1 Supplementary Material

1.1 Graph Attention Network (GAT) modifies a message-passing neural network by modifying the weights on the edges of the graph. This changes how much each neighbor j of a node i plays a role in the output of node i.

$$e\left(\boldsymbol{h}_i, \boldsymbol{h}_j\right) = \text{ LeakyReLU } \left(\boldsymbol{a}^{\top} \cdot \left[\boldsymbol{W}\boldsymbol{h}_i \| \boldsymbol{W}\boldsymbol{h}_j\right]\right).$$

GAT assigns edge weights in light of the corresponding linked nodes' features, and the pattern of learning edge weight about GAT is a typical neighborhood-based strategy as it apply a softmax function among all the immediate-neighbors.

$$\alpha_{ij} = \operatorname{softmax}_j\left(e\left(\boldsymbol{h}_i, \boldsymbol{h}_j\right)\right) = \frac{\exp\left(e\left(\boldsymbol{h}_i, \boldsymbol{h}_j\right)\right)}{\sum_{j' \in \mathcal{N}_i} \exp\left(e\left(\boldsymbol{h}_i, \boldsymbol{h}_{j'}\right)\right)}.$$

Tanks to the addition of softmax function, each central node is a weighted sum of the immediate-neighbors' features. If the edge weight which linked the node i and node j learned as $\alpha_{ij} = e\left(Wh_i, Wh_j\right)$, then the learned representation can be shown as:

$$\boldsymbol{h}_i = \sigma\left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \cdot \boldsymbol{W}\boldsymbol{h}_j\right).$$

However, e_{ij} is only computed for neighbors in GAT, meaning that what we really calculate is:

$$\alpha_{ij} = e\left(Wh_i, Wh_j\right) \circ A_{ij}$$

where $\circ$ denotes element-wise multiplication and A is the unweighted adjacency matrix of the graph. The attention mechanism is a function over neighbors only. However, the given topology may be not appropriate to capture distinguishable information especially for strong heterophily graphs.

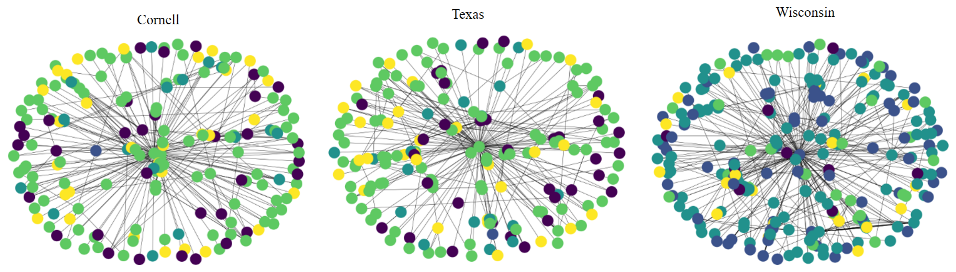

Figure 1: Visualization of three heterophily data-sets, colors indicate different node labels.

1.2 Experiments

1.2.1 Baselines. We compare our work with some state-of-the-art methods, covering some graph convolutional networks based methods.

• GCN is a typical model which learns node representations by entangling topology structure and node features, and GCN can be termed as a simple variation of ChebNet as it only construct two layers to learn node presentation. The project of GCN and ChebNet can be obtained from https://github.com/tkipf/gcn

• SGC is a simper model based on GCN. It removes the nonlinear activation and only reserve the 'Softmax' in the last layer, and it needs less time consumption without decreasing the accuracy results compared to GCN. The project can be obtained from https://github.com/Tiiiger/SGC

• GraphHeat captures the smoothness over graph by suppressing high-frequency signals, which acted as a low-pass filter. The project can be obtained from https://github.com/Eilene/GraphHeat

• GraphSAGE is a classical model which utilizes neighbor selection to learn node representation. The project can be obtained from https://github.com/williamleif/GraphSAGE

• GAT applies self-attention mechanism in graphs, and it uses non-negative weights to aggregate node features from neighbors. The project can be obtained from https://github.com/Diego999/pyGAT

• GCNII adds initial residual and identity mapping to deepen the convolutional layers, and it can achieve better accuracy among some assortative and disassortative data-sets than GCN. The project can be obtained from https://github.com/chennnM/GCNII

• AM-GCN pay attention to features fusion strategy through constructing a new topology with using KNN to make up the weakness of the given topology, and it employs a multi-channel GCN framework to learn some representations individually and then use a attention mechanism to learn importance weights of the learned representations. The project can be obtained from https://github.com/zhumeiqiBUPT/AM-GCN

• BernNet learning graph spectral filters via Bernstein approximation, which can be acted as low-pass filter or high-pass in different data-sets. The project can be obtained from https://github.com/ivam-he/BernNet

• SimP-GCN employs self-supervised learning to explicitly capture the complex feature similarity relations between nodes. The project can be obtained from https://github.com/ChandlerBang/SimP-GCN

1.2.2 Implementation.

We investigated our EAEAGCN from the view of edge-oriented attention, and the model is constructed with using DGL API. DGL is an open-source in-memory tensor structure specification for sharing tensors among deep learning frameworks, and it takes advantage of DLPack to directly process and return DL framework tensors without data copying. Many frameworks, including Pytorch, MXNet, and TensorFlow, natively support DLPack. The above functionality calls for memory allocation and management. DGL delegates memory management to the base frameworks. A base framework usually implements sophisticated memory management, which is especially important for GPU memory. Because the output shape of a graph kernel is well determined before execution, DGL calculates the output memory size, allocates memory from the frameworks for the output tensors and pass the memory to the kernel for execution. We conduct MAGCN model in Dgl-Pytorch version. All experiments are conducted on a Linux server with GPU. The Python and PyTorch versions are 3.7 and 1.4.0, respectively. The architecture are trained on a single NVIDIA GTX 1080 Ti GPU with 12GB memory.

1.2.3 Number of Parameters

For our method, the number of parameters is $O((l+2)d^2+M)$, i.e., $O(d^2+M)$ where l is the number of edge-oriented convolution layers, and d is the dimensionality of the embeddings, and M is the edge number of the new topology. For the baselines including GCN and GAT the time complexity is also $O(d^2+M)$. Notice that SimP-GCN, one of the baselines, has $O(n^2)$ time complexity due to the adjacency matrix reconstruction each iteration in the end-to-end framework. Although our model also make adjacency matrix reconstruction, we construct this step out-of the training repeat iterations. Therefore, the number of parameters of the proposed method and the GCN and GAT are comparable. And more works can be done for investigating the other efficient topology optimization strategy.

1.2.4 Hidden dimension sensitivity

We show the change of accuracy with hidden dimension d in Figure. 2. Solid lines represent the performance among assortative data-sets and dashed line represent the performance among disassortative data-sets. In our experiments, we can find that the change of accuracy for assortative graphs does not so obvious with the increasing of features dimension. While the accuracy among disassortative graphs are much affected by the dimension of hidden features, as the dimension becoming larger and larger, the accuracy lines tend to increase first and get the best performance in $d = 64$, then they begin to decrease roughly.

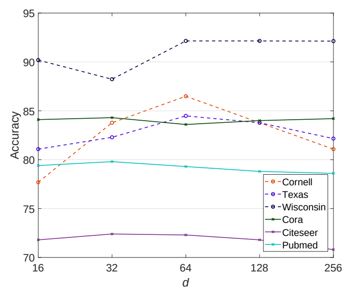

Figure 2: The change of accuracy with d.