Graph Convolutional Networks for Graphs with Multi-Dimensionally Weighted Edges

Zhengdao Chen

July 20, 2020

1 Introduction

Graph neural networks (GNNs) are able to exploit relational structures in data that are presented in the form of graphs. A graph typically consists of nodes, which can represent agents, websites, atoms, concepts, etc., and edges, which represent pair-wise relations between nodes, such as connections in social networks, hyperlinks between websites, chemical bonds between atoms and conceptual relations. A graph may further contain node and edge features, which are typically represented by vectors. A GNN can take advantage of such information in learning in order to predict properties of the nodes, the existence of missing edges or properties of the entire graph.

Analogous to convolution operations on grid space, which underlie the immense empirical success of Convolutional Neural Networks (CNNs) on image data, convolution operations on graphs have been studied with the mathematics of spectral graph theory and give rise to a series of popular models (Bruna et al., 2013; Defferrard et al., 2016). In particular, Kipf and Welling (2016) propose simplifications of spectral graph convolutions via approximations to the Chebyshev polynomials of the graph Laplacian, resulting in the Graph Convolution Network (GCN) model, which has since become popular. It has been generalized to the Relational Graph Convolutional Network (RGCN) applied to data on relational graphs, where nodes can be connected to each other via multiple relations (Schlichtkrull et al., 2018).

One direction of research that remains interesting is how to exploit edge features properly. Extending from graph convolutions to general message passing on graphs, the framework of Message Passing Neural Networks (MPNNs) indeed allows the incorporation of edge features into model (Gilmer et al., 2017). Nonetheless, many of its edge-feature-aware variants, such as the ones involving the edge networks, are computationally expensive. Inspired by Graph Attention Networks (GATs) (Veličković et al., 2017), it is natural to consider edge features as playing a similar role as attention coefficients that modulate the influences of neighboring nodes in the update of node features (Gong and Cheng, 2019). When edge features have multiple channels (i.e. the edge features are represented by multi-dimensional vectors), however, the design of the convolution and the aggregation of information across the different channels both become crucial questions.

To address this goal, we propose *Graph Convolutional Networks for Multi-dimensionally Weighted Edges* (*MWE-GCN*).

2 Model

2.1 Notations

Let G be a graph with N nodes. The node features are represented by an $N \times F$ matrix, X, where X_{ik} gives the kth entry of the F-dimensional feature vector of the ith node in the graph. The edge weights are represented by an $N \times N \times P$ tensor, E, where E_{ijp} gives the pth channel of the P-dimensional weight vector of the edge (i, j). If the edge (i, j) does not exist, we set $E_{ij\cdot} = 0$. Note that both directed edges and undirected edges can be represented in this way. Assuming below that the edges are undirected, we use D to

denote the $N \times N$ diagonal degree matrix, where D_{ii} gives the degree of the ith node. We will omit the bias terms of the linear layers in our exposition.

2.2 Architecture

2.2.1 Normalization of Edge Weights

To begin with, an optional preprocessing step is to normalize the edge features. We propose to normalize them using the node degrees in a symmetric fashion, which is similar to the normalization performed in GCNs except for replacing the adjacency matrix with the edge weight tensor. Mathematically, we compute

$$\tilde{E}_{ijp} = D_{ii}^{-\frac{1}{2}} D_{jj}^{-\frac{1}{2}} E_{ijp}, \tag{1}$$

or, in matrix notations,

$$\tilde{E}_{\cdot\cdot p} = D^{-\frac{1}{2}} \cdot E_{\cdot\cdot p} \cdot D^{-\frac{1}{2}} \tag{2}$$

2.2.2 Convolution with Edge Weights as Coefficients

We next describe the architecture of the GNN model. We use $H^{(l)}$ to denote the $N \times d^{(l)}$ matrix of hidden node states at the lth layer, whose ith row, $H_{i\cdot}^{(l)}$ is the $d^{(l)}$ dimensional hidden state vector associated with the ith node in the lth layer. At $l = 0$, we initialize $H^{(0)} = X$. For $l \geq 0$, the hidden node states of the $(l+1)$th layer are obtained iteratively in the following way. For each of the P channels of the edge weights, we first perform a weighted convolution operation on $H^{(l)}$ with $\tilde{E}_{\cdot\cdot p}$ as the convolution coefficients together with a usual weight matrix, $W^{(l,p)}$, of dimension $d^{(l)} \times \hat{d}^{(l)}$. That is, for each $p \in \{1, ..., P\}$, we compute an $N \times \hat{d}^{(l)}$ matrix, $\hat{H}^{(l,p)}$, by

$$\hat{H}^{(l,p)} = \sigma(\tilde{E}_{\cdot\cdot p} \cdot H^{(l)} \cdot W^{(l,p)}), \tag{3}$$

where σ is a pointwise nonlinear activation function.

2.2.3 Aggregation across Edge Weight Channels

We then aggregate information across the different edge channels by combining $\hat{H}^{(l,p)}$ for $p \in \{1, ..., P\}$ into the hidden node states of the next layer, $H^{(l+1)}$. To this end, we propose two different aggregation strategies:

- Sum-Aggregation

$$H^{(l+1)} = \sigma\Big(\sum_{p=1}^{P} \hat{H}^{(l,p)} \cdot W_{sum}^{(l)}\Big), \tag{4}$$

  where $W_{sum}^{(l)}$ is a learnable weight matrix of size $\hat{d}^{(l)} \times d^{(l+1)}$.

- Concat-Aggregation

$$H^{(l+1)} = \sigma\Big(\Big|\Big|_{p=1}^{P} \hat{H}^{(l,p)} \cdot W_{cat}^{(l)}\Big), \tag{5}$$

  where $\Big|\Big|_{p=1}^{P}$ denotes the concatenation along the p dimension, and $W_{cat}^{(l)}$ is a learnable weight matrix of size $(P\hat{d}^{(l)}) \times d^{(l+1)}$.

Therefore, for the variant using sum-aggregation (which we call MWE-GCN-s,) the entire update rule can be written as

$$H^{(l+1)} = \sigma\Big(\sum_{p=1}^{P} \sigma(\tilde{E}_{\cdot\cdot p} \cdot H^{(l)} \cdot W^{(l,p)}) \cdot W_{sum}^{(l)}\Big), \tag{6}$$

whereas for the variant using concat-aggregation, (which we call MWE-GCN-c,) the entire update rule can be written as

$$H^{(l+1)} = \sigma\Big(\Big|\Big|_{p=1}^{P} \sigma(\tilde{E}_{\cdot\cdot p} \cdot H^{(l)} \cdot W^{(l,p)}) \cdot W_{sum}^{(l)}\Big), \tag{7}$$

Finally, in the last layer $l = L$, we pass $H^{(L)}$ through a linear layer to obtain node-level outputs or a global pooling layer to obtain a graph-level output.

2.2.4 A Deeper Version

Concurrently to our work, Li et al. (2020) proposes the DeeperGCN model that takes advantage of a large number of layers as well as techniques including layer normalization (Ba et al., 2016) and residual connections (He et al., 2016) to achieve state-of-the-art performances on several datasets. Inspired by this work, we also implement a deeper version of our model with layer normalization and residual connections added, which uses sum-aggregation and does not perform edge weight normalization. We call this model MWE-DGCN-s, and will compare its performance against DeeperGCN.

3 Numerical Experiments

We perform numerical experiments on the *ogbn-proteins* dataset from the Open Graph Benchmark (OGB) (Hu et al., 2020). The dataset contains an undirected graph with edge weights, where each node represents a protein, and each edge represents biologically meaningful associations between two proteins. Each edge has a weight vector of 8 dimensions, where each entry represents the strength of one of the 8 types of associations and takes value between 0 and 1, with larger values corresponding to stronger associations. The task is to predict 112 binary labels of the nodes that represent protein functions. MWE-GCN-s, MWE-GCN-c and RGCN are all implemented with 4 layers and 100 hidden units in each hidden layer. MWE-DGCN-s is implemented with 15 layers and 64 hidden units in each hidden layer. The models are implemented in Python with PyTorch (Paszke et al., 2019) and the Deep Graph Library (Wang et al., 2019), and the code can be found at https://github.com/dmlc/dgl/tree/master/examples/pytorch/ogb/ogbn-proteins.

3.1 Results

The experimental results are presented in Table 1. We see that our models outperform all the other baseline models except for DeeperGCN (Li et al., 2020) and DeepGCN (Li et al., 2019) by significant margins. The deeper version of our model has slightly worse test performance than DeeperGCN and DeepGCN but also contains fewer parameters: MWE-DGCN-s has 538544 parameters, while DeepGCN has 2374456 and DeeperGCN has 2374568, as reported on the OGB leaderboard (Hu et al., 2020).

Model	Test ROC-AUC
DeeperGCN†	0.8580 ± 0.0017
DeepGCN†	0.8486 ± 0.0028
GeniePath-BS†	0.7825 ± 0.0035
GaAN†	0.7803 ± 0.0073
RGCN	0.7859 ± 0.4
GraphSAGE†	0.7768 ± 0.0020
GCN†	0.7251 ± 0.0035
MLP†	0.7204 ± 0.0048
Node2vec†	0.6881 ± 0.0065
MWE-GCN-s	0.8147 ± 0.0083
MWE-GCN-c	0.8167 ± 0.0017
MWE-DGCN-s	0.8436 ± 0.0065

Table 1: Performance of various models on the *ogbn-proteins* dataset. MWE-GCN-s and MWE-GCN-c refer to two variants of our model, using sum-aggregation and concat-aggregation, respectively. †: Reported results on the OGB leaderboard (Hu et al., 2020) as of July 20th, 2020.

3.2 Ablation Study - Value of the Inner Nonlinearity

If we omit the nonlinear activation function in the convolution step (3), then our model coincides with RGCN in the case where all weights have value either 0 and 1. Therefore, to investigate the difference, we perform

ablation experiments with the two variants of our model (using sum- and concat-aggregation, respectively) in which the convolution step is (3) replaced by

$$\hat{H}^{(l,p)} = \tilde{E}_{\cdot\cdot p} \cdot H^{(l)} \cdot W^{(l,p)} \tag{8}$$

We see from Table 2 that though these two variants still perform competitively on *ogbn-proteins*, their performances are significantly lower than those of MWE-GCN-s and MWE-GCN-c.

Variant	Train	Valid	Test
MWE-GCN-s	0.9249 ± 0.0054	0.8769 ± 0.0060	0.8147 ± 0.0083
MWE-GCN-c	0.9203 ± 0.0033	0.8757 ± 0.0025	0.8167 ± 0.0017
Variant-s	0.8919 ± 0.0130	0.8536 ± 0.0088	0.7908 ± 0.0047
Variant-c	0.8802 ± 0.0273	0.8398 ± 0.0214	0.7773 ± 0.0120

Table 2: Results of the ablation experiments on the *ogbn-proteins* dataset.

4 Related Works

A plethora of GNN models allow the exploitation of edge weights. One type of solution is to concatenate edge weights into node features (Duvenaud et al., 2015; Battaglia et al., 2016), which is not necessarily a strategy with high expressive power. Another type of solution is to learn from the edge weights a transformation that is then applied to the hidden node states Gilmer et al. (2017); Li et al. (2015). While achieving high expressive power, this strategy typically leads to high computational complexity. A third type of solution takes advantage of the line graph of the original graph, but is also hard to scale to large graphs (Chen et al., 2019). A model that similar to ours in spirit is the Edge Graph Neural Networks (EGNN) model proposed by Gong and Cheng (2019). Our model differs with EGNN in the normalization of the edge weights, the aggregation across the channels, as well as the absence of weight sharing across the channels. In particular, the last two decisions are important for enhancing the overall expressive power.

5 Acknowledgements

The author greatly appreciates the helps from and the discussions with Mufei Li, Minjie Wang, Xiang Song and Lingfan Yu, and would also like to thank Joan Bruna and Lei Chen for the conversations and feedbacks.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Battaglia, P., Pascanu, R., Lai, M., Rezende, D. J., et al. (2016). Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.

Chen, Z., Li, L., and Bruna, J. (2019). Supervised community detection with line graph neural networks. *Internation Conference on Learning Representations*.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. (2015). Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org.

Gong, L. and Cheng, Q. (2019). Exploiting edge features for graph neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9211–9219.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. (2020). Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Li, G., Muller, M., Thabet, A., and Ghanem, B. (2019). Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9267–9276.

Li, G., Xiong, C., Thabet, A., and Ghanem, B. (2020). Deepergcn: All you need to train deeper gcns. *arXiv preprint arXiv:2006.07739*.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2015). Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035.

Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I., and Welling, M. (2018). Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607. Springer.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., Li, M., Zhou, J., Huang, Q., Ma, C., et al. (2019). Deep graph library: Towards efficient and scalable deep learning on graphs. *arXiv preprint arXiv:1909.01315*.