A Higher-Order Graph Convolutional Layer

Sami Abu-El-Haija[1], **Nazanin Alipourfard**[1], **Hrayr Harutyunyan**[1],
Amol Kapoor[2], **Bryan Perozzi**[2]

[1]Information Sciences Institute
University of Southern California

[2]Google AI
New York City, NY

{haija, nazanina, hrayrh}@isi.edu, {ajkapoor, bperozzi}@google.com,

Abstract

Recent methods generalize *convolutional layers* from Euclidean domains to graph-structured data by approximating the eigenbasis of the graph Laplacian. The computationally-efficient and broadly-used Graph ConvNet of Kipf & Welling [11], over-simplifies the approximation, effectively rendering graph convolution as a *neighborhood-averaging operator*. This simplification restricts the model from learning *delta operators*, the very premise of the graph Laplacian. In this work, we propose a new Graph Convolutional layer which mixes multiple powers of the adjacency matrix, allowing it to learn delta operators. Our layer exhibits the same memory footprint and computational complexity as a GCN. We illustrate the strength of our proposed layer on both synthetic graph datasets, and on several real-world citation graphs, setting the record state-of-the-art on Pubmed.

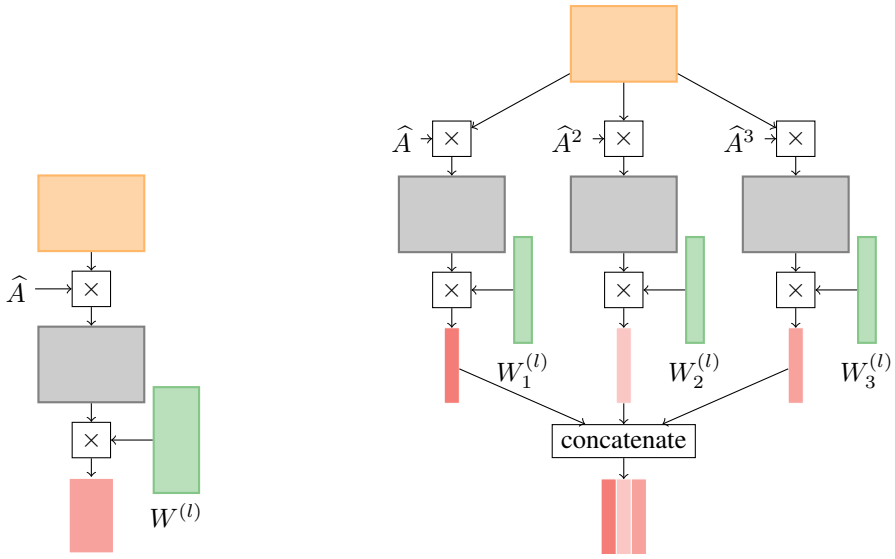
1 Introduction

Convolutional Neural Networks (CNNs) establish state-of-the-art performance on many Computer Vision applications [12, 8, 17]. CNNs consist of a series of *convolutional layers*, each is parameterized by a *filter* with pre-specified spatial dimensions. CNNs are powerful because they are able to learn a hierarchy of feature detectors that is invariant to translations. Visualization experiments show that the first layer learns oriented *edge detectors*. Higher layers can then learn a hierarchical representation of these features to represent objects e.g. see [13, 12] for visualizations.

The success of CNNs on Computer Vision and other domains has motivated researchers [4, 5, 11] to extend the convolutional operator from regular grids, in which the structure is fixed and repeated everywhere, to graph-structured data, where nodes' neighborhoods can greatly vary in structure across the graph. Generalizing Convolution to graph structures should allow models to learn location-invariant features. Consider a motivating example of predicting interests of users in an online social network. Given user features (e.g. age, education) as well as their relationships (e.g. friend lists), convolution should help to generalize "feature detectors" across graph nodes. For example, if user v has similar features to, but shares no friends with user u, they should have similar representations if their neighbors' features are also similar.

The early extension of convolution onto graph-structed data [4], albeit theoretically motivated, is not scalable to large graphs, requiring quadratic computational complexity in number of nodes. In addition, it requires the graph to be completely observed during training, targeting the *transductive* setting. Defferrard et al [5], Kipf & Welling [11] propose *approximations* to Graph Convolution that are computationally-efficient (linear complexity, in the number of edges), and can be applied in *inductive* settings, where the test graph is not observed during training. In this work, we study the model of [11] because for its practical value: it is relatively-simple to implement and fast to

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.



(a) Graph Convolution layer of [11].

(b) Our proposed layer

Figure 1: Architectures of the Graph Convolution (GC) layer proposed by Kipf & Welling [11] (left) compared with ours (right). Orange denotes the input activation matrix, one row per node, gray denotes adjacency-times-input, red denotes layer output, and finally, trainable parameters are in green. We highlight left- VS right-multiplication by the relative poisiton to the $\times$ operator. In all our experiments, we set the total size of our model parameters $W_1^{(l)}$, $W_2^{(l)}$, and $W_3^{(l)}$, equal to the size of baseline model parameters $W^{(l)}$. For saving vertical space, we skip depicting the element-wise activation σ that is applied after the (red) output matrix in both models.

train. However, we try to circumvent the reduced modeling capacity made by the *approximations*. Specifically, ones that prevent the model from capturing patterns analogous to *edge*[1] *detectors* in Computer Vision. **Our main contribution** is a new Graph Convolutional layer that mixes powers of the adjacency matrix. Our model exhibits the same memory and computational complexity of [11], but additionally allows us to learn delta operators, which was not possible with the model of [11]. The strength of our method is demonstrated on a number of synthetic and real-world citation graphs.

2 Background

2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) establish state-of-the-art performance on many applications in Computer Vision [12, 8, 17]. CNNs consist of a series of *convolutional layers*. Each layer calculates a *location-equivarient* transformation. This equivariance allows CNNs to generalize across the spatial dimensions. For example, observing a cat breed during training, should allow CNNs to classify the cat, even if it appears in different image position at test time. The convolution operator computes for each pixel, a representation from a patch surrounding the pixel. The size of the patch is determined by the support of the convolutional filter that parametrizes the layer.

2.2 Graph Convolutional Networks

Model: We review the Graph Convolutional Network proposed by Kipf & Welling [11]. They define the Graph Convolutional (GC) Layer, as:

$$H^{(l+1)} = \sigma(\widehat{A}H^{(l)}W^{(l)}), \tag{1}$$

[1]Edge in vision refer to adjacent patches with different colors. In the graph's analogy, we do not refer to an edge between two nodes, but refer to nodes at the boundary of sub-graphs with different neighborhood structures.

Model	2k-0.5	2k-0.9	5k-0.5	5k-0.9
GCN [11]	48.5	88.8	42.2	82.7
P=$\{1, 2\}$	48.4	98.8	43.5	92.6
P=$\{0, 1, 2\}$	53.0	95.1	42.9	97.4
P=$\{0, 1, 3\}$	**54.3**	**100.0**	45.6	**99.9**
P=$\{1, 2, 3\}$	51.2	98.4	46.9	**99.9**
P=$\{1, 2, 3, 4\}$	48.7	98.7	**47.4**	98.9

Dataset	N	M	d_0	Classes
Citeseer	3,327	4,732	3,703	6
Cora	2,708	5,429	1,433	7
Pubmed	19,717	44,338	500	3

(a) **Top:** Classification accuracy on the four synthetic datasets. **Bottom:** statistics of citation datasets, where d_0 denotes the node features dimensionality

Model	Citeseer	Cora	Pubmed
ManiReg [3]	60.1	59.5	70.7
SemiEmb [19]	59.6	59.0	71.1
LP [21]	45.3	68.0	63.0
DeepWalk [16]	43.2	67.2	65.3
ICA [15]	69.1	75.1	73.9
Planetoid [20]	64.7	75.7	77.2
GCN [11]	70.3	81.5	79.0
P=$\{1, 2, 3\}$	**71.2**$\pm$0.94	**81.6**$\pm$0.47	80.0$\pm$0.64
P=$\{1, 2, 3, 4\}$	**71.2**$\pm$0.84	**81.6**$\pm$0.63	**80.1**$\pm$0.65

(b) Node Classification Accuracy on the citation datasets generated by [20]. $\pm$ represents standard deviation, each corresponding to 50 runs with different random initializations.

Table 1: Experiments on synthetic and citation graph datasets. Our models are denoted P=$\{\dots\}$.

Where $H^{(l)} \in \mathbb{R}^{N \times d_l}$ and $H^{(l+1)} \in \mathbb{R}^{N \times d_{l+1}}$ are the input and output activations for layer l. N denotes the number of graph nodes. $W^{(l)} \in \mathbb{R}^{d_l \times d_{l+1}}$ is a trainable weights matrix and σ is element-wise non-linearity. $\widehat{A}$ is a symmetrically normalized adjacency matrix with self-connections. It can be constructed by first assembling the binary adjacency matrix $A \in \{0, 1\}^{N \times N}$, add self-connections: $A := A + I_N$, calculate node degrees $d \in \mathbb{R}^N$ as $d_i = \sum_j A_{ij}$, place in a diagonal matrix: $D = \text{diag}(d)$, then apply the symmetric normalization to produce $\widehat{A}$ as $\widehat{A} = D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. A GCN model with L layers can be constructed by setting $H^{(0)} = X$, applying the GC layer (Equation 1) L times for $l = 0, 1, \dots, L-1$, then set the output of the GCN as $Y = H^{(L)}$, with $Y \in \mathbb{R}^{N \times d_L}$.

Popularity: The rapid widespread of the GCN model of [11] can be attributed to many factors, including: (1) its linear computional complexity in the number of edges M, (2) it can be converted into a stochastic (mini-batch) settings [7], and (3) it is relatively simple to program using computational software frameworks including TensorFlow [6]. Nonetheless, the **main issue** we address in this paper, is that GCN of [11] is an over-simplification of the graph convolution, defined as multiplication in the Graph Fourier Basis (i.e. the eigenvectors of the Graph Laplacian) [4, 5, 11]. Concretely, the simplifications of [11] effectively render the GC layer of [11] as a neighborhood-averaging operator.

We summarize three simplifying assumptions of [11]. First, it is a rank-2 approximation to multiplication in the Graph Fourier basis, defined to be the eigenbasis of the graph Laplacian; Second, they assume that the two co-efficients to the Chebyshev polynomials multiply to -1, which is then later justified by their, third, *renormalization trick*, of adding the self-connections (identity matrix) to A before, rather than after, normalization; Altogether, these simplifications are well motivated: for computational efficiency and for preventing exploding/vanishing gradients, but the second and third simplifications restricts their convolution to be a neighborhood-averaging operator. Even though GraphSAGE [7], propose replacing the averaging with arbitrary aggregation, their results report negligible wins compared to averaging, even with expressive aggregators such as LSTM [9].

2.3 Semi-supervised Node Classification

We evaluate our model in semi-supervised node classification tasks. That is, only a portion of the node labels are given. For training a GCN model on such a task, it is possible to select row slices from Y, corresponding to nodes with known labels, on which a loss and its gradients can be evaluated. The gradient of the loss can be backpropagated, through the GC layers, each multiplying by the transpose of $\widehat{A}$, effectively spreading the gradients from labeled to unlabeled examples.

3 Our Proposed Architecture

Motivation: Unlike the model of [4], if we convert the model of [11] back to the Euclidean domain, it will no longer be able to learn the *oriented edge detectors*. Relating to the motivating example of online social networks, it can be informative to detect users that live around the "boundary" of social circles, that is: their immediate friends mostly exhibit some features, but the friends of their friends exhibit different features. In this work, we are motivated to design a layer that works on *patches*

centered around the node, which includes 1-hop, 2-hop, ... neighbors in distinct feature spaces such, that they can be effectively combined.

Model: We propose replacing the Graph Convolution layer (GC) of [11], defined in Equation 1, with:

$$H^{(l+1)} = \Big\Vert_{j \in P} \sigma\left(\widehat{A}^j H^{(l)} W_j^{(l)}\right), \tag{2}$$

where P is a set of integers, the powers that we consider (Figure 1b corresponds to $P = \{1, 2, 3\}$) , $\widehat{A}^j$ denotes the matrix $\widehat{A}$ multiplied by itself j times, and $\Vert$ denotes column-wise concatenation. Each layer now contains $|P|$ distinct parameter matrices. However, in our experiments, we used *thin* W_j's e.g. $W_j \in \mathbb{R}^{\cdot \times 10}$, so that the total number of parameters is the same as vanilla GCN's.

Computational Complexity: It is key to our algorithm that we never compute $\widehat{A}^j$. Instead, we multiply $\widehat{A}^j H^{(l)}$ right-to-left. Specifically, if $j = 3$, we calcluate $\widehat{A}^3 H^{(l)}$ as $\widehat{A}\left(\widehat{A}\left(\widehat{A}H^{(l)}\right)\right)$. Since we store $\widehat{A}$ as a sparse matrix with M non-zero entries, this chain of multiplications takes $\mathcal{O}(j \times M \times d_l)$. An efficient dynamic-programming implementation of our layer (Equation 2) takes $\mathcal{O}(j_{\text{max}} \times M \times d_l)$ computational time, where j_{max} is the largest element in P. Under the realistic assumptions of $j_{\text{max}} \ll M$ and $d_l \ll M$, running an L-layer model takes $\mathcal{O}(LM)$ computational time, matching the vanilla GCN [11].

Representational Capability: Since each layer outputs the multiplication of different adjacency powers in different columns, the next layer's weights (when multiplied from the right) can learn abitrary linear combinations of the columns. As such, it is able to assign positive coefficient to columns produced by some $\widehat{A}$ power and negative to another, allowing it to learn a *delta operator*. Such a representational capability is not possible with vanilla GCNs, even if they were deep, as their depth only allows them to average from more neighborhoods but in no way their model can learn the difference (in the feature space) between immediate and further neighbors.

Related Work: Other methods also mix powers of the adjaceny matrix, including [1, 2] which combine the powers at the end of the network (right before classification layer), and [14] which combines them at the input of the network. In our work, we mix the powers at every layer, enabling our method to learn delta operators.

4 Experimental Results

Datasets: As summarized in Table 1, We conduct node classification experiments on synthetic and real-world datasets. Our synthetic datasets are generated in the manner of [10]. We generate 4 graphs: 2k-0.5, 2k-0.9, 5k-0.5 and 5k-0.9. The 2k-* each contain 2000 nodes and 4 classes, and the 5k contain 5000 nodes and 10 classes. The fraction indicate the likelihood that a node forms a connection to another with the same label (ie, the network's *homophily* coefficient: 0.5 or 0.9). The features for all synthetic nodes were sampled from overlapping multi-gaussian distributions. We randomly partition each graph nodes into train, test, and validation splits, all of equal size. The experiments with real-world datasets follow the methodology proposed in [20].

Training: For all experiments, we construct a 2-layer network of our model using TensorFlow [6]. We train our models, using Gradient Descent optimizer, for a maximum of 1000 steps, with initial learning rate of 0.005 that decays by 15% every 20 steps. As we use early-stopping, specifically terminate training if validation accuracy does not improve for 40 consecutive steps, most runs finish in less than 200 steps. We use 5×10^{-4} L2 regularization on the first layer, and 50% dropout. We note that the citation datasets are extremely sensitve to initializations, with some runs train accuracy quickly approaching 100% but with bad validation and test accuracies. For that, we run all models 100 times, sort by the validation accuracy, then report the test accuracy for the top 50 runs. For all model configurations (GCN and ours), we use latent dimension of 30 i.e. for GCN, $W \in \mathbb{R}^{d_0 \times 30}$ for the first layer and for us, we divide the 30 evenly to all $|P|$ powers. We plan to release our synthetic datasets and code so that others can reproduce and extend our method.

5 Conclusion

We propose a graph convolutional layer that utilizes multiple powers of the adjacency matrix. Repeated application of this layer into a graph convolutional network model, allows the model to learn both averaging and delta operators in the feature space, as a function of node distances. While

we focused this short paper on applying our proposal to vanilla GCNs, it is possible to implement our method in more sophisticated frameworks including the recent GAT [18]. We believe that our method is general, and can apply to different applications of Graph Convolution.

References

[1] S. Abu-El-Haija, A. Kapoor, B. Perozzi, and J. Lee. N-gcn: Multi-scale graph convolution for semi-supervised node classification. In *KDD Workshop: Mining and Learning with Graphs*, 2018.

[2] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[3] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. In *Journal of machine learning research (JMLR)*, 2006.

[4] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, 2014.

[5] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[6] M. A. et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015.

[7] W. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. In *Neural Computation*, 1997.

[10] F. Karimi, M. Genois, C. Wagner, P. Singer, and M. Strohmaier. Visibility of minorities in social networks. In *arxiv/1702.00150*, 2017.

[11] T. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[13] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *International Conference on Machine Learning*, 2009.

[14] J. B. Lee, R. A. Rossi, X. Kong, S. Kim, E. Koh, and A. Rao. Higher-order graph convolutional networks. In *arxiv/1809.07697*, 2018.

[15] Q. Lu and L. Getoor. Link-based classification. In *International Conference on Machine Learning (ICML)*, 2003.

[16] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. In *Knowledge Discovery and Data Mining*, 2014.

[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[18] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[19] J. Weston, F. Ratle, H. Mobahi, and R. Collobert. Deeplearning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655, 2012.

[20] Z. Yang, W. Cohen, and R. Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *International Conference on Machine Learning (ICML)*, 2016.

[21] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *International Conference on Machine Learning (ICML)*, 2003.