Yale

Observations and Analyses of Multi-layer Graph Convolutional Networks

Ngan Vu

Advisor: Daniel Spielman

Bachelor's Thesis

Statistics and Data Science

Submitted May 02, 2019

YALE UNIVERSITY

Acknowledgements

I would first like to thank my advisor, Professor Daniel Spielman, for his guidance before and throughout this project. Professor Spielman offered not only invaluable intuition and suggestion for advancement of this project, but also support and encouragement in difficult times. I would also like to thank Professor Marynel Vazquez, for her suggestions for initial research directions on Graph Convolutional Networks.

I also want to acknowledge Krishnaswamy Lab, where I have had a chance to work extensively with diffusion-based methods that gave me the necessary background for this project.

Ngan Vu

Yale University

New Haven, May 2019

Abstract

Graph Convolutional Networks are one of the most popular frameworks in Geometric Deep Learning, an emerging field focusing on extending the success of Deep Learning techniques to non-Euclidean domains, such as graphs and manifolds. However, is Deep Learning really necessary for learning on graphs, and if so, for what kinds of problems? This thesis analyzes the Graph Convolutional Networks framework presented by Kipf and Welling (ICLR 2017) and observes that removing the nonlinear activation function between layers of the neural network does not negatively affect its accuracy. The absence of nonlinearity means that all layers of the Graph Convolutional Networks can be collapsed into one linear layer, which is mathematically equivalent to a multi-step diffusion of data on graphs. This simplification removes the redundant complexity of Graph Convolutional Networks and increases interpretability of the framework. This thesis also provides an explanation on why this simplification is possible, as well as a speculation of cases in which Graph Convolutional Networks can be more powerful.

Keywords – Graph Convolutional Networks, Diffusion on Graphs, Non-linearity

Contents

List of Figures

List of Tables

1 Introduction

In the last decade, Deep Learning methods, which are based on the use multiple layers in artificial Neural Networks, have proven to be very successful on a broad range of tasks coming from very different fields. Some of the powerful models that have achieved excellent performance on difficult learning tasks include a Convolutional Neural Network that was trained to classify the 1.3 million high-resolution images in the LSVRC-2010 ImageNet training set into the 1000 different classes with unprecedentedly low error rates (Krizhevsky et al., 2012), a Long-Short Term Memory Recurrent Neural Network that achieves significantly better results on the WMT'14 English to French translation task (Sutskever et al., 2014), and so on.

Research on Deep Learning techniques has mainly focused so far on data defined in Euclidean domains, such as text and audio (one-dimensional), images (two-dimensional), and videos (three-dimensional). Geometric Deep Learning is an emerging field in the Machine Learning community that aims to generalize Deep Learning methods to non-Euclidean domains. Among the most popular methods in Geometric Deep Learning are Graph Convolutional Networks (Kipf and Welling, 2016), which generalize standard Convolutional Neural Networks to graphs by defining the graph convolution operation.

While Graph Convolutional Networks garnered much recognition, they have also raised some doubts about their capability. For example, it has been pointed out that they cannot learn to distinguish certain simple graph structures (Xu et al., 2018).

A question that I asked myself when I first started reading about Graph Neural Networks is how it performs compared to older methods on graphs. Being familiar with diffusion-based methods on graphs, I recognized that Graph Neural Networks make use of the graph diffusion operator, a normalized adjacency matrix, in their definition of graph convolution. The diffusion operator and its exponents are often used in diffusion based methods. Graph Neural Networks differ from these methods in that they include a non-linear activation function after each usage of the diffusion operator. This observation made me wonder how much of an improvement the non-linear activation makes, which is the main research question of this project.

2 Background

2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a "regularized" variant of multi-layer perceptrons – fully connected neural networks in which each node in one layer contributes to all nodes in the next layer. In CNNs, two layers are connected through a convolution operation. With a $(x_i \times y_i)$ input matrix I, a $(x_k \times y_k)$ kernel matrix K, and a $(x_o \times y_o)$ output matrix O, the formula for convolution in two dimensions is:

$$O = I * K \Leftrightarrow O(i,j) = \sum_{u=0}^{x_k-1} \sum_{v=0}^{y_k-1} I(i-u, j-v)K(u,v) \quad \forall i \in [x_o], j \in [y_o] \tag{2.1}$$

Each node in the output layer only receives weighted contributions from a small subset of nodes in a small neighborhood of the input layer. The weighted contributions are defined by the kernel of the convolution operation, which is shared across all nodes in a layer. Weight sharing dramatically reduces the number of free parameters to learn (hence "regularized"), lowering the computing resources needed for training and allowing for more complex and powerful networks.

Convolution is a linear operation. Therefore, to enable neural networks to learn more complex, non-linear functions, the output of each convolution operation is almost always passed through a non-linear activation function, such as sigmoid ($f(x) = \frac{e^x}{e^x+1}$), hyperbolic tangent ($f(x) = tanh(x)$), or ReLU ($f(x) = max(0,x)$). With σ as a non-linear function, a complete convolutional layer is defined as:

$$O = \sigma(I * K) \tag{2.2}$$

Another method to enable neural network to learn complex functions is to change the number of "channels" throughout the hidden layers. The number of channel of the input layer is equal to the number of features of each node in this layer. With only one kernel, the number of channels would stay the same after convolution. However, one can use multiple kernels to increase the numbers of channels, effectively "lifting" the data to a

higher dimensional space, where a simple linear function could be equivalent to a complex function a lower dimensional space.

Together with pooling layers and fully connected layers, convolutional layers are the main ingredient of Convolutional Neural Networks. CNNs have proven to be quite powerful in image analysis tasks. For example, LeNet-5 (Lecun et al., 1998), a simple CNN with 7 layers, is capable of classifying numbers in 32x32 pixel grayscale images digits and was used to recognize hand-written digits on bank checks. AlexNet (Krizhevsky et al., 2012), a variant of LeNet-5 with deeper, stacked convolutional layers and more filters per layer, won the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge), achieving a top 5 test error rate of 15.4%, while next best entry achieved an error of 26.2%.

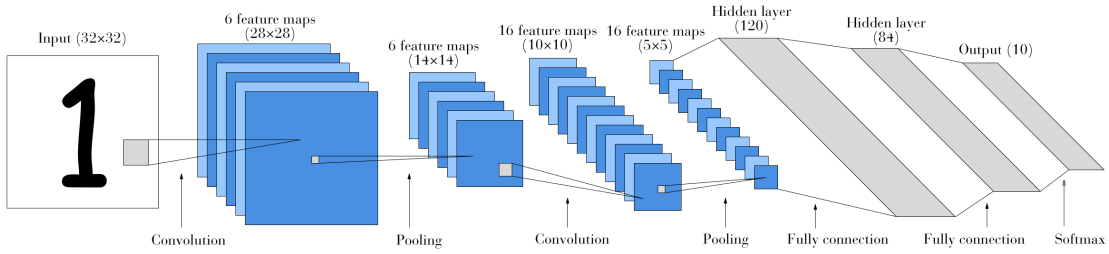

Figure 2.1: The architecture of LeNet-5, a CNN capable of classifying handwritten digits

The success of Convolutional Neural Networks on images, as well as other Euclidean domains, has driven the effort to generalize neural network models to non-Euclidean domain, including graphs. Given that many real world data sets (protein-protein interactions, social networks, knowledge graph, etc.) are graph-structured, more and more attention has been devoted to extending the application of CNN to this domain.

2.2 Graph Convolutional Networks

Graph Convolutional Networks attempt to generalize standard Convolutional Neural Networks to graphs by defining the graph convolution operator. The word "convolution" mostly comes from the "weight sharing" property of the convolution operation: filter parameters are shared among all nodes in the graphs.

In a standard convolution, each node essential pulls weighted contributions from nodes in its local neighborhood (including itself) to determine its value in the next layer. The

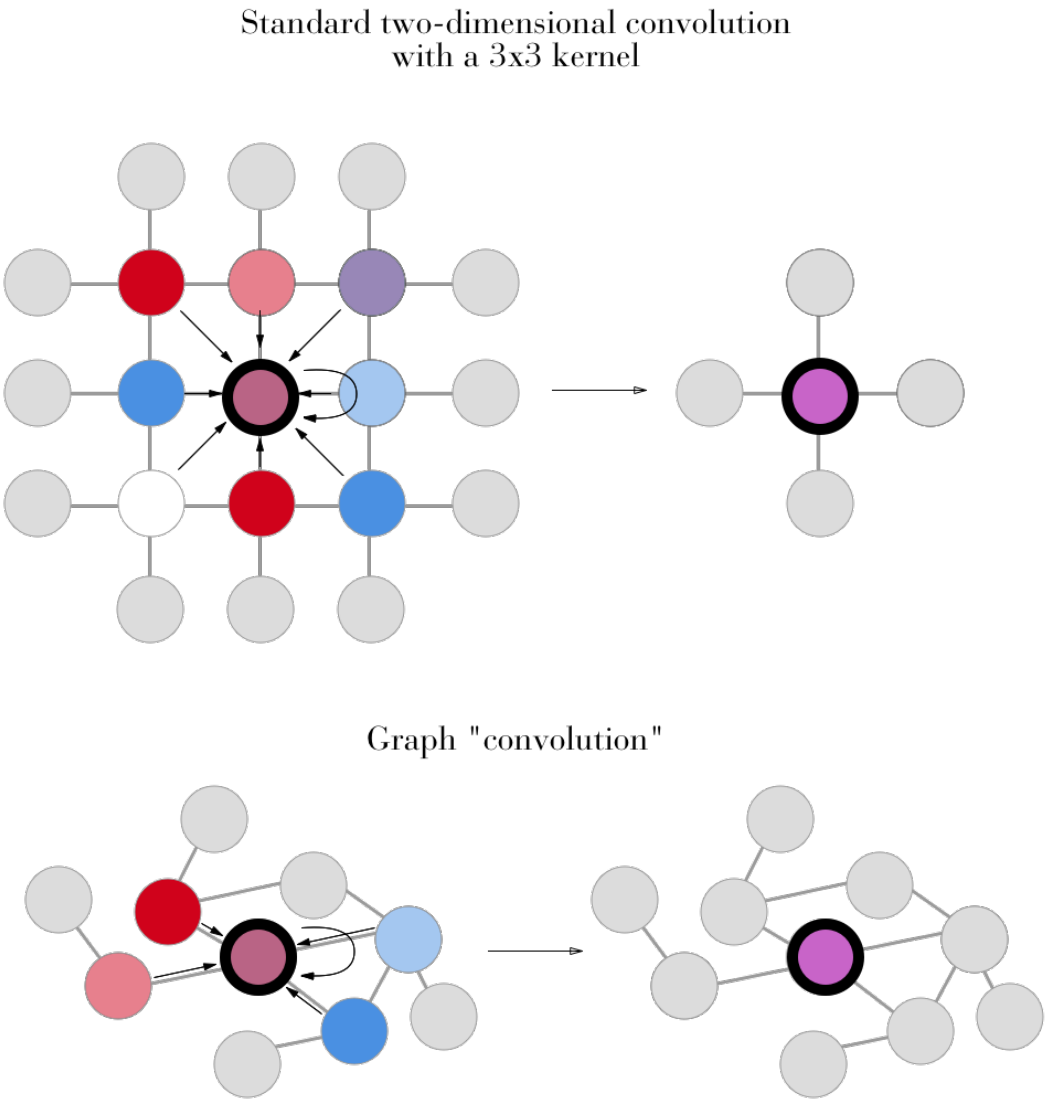

Figure 2.2: A comparison of standard convolution to graph convolution.

weights of the contributions are determined by the convolution kernel, which is shared among all nodes. Similarly, in a graph convolution, each node pulls weighted contributions from its neighbors – nodes that it shares an edge with – and itself. The weights of the contributions are determined by the weights of the edges in the graph, which are also shared information among all nodes in the graph.

To understand graph convolutions, we need three important matrices that represent a graph: the feature matrix X, the adjacency matrix A and the degree matrix D.

- The feature matrix X is an $n \times p$ matrix, where n is the number of nodes in the graph and p is the number of properties of each node. Each row i of X contains all

the information about node i in the graph.

- The adjacency matrix A is an $n \times n$ defined such that $A(i, j)$ has the value of the edge from i to j. If the graph is not directed, $A(i, j) = A(j, i)$.
- The degree matrix D is an $n \times n$ diagonal matrix defined such that $D(i, i)$ has the degree of node i. This also means $D(i, i)$ is the sum of row i in matrix A. All off-diagonal entries of D are zeros.

Our goal is to find a formula for graph convolution ($f(\cdot)$) that resembles standard convolution. The simplest way to make each node integrate weighted contributions from its neighbors is to left multiply the feature matrix with adjacency matrix. This would result in a graph in the same number of dimensions as the original graph. To transform this graph to a different number of dimensions (which is equivalent to changing the number of channels in standard convolution), we can right multiply the aggregated feature matrix with a weight matrix, which is learned during training.

$$f(X) = AXW \tag{2.3}$$

This formula does not allow a node to include contribution from itself. To fix this issue, we can self-edge to every node, effectively making each node a neighbor of itself. The formula should use $\hat{A} = A + I$ instead.

$$f(X) = \hat{A}XW \tag{2.4}$$

Another issue with this formula is that recursive application of $f(\cdot)$ will shrink X if the values in $\hat{A}$ are too small, or explode X if the values in $\hat{A}$ are too large. To avoid this problem, we can normalize $\hat{A}$ by using the normalized adjacency matrix $\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$, where $\hat{D} = D + I$ instead. The new formula for graph convolution is:

$$f(X) = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}XW \tag{2.5}$$

Lastly, to make graph convolution similar to standard convolution, we can use a non-linear

function.

$$f(X) = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}XW) \tag{2.6}$$

This is the formula for graph convolutional network. (Kipf and Welling, 2016)

2.3 Graph Diffusion Operator

$\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ is the symmetric, self-edged variant of the row-normalized adjacency matrix $D^{-1}A$. This matrix is a right stochastic matrix: the entries of each row of this matrix are non-negative and sum to one. This matrix is usually used to describe the transitions of a Markov chain, in which the probability of transitioning from node i to node j is proportional to the weight of the edge from i to j. Left multiplying the data matrix with this right stochastic matrix effectively "diffuses" each data point in the data matrix while preserving the total amount of data; hence, this matrix is also considered a graph diffusion operator.

Many machine learning algorithms make use of the graph diffusion operator. For example, Diffusion Maps (Coifman and Lafon, 2006) is a dimensionality reduction algorithm which computes a low-dimensional embeddings of a data set into using the eigenvectors and eigenvalues of the diffusion operator; MAGIC (Markov Affinity-based Graph Imputation of Cells) (van Dijk et al., 2018) is a method for imputing missing values restoring structure of large biological data sets using exponents of the graph diffusion operator. These methods are considered diffusion-based methods.

Each layer of the Graph Convolutional Networks is effectively one step of data diffusion. However, Graph Convolutional Networks differ from diffusion-based methods primarily in that each step of diffusion is followed by a non-linear activation function. This leads to the main question of this thesis: How effective is this non-linear transformation in GCNs?

3 Data

To test my hypothesis, I first used the same three data sets in Kipf and Welling (2016). These data sets, named Citeseer, Cora, and Pubmed, are citation data sets. Each node is a paper, and a directed edge exists between a paper that cites another paper.

The aforementioned data sets are decently small: it only takes less than a minute to train 200 iterations of Graph Convolutional Networks that achieve the same accuracy as mentioned in Kipf and Welling (2016) on a 2.7 GHz Intel Core i5 processor. Therefore, I also used a much bigger data set for a more comprehensive comparison: the Reddit dataset. Reddit is a social network data set, in which each node is a post on this online forum, and an edge exists between posts that have comments from the same user. This data set comes from Hamilton et al. (2017), a paper on another framework for inductive representation learning on large graphs, GraphSAGE. With the same processor, it takes 5 hours to finish 200 iterations of training on this data set.

Table 3.1: Information on data sets used

Data set	Type	Nodes	Features	Edges	Classes	Training %
Citeseer	Citation network	3,327	3,703	9,228	6	3.60%
Cora	Citation network	2,708	1,433	10,556	7	5.17%
Pubmed	Citation network	19,717	500	88,651	3	0.30%
Reddit	Social network	232,965	602	23,446,803	41	65.42%

4 Methodology

To test the effectiveness of non-linearity in Graph Convolutional Neural Networks, I first used the same model and data sets in Kipf and Welling (2016) but removed all non-linear activation functions. I also ran tests on bigger models with more hidden layer, as well as on the bigger Reddit data set.

After confirming that non-linearity does not play an important role in Kipf and Welling (2016), I restructured GCNs by collapsing all linear layers. Concretely, graph convolution is defined in Kipf and Welling (2016) as:

$$H^{(l+1)} = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{4.1}$$

where:

$$\begin{aligned}
\hat{D} &= D + I \text{ is the degree matrix with self-edges} \\
\hat{A} &= A + I \text{ is the adjacency matrix with self-edges} \\
H^{(l)} &\text{ is the } l^{th} \text{ layer, with the } 0^{th} \text{ layer being input} \\
W^{(l)} &\text{ is the } l^{th} \text{ weight matrix to be learned}
\end{aligned}$$

Without non-linearity, the recursive formula for the $(l+1)^{th}$ layer of GCNs can be rewritten in explicit form as:

$$H^{(l+1)} = \left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}\right)^{l} H^{(0)}W^{(0)}...W^{(l)} \tag{4.2}$$

The weight matrices $W^{(i)}$ can then be recombined into one single matrix $W = W^{(i)}...W^{(i)}$:

$$H^{(l+1)} = \left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}\right)^{l} H^{(0)}W \tag{4.3}$$

With this new model, we only need to learn one weight matrix. The exponentiated normalized adjacency matrix can be precomputed before training the single-layer neural network, reducing the number of matrix multiplications during each iteration of feedforward

and backpropagation through the neural network. With fewer parameters to learn and fewer operations to perform, hypothetically, it will take less time and computing resources to train the neural network.

The final output of GCNs for a node classification problem, either with or without non-linearity, is the result of a softmax function on the last layer of the network.

$$\hat{Y} = softmax(H^{(L)}) \tag{4.4}$$

where L is the number of layers.

I also compared GCN to Harmonic Classifier Zhu et al. (2003), another method for node classification using only adjacency matrix. This comparison is revealing in why non-linearity is not necessary for the data sets in Kipf and Welling (2016).

5 Analysis

5.0.1 Retraining Graph Convolutional Networks

The code in Kipf and Welling (2016) is open-sourced on Github (Kipf, 2016). There are multiple implementations using different machine learning frameworks by the same author; I chose the one that used Tensorflow based on the number of stars and forks of this repository.

I was able to successfully recreated the results from Kipf and Welling (2016). The Graph Convolutional Networks used for all data sets have one hidden layer and are trained 200 iterations with possible early stopping when the train loss plateaus out.

Table 5.1: Results of training original two-layer GCNs

Data set	Effective epochs	Test accuracy %
Citeseer	123	70.80%
Cora	133	81.30%
Pubmed	103	79.10%

5.0.2 Removing non-linearity

To test the effectiveness of the non-linear activation function in GCNs (which is ReLU by default), I retrained the network with non-linearity removed. The test accuracy on all data set stays roughly the same, suggesting that non-linearity might not be necessary.

Table 5.2: Results of training two-layer GCNs without non-linearity

Data set	Effective epochs	Test accuracy %
Citeseer	124	70.90%
Cora	139	81.90%
Pubmed	115	79.30%

5.0.3 Testing with a bigger network

Given that the GCNs above are quite small and non-linearity only appears once, I also tested GCNs with more layers. There are three observations to be made:

1. In general, the test accuracy gets worse with more number of hidden layers. This finding suggests that GCNs are prone to overfitting even with few layers.

2. The train loss plateaus faster with higher number of hidden layers, leading to lower number of effective epochs. However, training time for each epoch (which is not reported here since it depends on many factors of the processors used for training) is always longer due to higher numbers of parameters to learn.

3. There is no indication that having a non-linear activation function improves test accuracy. In fact, when there are more hidden layers, models with no non-linearity perform better.

Table 5.3: Results of training GCNs with more hidden layers on Citeseer dataset

Hidden layers	With non-linearity		Without non-linearity	
	Effective epochs	Test acc. %	Effective epochs	Test acc. %
2	28	66.70%	24	66.60%
3	15	64.80%	15	65.50%
4	12	60.20%	12	62.00%

Table 5.4: Results of training GCNs with more hidden layers on Cora dataset

Hidden layers	With non-linearity		Without non-linearity	
	Effective epochs	Test acc. %	Effective epochs	Test acc. %
2	30	79.80%	35	79.70%
3	19	79.00%	26	78.30%
4	15	74.00%	20	76.90%

Table 5.5: Results of training GCNs with more hidden layers on Pubmed dataset

Hidden layers	With non-linearity		Without non-linearity	
	Effective epochs	Test acc. %	Effective epochs	Test acc. %
2	25	75.90%	19	73.20%
3	17	73.70%	12	70.00%
4	15	70.90%	12	76.20%

5.0.4 Testing with a bigger data set

For a more comprehensive comparison between models with and without non-linear activation, I included a much bigger data set that was introduced in Hamilton et al. (2017)

but was not used in Kipf and Welling (2016). The test accuracy after 200 iterations of training is again comparable between two models.

Table 5.6: Results of training two-layer GCNs on Reddit dataset

	Effective epochs	Test acc. %
With non-linearity	200	58.72%
Without non-linearity	200	58.90%

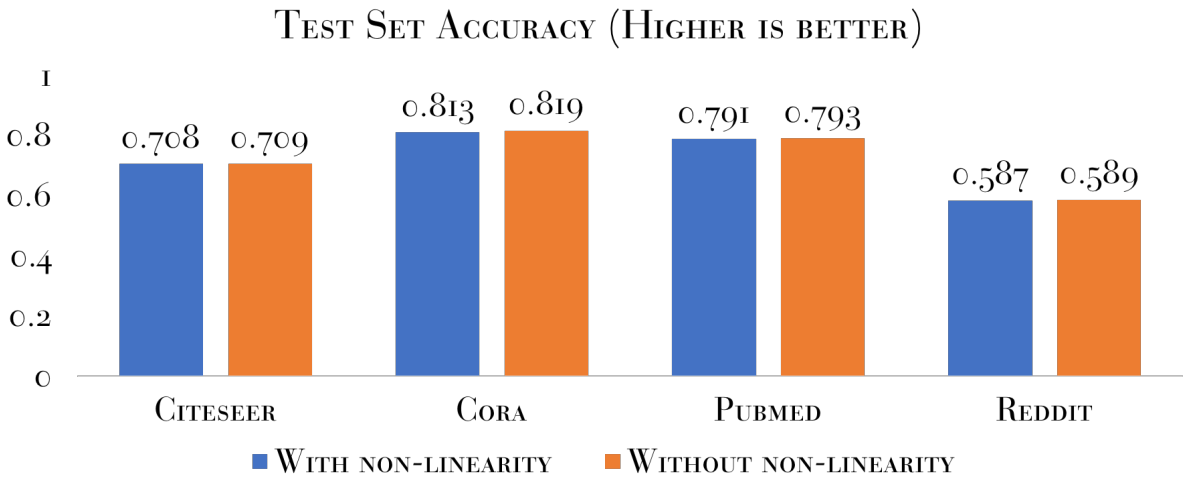

Figure 5.1: Test accuracy of two-layer GCNs on all data sets

5.0.5 Constructing a new model with collapsed hidden layers

Given the non-linear activation does not appear to contribute to the accuracy of GCNs, I constructed a new model in which all layers of GCNs are collapsed into one layer as described in Methodology. To mimic two layers of GCNs, I applied two-step diffusion on the input data before passing them to a Fully Connected layer. All other hyperparameters of this model is identical to those used in the original GCNs.

Table 5.7: Results of training linear GCN model

Data set	Effective epochs	Test accuracy %
Citeseer	129	65.70%
Cora	118	80.30%
Pubmed	137	76.70%

Theoretically, the results of this model should be identical to that of a two-layer GCNs without non-linearity. Empirically, however, the accuracy of this model is slightly lower

than that of linear GCNs on all data sets. I suspect that the difference comes from how Adaptive Moment Estimation (Adam) works: each step of gradient descent in a multi-layer network is different from that in a single-layer network. I did not attempt to fine tune hyper-parameters in this model since that would result in unfair comparisons between models of different hyperparameters.

A strong advantage of this model is the small number of free parameters. For all data sets, the number of free parameters to learn is much lower. This property is desirable for simplicity and interpretability of the model.

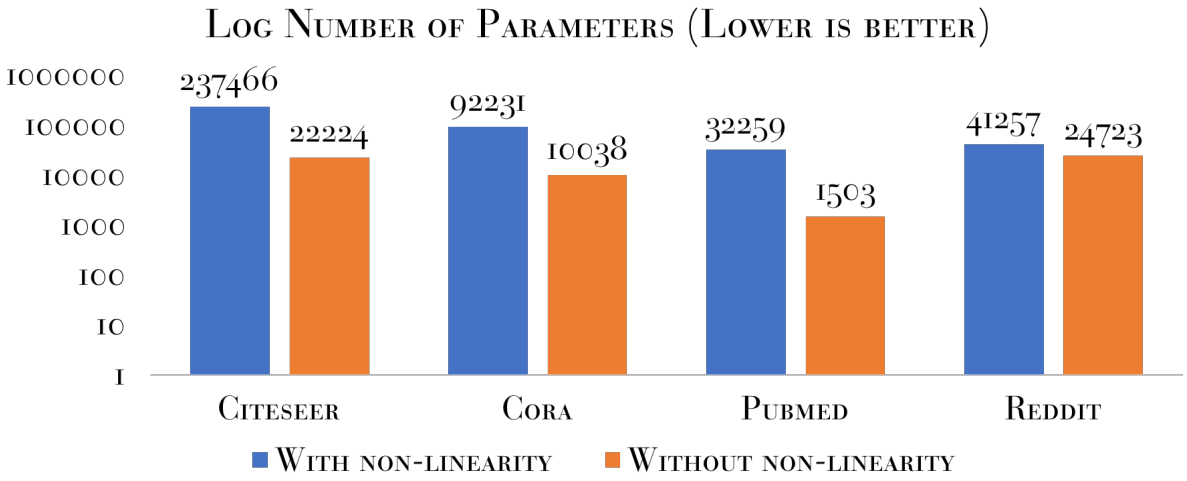

Figure 5.2: Number of free parameters of GCN vs. a model with collapsed layers

5.0.6 Comparing with Harmonic Classifier

Semi-supervised learning using Gaussian fields and harmonic functions (Harmonic Classifier) (Zhu et al., 2003) is another method for node classification. This method only makes use of the adjacency matrix and not the feature matrix at all. One issue with this method is that it can only classify nodes belonging to components that have at least one train label. To compare GCNs to Harmonic Classifier, I only consider such nodes. While Harmonic Classifier gives a poor result on Pubmed data set, it performs surprisingly well compared to GCNs on the Citeseer and Cora data sets, even without access to the feature matrices. This observation suggests that the adjacency matrices themselves already tell us a lot about the structure of these two graphs and perhaps encode the majority of the information needed from feature matrices for node classification.

Table 5.8: Results of Harmonic Classifier on nodes in components with train labels

Data set	Test accuracy %
Citeseer	71.08%
Cora	76.80%
Pubmed	24.88%

6 Discussion

In Deep Learning models, a non-linear activation is necessary between each layers for the neural networks to approximate functions that are more complex than linear transformations. The comparable performance of Graph Convolutional Networks in the absence of non-linearity suggests that a complicated function is not necessary for learning on graph data sets presented in this thesis.

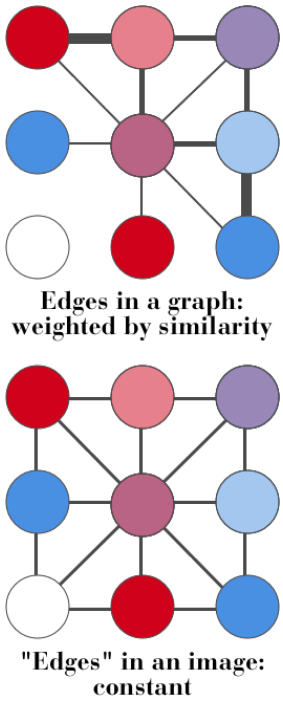

Figure 6.1: Comparing edges in a graph and "edges" in an image

Citeseer, Cora, and Pubmed are citation data sets, while Reddit is a social network data set. These graphs are considered "smooth": two neighboring nodes tend to be similar and the degree of similarity is proportional to the weight of the edge between them. In this sense, the edges reveal a lot about the structure of the data. In fact, we can see how much information the edges contain by looking the results of Harmonic Classifier, the node classification method that uses only the edges and no features. The test accuracy of this method is almost as high as that of GCNs for Cora data set and higher that that of GCNs for Citeseer data set.

In data sets used for standard Convolutional Neural Networks, the "edges" are not as revealing: for example, a pixel is always connected to its eight neighbors regardless of how similar they are. Therefore, we need a more complicated model to extract more features about the data before learning.

In brief, non-linearity is necessary in approximating a complicated non-linear function of input data. It appears, however, that at such a function is not needed for high accuracy in node classifcation tasks, at least on smooth data sets like those used in Kipf and Welling (2016). To fully utilize of the power of GCNs, we should use them on non-smooth data sets. An example of a non-smooth data set is a social network graph where the classes are genders: a female does not necessarily have the majority of her friends other females. In such case, the relationships between nodes are a lot more complicated and

the adjacency matrix itself cannot contain all necessary information, and GCNs might be more useful. GCNs would also be more powerful when the graph is more complex; for example, when the graph is multi-modal. An example of such a graph is introduced in Hamilton et al. (2017), where each node is either a protein or a drug, and each edge is either protein-protein interaction, protein-drug interaction, or drug-drug interaction. In this case, diffusion would not work as the nodes do not necessarily have the same kinds of features.

7 Conclusion

In this thesis, I found that the non-linear activation function is not necessary for Graph Convolutional Neural Networks in Kipf and Welling (2016) that are used to classify nodes in Citeseer, Cora, Pubmed, and Reddit data sets. I constructed a variant of GCNs that diffuses the feature matrix before passing through a single-layer neural network that is mathematically equivalent to GCNs with no non-linearity. I postulated that the removal of non-linearity is possible because the aforementioned data sets are smooth and do not require complicated functions for node classification. I suggested some cases in which GCNs could be more powerful.

References

Coifman, R. R. and Lafon, S. (2006). Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5 – 30. Special Issue: Diffusion Maps and Wavelets.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Inductive Representation Learning on Large Graphs. *arXiv e-prints*, page arXiv:1706.02216.

Kipf, T. N. (2016). Graph convolutional networks. https://github.com/tkipf/gcn/.

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

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

van Dijk, D., Sharma, R., Nainys, J., Yim, K., Kathail, P., Carr, A. J., Burdziak, C., Moon, K. R., Chaffer, C. L., Pattabiraman, D., Bierie, B., Mazutis, L., Wolf, G., Krishnaswamy, S., and Pe'er, D. (2018). Recovering gene interactions from single-cell data using data diffusion. *Cell*, 174(3):716 – 729.e27.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How Powerful are Graph Neural Networks? *arXiv e-prints*, page arXiv:1810.00826.

Zhu, X., Ghahramani, Z., and Lafferty, J. (2003). Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 912–919. AAAI Press.