-loops. This is defined as

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{M}, \tag{5}$$

where $\mathbf{M}$ is a diagonal square matrix with $M_{i,i} = \max_{1\leq j\leq N} A_{i,j}$. Intuitively, this allows us to assign an equal amount of importance to a self-loop consistent with that given to each node's most important neighbor.

• **Motif Transition w/ Row-wise Max:** The random walk on the weighted graph with added row-wise maximum self-loops has transition probabilities $P_{i,j} = \frac{A_{i,j}}{(\sum_k A_{i,k}) + (\max_{1\leq k\leq N} A_{i,k})}$. Our random walk motif transition matrix can thus be calculated by

$$\tilde{\mathbf{A}} = \mathbf{D}^{-1}(\mathbf{A} + \mathbf{M}), \tag{6}$$

where, in this context, the matrix $\mathbf{D}$ is the diagonal square degree matrix of $\mathbf{A} + \mathbf{M}$ (*i.e.*, $D_{i,i} = (\sum_k A_{i,k}) + (\max_{1\leq k\leq N} A_{i,k})$) while $\mathbf{M}$ is defined as above. Here, $\tilde{A}_{i,j} = P_{i,j}$ or the transition probability from node i to j is proportional to the motif count between nodes i and j relative to the total motif count between i and all its neighbors.

• **Absolute Motif Laplacian:** The absolute Laplacian matrix can be constructed as follows:

$$\tilde{\mathbf{A}} = \mathbf{D} + \mathbf{A}. \tag{7}$$

Here, the matrix $\mathbf{D}$ is the degree matrix of $\mathbf{A}$. Note that because the self-loop is a sum of all the weights to a node's neighbors, the initial importance of the node itself can be disproportionately large.

• **Symmetric Normalized Matrix w/ Row-wise Max:** Finally, we calculate a symmetric normalized matrix (similar to the normalized Laplacian) via:

$$\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{M})\mathbf{D}^{-\frac{1}{2}}. \tag{8}$$

Here, based on the context, the matrix $\mathbf{D}$ is the diagonal degree matrix of $\mathbf{A} + \mathbf{M}$.

K-Step Motif Matrices Given a step-size K, we further define K different k-step motif-based matrices for each of the T motifs which gives a total of $K \times T$ adjacency matrices. Formally, this is formulated as follows:

$$\tilde{\mathbf{A}}_t^{(k)} = \Psi(\mathbf{A}_t^k), \text{ for } k = 1, \cdots, K \text{ and } t = 1, \cdots, T \tag{9}$$

where

$$\Psi(\mathbf{A}_t^k) = \Psi(\underbrace{\mathbf{A}_t \cdots \mathbf{A}_t}_{k}) \tag{10}$$

When we set $K > 1$, we allow nodes to accumulate information from a wider neighborhood. For instance if we choose to use Eq. 4 (for Ψ) and use an edge as our motif, $\tilde{\mathbf{A}}^{(k)}$ (we omit the motif-type subscript here) then captures k-hop neighborhoods of each node. While, in theory, using $\tilde{\mathbf{A}}^{(k)}$ is equivalent to a k-layer GCN or GAT model (Abu-El-Haija et al. 2018) have shown that GCNs don't necessarily benefit from a wider receptive field from increasing model depth. This may be for reasons similar as to why skip-connections are needed in deep architectures since the signal starts to degrade as the model gets deeper (He et al. 2016).

As another example, we set Ψ to Eq. 6. Now for an arbitrary motif, we see that $(\tilde{\mathbf{A}}^{(k)})_{i,j}$ encodes the probability of transitioning from node i to node j in k steps.

Motif Matrix Selection via Attention Given T different motifs and a step-size of K, we now have $K \times T$ motif matrices we could use with Eq. 1 to define layer-wise propagations. A simple approach would be to implement $K \times T$ idependent GCN instances and concatenate the final node outputs before classification. However, this approach may have problems scaling when T and/or K is large.

Instead, we propose to use an attention mechanism, at each layer, to allow each node to select *a single* most relevant neighborhood to integrate or accumulate information from. For a layer l, this can be defined by two functions $f_l : \mathbb{R}^{S_l} \rightarrow \mathbb{R}^T$ and $f_l' : \mathbb{R}^{S_l} \times \mathbb{R}^T \rightarrow \mathbb{R}^K$, where S_l is the dimension of the state-space for layer l. The functions' outputs are *softmaxed* to form probability distributions over $\{1, \cdots, T\}$ and $\{1, \cdots, K\}$, respectively. Essentially, what this means is that given a node i's state, the functions recommend the most relevant motif t and step size k for node i to integrate information from.

Specifically, we define the state matrix encoding node states at layer l as a concatenation of two matrices:

$$\mathbf{S}_l = \begin{bmatrix} \Psi(\mathbf{A})\mathbf{H}^{(l)}\mathbf{W}^{(l)} & \mathbf{C} \end{bmatrix}, \tag{11}$$

where $\mathbf{W}^{(l)} \in \mathbb{R}^{N\times D_l}$ is the weight matrix that embeds the inputs to dimension D_l, $\Psi(\mathbf{A})\mathbf{H}^{(l)}\mathbf{W}^{(l)}$ is the matrix containing local information obtained by doing a weighted sum of the features in the simple one-hop neighborhood for each node (from the original adjacency $\mathbf{A}$), and $\mathbf{C} \in \mathbb{R}^{N\times C}$ is a motif count matrix that gives us basic local structural information about each node by counting the number of C different motifs that each node belongs to. We note here that $\mathbf{C}$ is *not* appended to the node attribute matrix $\mathbf{X}$ and is not used for prediction. It's only purpose is to capture the local structural information of each node. $\mathbf{C}$ is precomputed once.

Let us consider an arbitrary layer. Recall that f (for brevity, we omit subscripts l) produces a probability vector specifying the importance of the various motifs, let $\vec{\mathbf{f}}_i = f(\vec{\mathbf{s}}_i)$ be the motif probabilities for node i. Similarly, let $\vec{\mathbf{f}}'_i = f'(\vec{\mathbf{s}}_i, \vec{\mathbf{f}}_i)$ be the probability vector recommending the step size. Now let t_i be the index of the largest value in $\vec{\mathbf{f}}_i$ and similarly, let k_i be the index of the largest value in $\vec{\mathbf{f}}'_i$. In other words, t_i is the recommended motif for i while k_i is the recommended step-size. Attention can now be used to define an $N \times N$ propagation matrix as follows:

$$\hat{\mathbf{A}} = \begin{bmatrix} \left(\tilde{\mathbf{A}}_{t_1}^{(k_1)}\right)_{1,:} \\ \vdots \\ \left(\tilde{\mathbf{A}}_{t_N}^{(k_N)}\right)_{N,:} \end{bmatrix}. \tag{12}$$

This layer-specific matrix $\hat{\mathbf{A}}$ can now be plugged into Eq. 1 to replace $\tilde{\mathbf{A}}$. What this does is it gives each node the flexibility to select the most appropriate motif t and step-size k to integrate information from.

Training the Attention Mechanism Given a labeled graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \ell)$ with N nodes and a labeling function $\ell : \mathcal{V} \rightarrow \mathcal{L}$ which maps each node to one of L class labels in $\mathcal{L} = \{1, \cdots, L\}$, our goal is to train a classifier that can predict the label of all the nodes. Given a subset $\mathcal{T} \subset \mathcal{V}$, or the training set of nodes, we can train an L-layer MCN (the classifier) using standard cross-entropy loss as follows:

$$\mathcal{L}_C = -\sum_{v \in \mathcal{T}} \sum_{l=1}^{L} Y_{vl} \log \pi(H_{i,l}^{(L+1)}), \tag{13}$$

where Y_{vl} is a binary value indicating node v's true label (*i.e.*, $Y_{vl} = 1$ if $\ell(v) = l$, zero otherwise), and $\mathbf{H}^{L+1} \in \mathbb{R}^{N \times L}$ is the *softmaxed* output of the MCN's last layer.

While Eq. 13 is sufficient for training the MCN to classify inputs it does not tell us how we can train the attention mechanism that selects the best motif and step-size for each node at each layer. We define a second loss function based on reinforcement learning as follows:

$$\begin{aligned} \mathcal{L}_A = &- \textstyle\sum_{n_L \in \mathcal{T}} R_v \left[\log \pi \left(\left(\vec{\mathbf{f}}_{n_L}^{(L)} \right)_{t_{n_L}^{(L)}} \right) + \log \pi \left(\left(\vec{\mathbf{f}}_{n_L}^{(L)} \right)_{k_{n_L}^{(L)}} \right) \right] \\ &+ \textstyle\sum_{n_L \in \mathcal{T}} \sum_{n_{L-1} \in \mathcal{N}_{n_L}^{(\hat{\mathbf{A}}^{(L)})}} R_v \left[\log \pi \left(\left(\vec{\mathbf{f}}_{n_{L-1}}^{(L-1)} \right)_{t_{n_{L-1}}^{(L-1)}} \right) + \log \pi \left(\left(\vec{\mathbf{f}}_{n_{L-1}}^{(L-1)} \right)_{k_{n_{L-1}}^{(L-1)}} \right) \right] \\ &+ \cdots + \textstyle\sum_{n_L \in \mathcal{T}} \cdots \sum_{n_1 \in \mathcal{N}_{n_2}^{(\hat{\mathbf{A}}^{(2)})}} R_v \left[\log \pi \left(\left(\vec{\mathbf{f}}_{n_1}^{(1)} \right)_{t_{n_1}^{(1)}} \right) + \log \pi \left(\left(\vec{\mathbf{f}}_{n_1}^{(1)} \right)_{k_{n_1}^{(1)}} \right) \right] \end{aligned} \tag{14}$$

Here, R_v is the reward we give to the system ($R_v = 1$ if we classify v correctly, $R_v = -1$ otherwise). The intuition here is this: at the last layer we reward the actions of the classified nodes; we then go to the previous layer (if there is one) and reward the actions of the neighbors of the classified nodes since their actions affect the outcome, we continue this process until we reach the first layer.

There are a few important things to point out. In practice, we use an ϵ-greedy strategy when selecting a motif and step-size during training. Specifically, we pick the action with highest probability most of the time but during $1 - \epsilon$ instances we select a random action. During testing, we choose the action with highest probability. Also, in practice, we use dropout to train the network as in GAT (Velickovic et al. 2018) which is a good regularization technique but also has the added advantage of being a way to sample the neighborhood during training to keep the receptive field from growing too large during training. Finally, to reduce model variance we can also include an advantage term (see Eq. 2 in (Lee, Rossi, and Kong 2018), for instance). Our final loss can then be written as:

$$\mathcal{L} = \mathcal{L}_C + \mathcal{L}_A. \tag{15}$$

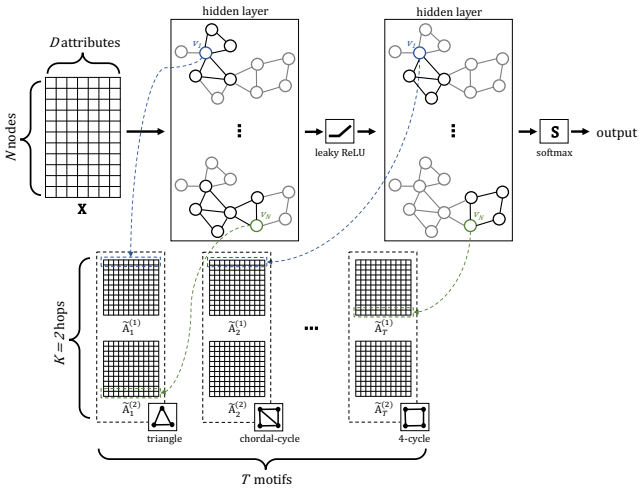

Figure 3: An example of a 2-layer MCN with $N = 11$ nodes, step-size $K = 2$, and T motifs. Attention allows each node to select a different motif-induced neighborhood to accumulate information from for each layer. For instance, in layer 1, the node v_N considers neighbors (up to 2-hops) that share a stronger bond (in this case, triangles) with it.

We show a simple (2-layer) example of the proposed MCN model in Fig. 3. As mentioned, MCN generalizes both GCN and GAT. We list settings of these methods in Tab. 1.

Table 1: Space of methods expressed by MCN. GCN and GAT are shown below to be special cases of MCN.

Method	Motif	Adj.	K	Self-attention	Motif-attention
GCN	edge	Eq. 4	$K = 1$	no	no
GAT	edge	Eq. 4	$K = 1$	yes	no
MCN-*	any	Eqs. 4-8	$K = \{1, \cdots\}$	yes	yes

Experimental Results

Semi-supervised node classification

We first compare our proposed approach against a set of strong baselines (including methods that are considered the current state-of-the-art) on three well-known graph benchmark datasets for semi-supervised node classification. We show that the proposed method is able to achieve state-of-the-art results on all compared datasets.

Table 2: Summary of experimental results: "average accuracy ± SD (rank)". The "avg. rank" column shows the average rank of each method. The lower the average rank, the better the overall performance of the method.

method	dataset			avg. rank
	Cora	Citeseer	Pubmed	
DeepWalk (Perozzi, Al-Rfou, and Skiena 2014)	67.2% (9)	43.2% (11)	65.3% (11)	10.3
MLP	55.1% (12)	46.5% (9)	71.4% (9)	10.0
LP (Zhu, Ghahramani, and Lafferty 2003)	68.0% (8)	45.3% (10)	63.0% (12)	10.0
ManiReg (Belkin, Niyogi, and Sindhwani 2006)	59.5% (10)	60.1% (7)	70.7% (10)	9.0
SemiEmb (Weston et al. 2012)	59.0% (11)	59.6% (8)	71.7% (8)	9.0
ICA (Lu and Getoor 2003)	75.1% (7)	69.1% (5)	73.9% (7)	6.3
Planetoid (Yang, Cohen, and Salakhutdinov 2016)	75.7% (6)	64.7% (6)	77.2% (5)	5.7
Chebyshev (Defferrard, Bresson, and Vandergheynst 2016)	81.2% (5)	69.8% (4)	74.4% (6)	5.0
MoNet (Monti et al. 2016)	81.7% (3)	–	78.8% (4)	3.5
GCN (Kipf and Welling 2017)	81.5% (4)	70.3% (3)	79.0% (2)	3.0
GAT (Velickovic et al. 2018)	83.0 ± 0.7% (2)	72.5 ± 0.7% (2)	79.0 ± 0.3% (2)	2.0
MCN (this paper)	83.5 ± 0.4% (1)	73.3 ± 0.7% (1)	79.3 ± 0.3% (1)	1.0

Datasets The datasets used for comparison are Cora, Citeseer, and Pubmed. Specifically, we use the pre-processed version made available by Yang, Cohen, and Salakhutdinov (2016). The aforementioned graphs are undirected citation networks where nodes represent documents and edges denote citation; furthermore, a bag-of-words vector capturing word counts in each document serves as each node's feature. Each document is assigned a class label.

The graph in Cora has $|\mathcal{V}| = 2,708$, $|\mathcal{E}| = 5,429$, 7 classes, and $D = 1,433$ node features. The statistics for Citeseer are $|\mathcal{V}| = 3,327$, $|\mathcal{E}| = 4,732$, with 6 classes, and $D = 3,703$. Finally, Pubmed consists of a graph with $|\mathcal{V}| = 19,717$, $|\mathcal{E}| = 44,338$, with 3 classes, and $D = 500$. Following previous work, we use *only* 20 nodes per class for training (Yang, Cohen, and Salakhutdinov 2016; Kipf and Welling 2017; Velickovic et al. 2018). Again, following the procedure in previous work, we take 1,000 nodes per dataset for testing and further take an additional 500 for validation as in (Kipf and Welling 2017; Velickovic et al. 2018; Abu-El-Haija et al. 2018). We use the same train/test/validation splits as defined in (Kipf and Welling 2017; Velickovic et al. 2018).

Setup For Cora and Citeseer, we used the same 2-layer model architecture as that of GAT consisting of 8 self-attention heads each with a total of 8 hidden nodes (for a total of 64 hidden nodes) in the first layer, followed by a single softmax layer for classification (Velickovic et al. 2018). Similarly, we fixed early-stopping patience at 100 and ℓ_2-regularization at 0.0005. For Pubmed, we also used the same architecture as that of GAT (first layer remains the same but the output layer has 8 attention heads to deal with sparsity in the training data). Patience remains the same and similar to GAT, we use a strong ℓ_2-regularization at 0.001.

We further optimized all models by testing dropout values of $\{0.50, 0.55, 0.60, 0.65\}$, learning rates of $\{0.05, 0.005\}$, step-sizes $K \in \{1, 2, 3\}$, and motif adjacencies formed using combinations of the following motifs: *edge*, *2-star*, *triangle*, *3-star*, and *4-clique* (please refer to Tab. 1(d) for motifs). Self-attention learns to prioritize neighboring features that are more relevant. Ψ (Eqs. 4-8) can be used as a reasonable initial estimate of the importance of neighboring features. For each unique setting of the hyperparameters mentioned previously, we try Eqs. 4-8 and record the best result. Finally, we adopt an ϵ-greedy strategy ($\epsilon = 0.1$).

Comparison For all three datasets, we report the classification accuracy averaged over 15 runs on random seeds (including standard deviation). Since we utilize the same train/test splits as previous work, we follow (Velickovic et al. 2018; Kipf and Welling 2017) and compile all previously reported results.

A summary of the results is shown in Tab. 2. We see that our proposed method achieves superior performance against all tested baselines on all three benchmarks. On the Cora dataset, the best model used a learning rate of 0.005, dropout of 0.6, and both the edge and triangle motifs with step-size $K = 1$. For Citeseer, the learning rate was 0.05 and dropout was still 0.6 while the only motif used was the edge motif with step-size $K = 2$. However, the second best model for Citeseer – which had comparable performance – utilized the following motifs: edge, 2-star, and triangle. Finally, on Pubmed, the best model used learning rate 0.05 and dropout of 0.5. Once again, the best motifs were the edge and triangle motifs on $K = 1$.

We find that the triangle motif is useful in improving classification performance on the compared datasets. This highlights an advantage of MCN over past approaches (*e.g.*, GCN & GAT) which do not handle triangles (and other motifs) naturally. The results seem to indicate that it can be beneficial to consider stronger bonds (friends that are friends themselves) when selecting a neighborhood.

Comparison on Datasets exhibiting Heterophily

The benchmark datasets (Cora, Citeseer, and Pubmed) that we initially tested our method on exhibited strong homophily where nodes that share the same labels tend to form densely connected communities. Under these circumstances, methods like GAT or GCN that use a first-order propagation rule will perform reasonably well. However, not all real-world graphs share this characteristic and in some cases the node labels are more spread out. In this latter case,

Table 3: Micro-F1 scores of compared methods on DD.

method	dataset	
	DD-6	DD-7
GCN	$11.9 \pm 0.6\%$	$12.4 \pm 0.8\%$
GAT	$11.8 \pm 0.5\%$	$11.8 \pm 1.1\%$
MCN	$\mathbf{12.4 \pm 0.5\%}$	$\mathbf{13.1 \pm 0.9\%}$

there is reason to believe that neighborhoods constructed using different motifs – other than just edges and triangles – may be beneficial.

We test this hypothesis by comparing GAT and GCN against MCN on two graphs from the DD dataset (Kersting et al. 2016). Specifically, we chose two of the largest graphs in the dataset: DD-6 and DD-7 – with a total of $4,152$ and $1,396$ nodes, respectively. Both graphs had twenty different node labels with the labels being quite imbalanced.

We stick to the semi-supervised training regime, using only 15 nodes per class for training with the rest of the nodes split evenly between testing and validation. This makes the problem highly challenging since the graphs do not exhibit homophily. Since the nodes do not have any attributes, we use the well-known Weisfeiler-Lehman algorithm (we initialize node attributes to a single value and run the algorithm for 3 iterations) to generate node attributes that capture each node's neighborhood structure.

For the three approaches (GCN, GAT, and MCN), we fix early-stop patience at 50 and use a two-layer architecture with 32 hidden nodes in the first layer followed by the softmax output. We optimized the hyperparameters by searching over learning rate in $\{0.05, 0.005\}$, ℓ_2 regularization in $\{0.01, 0.001, 0.0001, 0.00001$, dropout at $\{0.2, 0.3, 0.4, 0.5, 0.6\}$. Furthermore, for MCN, we considered combinations of the following motifs {edge, 2-star, triangle, 4-path-edge, 3-star, 4-cycle, 4-clique} and considered K-steps from $1, \cdots, 4$. Since there are multiple classes and they are highly imbalanced, we report the Micro-F1 score averaged over 10 runs.

A summary of the results is shown in Tab. 3. While the methods do not perform very well (to be expected since we use a very small subset for training on graphs that do not have a high degree of homophily) we do find that with everything else constant (model architecture), it is actually valuable to use motifs. For DD-6, the best method utilized all motifs except for the 4-path-edge with $K = 1$ while in DD-7 the best approach only used the edge, triangle, and 4-clique motifs with $K = 4$.

Visualizing Motif Attention

We ran an instance of MCN with the following motifs: edge, 4-path, and triangle with $K = 1$ on the Cora dataset. Fig. 4 shows the motif that was selected by the attention mechanism. A few interesting things can be observed here. First, the nodes at the fringe prioritized the 4-path motif which is quite intuitive since this allows the nodes to aggregate information from a wider (4-hop) neighborhood which is useful for nodes near the fringe that are more separated from the other nodes in the class. On the other hand, we observe that

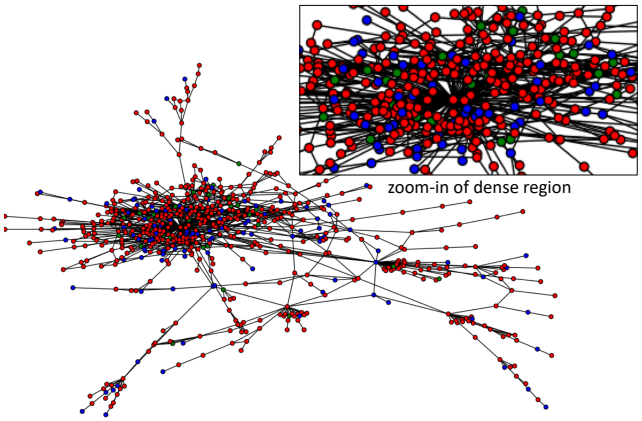

Figure 4: Nodes in the largest connected component of an induced subgraph in Cora (containing nodes from class 4). The nodes are colored to indicate the motif chosen by the attention mechanism in the first layer. The motifs are: edge (blue), 4-path (red), and triangle (green). We observe that the nodes near the fringe of the network tend to select the 4-path allowing them to aggregate information from a wider neighborhood. On the other hand, nodes that choose triangle motifs can be found in the denser regions where it may be helpful to take advantage of stronger bonds.

nodes that chose the triangle motif are almost always found in denser parts of the graph. This shows that it may be beneficial in these cases to consider stronger bonds (*e.g.*, when the neighborhood is noisy). Finally, we see that attention allows different nodes to select different motifs and the system is not "defaulting" to a single motif type.

Conclusion

In this work, we introduce the Motif Convolutional Network which uses a novel attention mechanism to allow different nodes to select a different neighborhood to integrate information from. The method generalizes both GAT and GCN. Experiments on three citation (Cora, Citeseer, & Pubmed) and two bioinformatic (DD-6 & DD-7) benchmark graphs show the advantage of the proposed approach. We also show experimentally that different nodes do utilize attention to select different neighborhoods, indicating that it may be useful to consider various motif-defined neighborhoods.

References

Abu-El-Haija, S.; Kapoor, A.; Perozzi, B.; and Lee, J. 2018. N-GCN: Multi-scale graph convolution for semi-supervised node classification. In *arXiv:1802.08888v1*.

Ahmed, N. K.; Neville, J.; Rossi, R. A.; Duffield, N.; and Willke, T. L. 2017. Graphlet decomposition: Framework, algorithms, and applications. *KAIS* 50(3):689–722.

Ahmed, N. K.; Rossi, R. A.; Zhou, R.; Lee, J. B.; Kong, X.; Willke, T. L.; and Eldardiry, H. 2018. Learning role-based graph embeddings. In *StarAI @ IJCAI*, 1–8.

Bahdanau, D.; Cho, K.; and Bengio, Y. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*, 1–15.

Belkin, M.; Niyogi, P.; and Sindhwani, V. 2006. Manifold

regularization: A geometric framework for learning from labeled and unlabeled examples. *JMLR* 7:2399–2434.

Choi, E.; Bahadori, M. T.; Song, L.; Stewart, W. F.; and Sun, J. 2017. Gram: Graph-based attention model for healthcare representation learning. In *KDD*, 787–795.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, 3837–3845.

Duvenaud, D. K.; Maclaurin, D.; Aguilera-Iparraguirre, J.; Gomez-Bombarelli, R.; Hirzel, T.; Aspuru-Guzik, A.; and Adams, R. P. 2015. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, 2224–2232.

Feichtenhofer, C.; Pinz, A.; and Zisserman, A. 2016. Convolutional two-stream network fusion for video action recognition. In *WSDM*, 601–610.

Frasconi, P.; Gori, M.; and Sperduti, A. 1998. A general framework for adaptive processing of data structures. *IEEE TNNLS* 9(5):768–786.

Friggeri, A.; Chelius, G.; and Fleury, E. 2011. Triangles to capture social cohesion. In *SocialCom/PASSAT*, 258–265.

Gori, M.; Monfardini, G.; and Scarselli, F. 2005. A new model for learning in graph domains. In *IJCNN*, 729–734.

Han, X.; Liu, Z.; and Sun, M. 2018. Neural knowledge acquisition via mutual attention between knowledge graph and text. In *AAAI*, 1–8.

Hausknecht, M., and Stone, P. 2015. Deep recurrent Q-learning for partially observable MDPs. In *AAAI Fall Symposium*, 1–9.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

Henaff, M.; Bruna, J.; and LeCun, Y. 2015. Deep convolutional networks on graph-structured data. In *arXiv:1506.05163v1*.

Kersting, K.; Kriege, N. M.; Morris, C.; Mutzel, P.; and Neumann, M. 2016. Benchmark data sets for graph kernels.

Kim, J.; Lee, J. K.; and Lee, K. M. 2016. Deeply-recursive convolutional network for image super-resolution. In *CVPR*, 1637–1645.

Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*, 1–14.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet classification with deep convolutional neural networks. In *NIPS*, 1106–1114.

Lee, J. B.; Rossi, R.; and Kong, X. 2018. Graph classification using structural attention. In *KDD*, 1666–1674.

Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. 2016. Gated graph sequence neural networks. In *ICLR*, 1–20.

Lu, Q., and Getoor, L. 2003. Link-based classification. In *ICML*, 496–503.

Milo, R.; Shen-Orr, S.; Itzkovitz, S.; Kashtan, N.; Chklovskii, D.; and Alon, U. 2002. Network motifs: Simple building blocks of complex networks. *Science* 298(5594):824–827.

Mnih, V.; Heess, N.; Graves, A.; and Kavukcuoglu, K. 2014. Recurrent models of visual attention. In *NIPS*, 2204–2212.

Monti, F.; Boscaini, D.; Masci, J.; Rodola, E.; Svoboda, J.; and Bronstein, M. M. 2016. Deep convolutional networks on graph-structured data. In *arXiv:1611.08402*.

Nguyen, T. H., and Grishman, R. 2018. Graph convolutional networks with argument-aware pooling for event detection. In *AAAI*, 5900–5907.

Paranjape, A.; Benson, A. R.; and Leskovec, J. 2017. Motifs in temporal networks. In *CVPR*, 1933–1941.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *KDD*, 701–710.

Prill, R. J.; Iglesias, P. A.; and Levchenko, A. 2005. Dynamic properties of network motifs contribute to biological network organization. *PLoS Biology* 3(11):1881–1892.

Rossi, R. A.; Ahmed, N. K.; and Koh, E. 2018. Higher-order network representation learning. In *WWW*, 3–4.

Rossi, R. A.; Zhou, R.; and Ahmed, N. K. 2018a. Deep inductive network representation learning. In *BigNet @ WWW*, 1–8.

Rossi, R. A.; Zhou, R.; and Ahmed, N. K. 2018b. Estimation of graphlet counts in massive networks. In *TNNLS*, 1–14.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The graph neural network model. *IEEE TNNLS* 20(1):61–80.

Sperduti, A., and Starita, A. 1997. Supervised neural networks for the classification of structures. *IEEE TNNLS* 8(3):714–735.

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In *ICLR*, 1–12.

Weston, J.; Ratle, F.; Mobahi, H.; and Collobert, R. 2012. *Deep Learning via Semi-supervised Embedding*. Springer. 639–655.

Xu, K.; Ba, J.; Kiros, R.; Cho, K.; Courville, A. C.; Salakhutdinov, R.; Zemel, R. S.; and Bengio, Y. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2048–2057.

Yan, S.; Xiong, Y.; and Lin, D. 2018. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI*, 3482–3489.

Yang, C.; Liu, M.; Zheng, V. W.; and Han, J. 2018. Node, motif and subgraph: Leveraging network functional blocks through structural convolution. In *ASONAM*, 1–8.

Yang, Z.; Cohen, W. W.; and Salakhutdinov, R. 2016. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 40–48.

Zhou, G.-B.; Wu, J.; Zhang, C.-L.; and Zhou, Z.-H. 2016. Minimal gated unit for recurrent neural networks. *IJAC* 13(3):226–234.

Zhu, X.; Ghahramani, Z.; and Lafferty, J. D. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, 912–919.