Simplifying Graph Convolutional Networks – Report

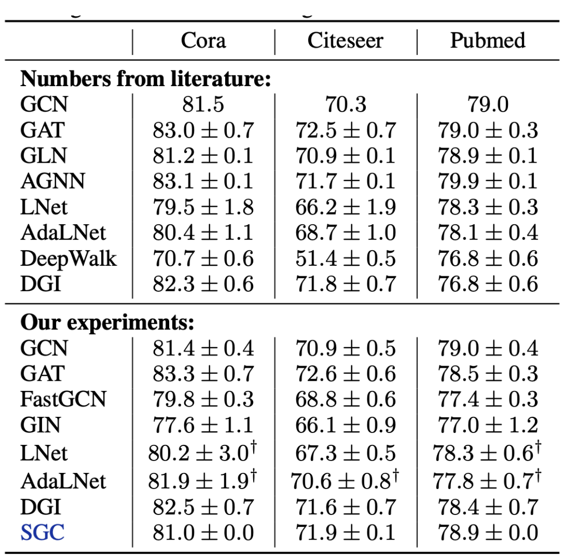
Elias Castro, Manolis Huerta-Stylianou, Tuni Le, Josue Ortiz-Ordonez
https://github.com/manolishs/cs4782-SGC/tree/main

Introduction

Graph Convolutional Networks (GCNs) have become the most widely-used models for learning graph representations. However, drawing inspiration from deep learning methods has caused them to inherit unnecessarily complex computational processes. In the paper Simplifying Graph Convolutional Networks, Felix Wu et al. (2019) introduced reduced computation in removing nonlinearities and collapsing weight matrices between consecutive layers without negatively affecting accuracy. Using three different citation networks as datasets, we reproduce the results in Wu's publication and demonstrate that its accuracy is not compromised by a substantial increase in efficiency, compared to the traditional GCN (Kpif et al. 2017).

Chosen Result

In Table 4 of the paper Simplifying Graph Convolutional Networks, we are presented with percentage test accuracy averaged on ten runs across numerous versions of graph networks on citation networks Cora (McCallum et al. 2000), Citeseer (Lee et al. 2000), and Pubmed (Sen et al. 2008). The authors of this paper provide both the original accuracies of these graph networks from their publications, and their own accuracies from reproducing the performance of these models on the citation networks. Additionally, we are provided with the test accuracy of the Simple Graph Convolution (SGC) on these datasets – which allowed the authors to display a similar accuracy to those of the SGC's counterparts, proving that this model does not compromise accuracy in favor of its improved compute efficiency. We aimed to reproduce the test accuracy of the original GCN – the first state-of-the-art model of its kind – and the SGC on the Cora, Citeseer, and Pubmed datasets as displayed in the paper. The very similar outcomes from these two models provide arguably the most important outcome of the paper, as it directly provides a strong argument for this novel method being as accurate as its predecessors.



	Cora	Citeseer	Pubmed
Numbers from literature:			
GCN	81.5	70.3	79.0
GAT	83.0 ± 0.7	72.5 ± 0.7	79.0 ± 0.3
GLN	81.2 ± 0.1	70.9 ± 0.1	78.9 ± 0.1
AGNN	83.1 ± 0.1	71.7 ± 0.1	79.9 ± 0.1
LNet	79.5 ± 1.8	66.2 ± 1.9	78.3 ± 0.3
AdaLNet	80.4 ± 1.1	68.7 ± 1.0	78.1 ± 0.4
DeepWalk	70.7 ± 0.6	51.4 ± 0.5	76.8 ± 0.6
DGI	82.3 ± 0.6	71.8 ± 0.7	76.8 ± 0.6
Our experiments:			
GCN	81.4 ± 0.4	70.9 ± 0.5	79.0 ± 0.4
GAT	83.3 ± 0.7	72.6 ± 0.6	78.5 ± 0.3
FastGCN	79.8 ± 0.3	68.8 ± 0.6	77.4 ± 0.3
GIN	77.6 ± 1.1	66.1 ± 0.9	77.0 ± 1.2
LNet	$80.2 \pm 3.0^{\dagger}$	67.3 ± 0.5	$78.3 \pm 0.6^{\dagger}$
AdaLNet	$81.9 \pm 1.9^{\dagger}$	$70.6 \pm 0.8^{\dagger}$	$77.8 \pm 0.7^{\dagger}$
DGI	82.5 ± 0.7	71.6 ± 0.7	78.4 ± 0.7
SGC	81.0 ± 0.0	71.9 ± 0.1	78.9 ± 0.0

Methodology

We begin by looking at the Graph Convolutional Network Paper, the paper which inspired our beloved SGC paper. We look at this paper because we aim to reimplement the GCN structure as we want to gain insight into the architecture and also reuse our own implementation of the GCN for our testing. In addition, by making our own GCN architecture, we will then use the SGC paper to simplify our own implementation of the GCN into a SGC as described in the paper.

To begin our reimplementation of the GCN we recreate a graph convolutional layer from the GCN paper. Our inputs are the input features and output features we want for the layer. We also have an optional parameter for the bias that we automatically set true. In this layer, we will create a linear layer with the input parameters above. We also have a function where we reset the parameters using the Xavier norm that the GCN paper uses. As such, when we initialize we do our reset. Finally for our forward function, our input goes through the linear layer and then is matrix multiplied with the adjacent norm. Now that we have our graph convolutional layer established, we will make the full GCN model architecture by using two convolutional layers. For our forward function, we go through the first layer, which transforms our input features to our hidden dimensions, go through a RELU activation, then a dropout layer and finally go through the final layer which transforms our hidden dimensions to the number of classes.

Upon analysis of the SGC paper, we see that the authors collapsed the entire architecture into a single linear layer that takes in our features and outputs it to the number of classes. We do the same and have our forward function simply be passing our input through the only linear layer in our SGC architecture.

We now move onto training, where we use the Cora, Citeseer, and Pubmed datasets that were preprocessed through the planetoid module. Our evaluation metric will be accuracy as we want to see how many labels we have correct. To preprocess we also row normalize the node feature matrix and construct the symmetrically normalized adjacency matrix for training both the GCN and SGC models. For SGC preprocessing, we propagate the features through the graph for K-steps as described in the paper. Note that we do this after constructing the normalized adjacency matrix. This is the reason why SGC just has 1 linear layer for its architecture. For the GCN and SGC training, we do a standard ML training loop and then evaluate our architecture by plotting the validation accuracy and the training loss. We acknowledge that unlike the SGC paper, we do not tune our weight decay to each dataset. Our results are in the table below.

We largely use hyperparameters as applied in the work of Wu et al. Our GCN applies 16 hidden units, a dropout rate of 50%, a learning rate of 0.01, and 200 training epochs. On the other hand, our SGC applies one linear layer with no dropout, a learning rate of 0.2, and 100 epochs. Across both models, we applied Adam as an optimizer and cross-entropy to evaluate loss.

Results & Analysis

As seen in the table in the following page, our reimplementation for the GCN is on par with the findings found on both papers (Kpit et al.'s and Wu et al.'s). As for the SGC, our average is a little below that of We et al.'s experiment. We suspect that the reason for this is because we did not tune the weight decay on each training set. A major challenge for reimplementation was understanding what GCNs are as we did not go over the topic in class. In addition, the theory required a review of linear algebra to understand as it was important for implementing the K-step feature propagation and understanding why they did it.

From a broader research perspective, this supports work as seen in Wu et al. toward placing emphasis on lighter, faster models over the depth or nonlinearity of a network. Traditionally, many computer scientists assume that the depth of a network allows for it to more effectively learn the features of a structure, but as our results support, this may not be necessarily true in cases such as this one, where graph structure exploitation provides more insight than the network's complexity.

Reflections

This project demonstrates that Simple Graph Convolution (SGC) achieves accuracy comparable to traditional Graph Convolutional Networks (GCNs) on the Cora citation dataset, while offering significant computational efficiency gains. We observed that the training time of the SGC and that of its predecessors demonstrates a significant increase, demonstrating an overall improvement in our ability to analyze and learn graph networks through these models. Our reimplementation supports the findings of Wu et al. (2019), indicating that removing nonlinearities and collapsing weight matrices across layers can be an effective strategy for certain graph learning tasks. Currently, our SGC succeeds only in limited settings under the assumption that data isn't noisy or incomplete. One way that we aim to address these limitations is through extending datasets outside of citation networks – some examples include social networks, protein interaction graphs, and recommendation systems. Another approach we can take to improving the

SGC lies in incorporating graph structure learning, which includes refining graph topology during training.

Table – Comparison of training accuracies across different papers

	Cora	Citeseer	Pubmed
GCN (Kipf et al.'s experiment)	81.3	70.3	79.0
GCN (Wu et al.'s experiment)	81.4 ± 0.4	70.9 ± 0.5	79.0 ± 0.4
GCN (our experiment)	80.7 ± 0.8	70.9 ± 0.7	79.0 ± 0.5
SGC (Wu et al.'s experiment)	81.0 ± 0.0	71.9 ± 0.1	78.9 ± 0.0
SGC (our experiment)	80.6 ± 0.1	68.3 ± 0.1	77.9 ± 0.1

References

Giles, C. Lee, et al. *CiteSeer: An Automatic Citation Indexing System*, Association for Computing Machinery, 11 May 1998, clgiles.ist.psu.edu/papers/DL-1998-citeseer.pdf.

McCallum, Andrew Kachites, et al. *Automating the Construction of Internet Portals with Machine Learning - Discover Computing*, Kluwer Academic Publishers, July 2000, link.springer.com/article/10.1023/A:1009953814988.

Sen, Prithviraj, et al. *Collective Classification in Network Data*, 6 Sept. 2008, eliassi.org/papers/ai-mag-tr08.pdf.

Thomas, Kipf N, et al. *Semi-Supervised Classification with Graph Convolutional Networks.* Arxiv, 22 Feb. 2017, arxiv.org/pdf/1609.02907.pdf.

Wu, Felix, et al. *Simplifying Graph Convolutional Networks.* Arxiv, 20 June 2019, arxiv.org/pdf/1902.07153.