Reproduction of Semi-Supervised Classification with Graph Convolutional Networks

Zhisheng Xu and Goutham Pakalapati
{zx41, gpakal2}@illinois.edu

Group ID: 28
Paper ID: 11
Presentation link: https://youtu.be/nHn2F-Seaas
Code link: https://github.com/xuzhisheng93/cs598-dl-healthcare-proj
Jupyter Notebook: src/project.ipynb within the code repository

1 Introduction

The Semi-Supervised Classification with Graph Convolutional Networks (Kipf and Welling, 2017) aims to solve the problem of semi-supervised classification on graph-structured data. GCNs are engineered to take advantage of the data's local structure within graphs by utilizing convolutional layers on the adjacency matrix of the graph. It develops a scalable and efficient method that can accurately classify nodes or predict labels for unseen data points in the graph which has limited amount of labeled data.

2 Scope of reproducibility

The proposed GCN model can achieve state-of-the-art performance on semi-supervised node classification tasks by effectively leveraging both labeled and unlabeled data through graph-based regularization. Specifically, the proposed GCN model outperforms previously proposed methods on three benchmark datasets in terms of

- classification accuracy, and
- maintaining the competitive runtime speed.

3 Methodology

To reproduce the results, we will utilize both the datasets and code from the original author. The code has undergone minor modifications to make it compatible with the latest TensorFlow and Scipy packages. Additionally, we plan to further tweak the code to test different activation functions later in the project. The code was obtained from the author's GitHub repository, located at https://github.com/tkipf/gcn. By following the instructions provided in the README file, we were able to successfully execute the models using the datasets provided. The computations were performed on an RTX 3080 Ti GPU.

3.1 Model descriptions

The GCN model is a two-layer neural network of graph convolutional layers. The first layer is a graph convolutional layer with 16 hidden units followed by a ReLU (Rectified Linear Unit) activation function. The second layer is another graph convolutional layer, which serves as the output layer and has as many output units as the number of classes in the classification problem. Graph convolutional layers are to graph structures what classic convolutional layers are in processing of images. In the graph convolutional layer, the features of each node are updated by using an aggregation of information from neighboring nodes to capture local relationships and structures present in the graph structure.

$$\mathcal{L} = -\sum_{l \in \mathcal{Y}_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf} \tag{1}$$

3.2 Data descriptions

We will be testing with 3 citation network datasets – Citeseer, Cora and Pubmed – nodes are documents and edges are citation links. These datasets are obtained from the author's GitHub repository (Kipf, 2016). And these dataset were first used by Collective classification in network data (Sen et al., 2008) The following table contains statistics of the datasets. The three datasets that we used are well-established benchmark datasets that are based on real-world applications. Specifically, for each of the citation network datasets, each of the scientific papers are represented by feature vectors in a sparse bag-of-words representation. In the implementation, we use undirected edges to represent citations between the scientific papers.

3.3 Hyperparameters

The model in the paper used a learning rate of 0.01, a weight decay (L2 regularization) of 5e-4, and

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

Table 1: Dataset statistics

a dropout rate of 0.5. The model was trained for 200 epochs using the Adam optimization algorithm. Early stopping was applied to prevent overfitting, with training stopping after 10 epochs with no improvement in validation loss. These hyperparameters were determined through a combination of grid search and manual tuning, and have been shown to perform well on the benchmark datasets used in the paper. Hence, we used the hyperparameters specified in the paper for our experiments.

3.4 Implementation

As mentioned previously, we plan to use the code provided by the authors to replicate the main experiments presented in the paper. Our code with the modifications for ablations will be available at https://github.com/xuzhisheng93/cs598-dl-healthcare-proj. The Jupyter Notebook of the reproduction and descriptions can be found at src/project.ipynb within the repository.

3.5 Computational requirements

The author used a 16-core Intel Xeon CPU E5-2640 v3 @ 2.60GHz with a NVIDIA GeForce GTX TITAN X. We will be running the model with AMD Ryzen 7 3700X 8-Core Processor and NVIDIA GeForce RTX 3080 Ti. In the Results section of the original paper, it states the running time on each of the datasets:

- Citeseer: 7 seconds
- Cora: 4 seconds
- Pubmed: 38 seconds

Since the power of our GPU is greater than theirs, we expect our running time to be similar to or faster than the original paper. And the total number of experiment combinations of ablation is 42. We ran one trial for each of the combination. Additionally, we also limit the maximum number of epochs to 200.

4 Results

4.1 Results of Reproduction

Overall, we managed to execute the original model with the provided datasets. And we are able to achieve similar accuracy to the original paper. Our runtime speed is faster than the original paper, which is due to the fact that we are using a more powerful GPU.

Table 2: Classification Accuracy and Runtime Comparison

Dataset	Accuracy		Runtime (s)	
	Paper	Ours	Paper	Ours
Citeseer	70.3%	70.6%	7	5.4
Cora	81.5%	81.7%	4	3.4
Pubmed	79.0%	79.4%	38	12.5

4.2 Analysis

4.2.1 Citeseer Dataset

Our reproduction results for the Citeseer dataset show a classification accuracy of 70.6%, which is slightly better than the original result of 70.3% reported in the GCN paper. This indicates that our model is consistent with the original paper's findings and that the GCN model outperforms previously proposed methods in terms of classification accuracy on the Citeseer dataset. Regarding the runtime, the original paper reported 7 seconds, while our results show an improvement in runtime at 5.4 seconds.

4.2.2 Cora Dataset

For the Cora dataset, our reproduction achieves a classification accuracy of 81.7%, which is very similar to the original accuracy of 81.5% reported in the GCN paper. This result confirms the effectiveness of the GCN model for the Cora dataset and supports the claim that the model outperforms other methods in terms of classification accuracy. In terms of runtime, the original paper reported 4 seconds, while our results show a slightly faster runtime of 3.4 seconds. Again, this shows another slight improvement on the original paper results.

4.2.3 Pubmed Dataset

Our model achieves an accuracy of 79.4% on the Pubmed dataset, which is slightly higher than the original result of 79.0%. Similar to the other two datasets, this suggests that our reproduction is consistent with the original findings, and the GCN

model outperforms other methods in terms of classification accuracy on the Pubmed dataset. For runtime, the original paper reported 38 seconds, while our results show a significant improvement with a runtime of 12.5 seconds. This is a great improvement upon the original time for the Pubmed dataset, demonstrating the efficiency of the GCN model.

In summary, our reproduction results confirm the claims made in the original GCN paper. The model outperforms previously proposed methods on all three benchmark datasets (Cora, Citeseer, and Pubmed) in terms of classification accuracy while improving on speed.

4.3 Results of Additional Experiments

Subsequently, we explored potential performance improvements by implementing several modifications to the original model. Specifically, we experimented with alternative activation functions in the first graph convolutional layer, in place of the original ReLU function. Additionally, we tested different optimization algorithms for minimizing the loss function. Finally, we examined the impact of altering the architecture of the GCN model by either removing or adding a layer, to assess the resulting differences in performance.

4.3.1 Exploration with Different Activation Functions

In terms of swapping activation functions, the majority of activation function (nn.tf.leaky_relu, nn.tf.tanh, nn.tf.elu) options resulted in very similar performance to the original results using tf.nn.relu. There were slight improvements or declines in performance depending on the dataset and activation function, as illustrated in the table above. However, the Sigmoid activation function yielded significantly worse results compared to the other four activation functions. This is possibly because the non-zero centered Sigmoid function can cause vanishing gradient issues, which might lead to slower convergence or getting stuck in local minima. Overall, modifying the activation function does not seem like a worthwhile step to add onto the original process.

Table 3 on page 3 shows the result of the experiments with different activation functions.

4.3.2 Exploration with Different Optimizers

The second experiment we performed on top of the original study was to try alternate optimizers.

Activation Function	**Cora**	**Citeseer**	**PubMed**
ReLU	0.817	0.706	0.794
Sigmoid	0.092	0.077	0.218
Tanh	0.816	0.704	0.793
Leaky ReLU	0.813	0.704	0.789
ELU	0.817	0.701	0.791

Table 3: Accuracies for different activation functions and datasets

Specifically, we tested the GradientDescentOptimizer, Adadelta, and RMSProp. For each optimizer, we obtained results using learning rates of 0.01 and 0.99, as shown in the table above. For the datasets, we achieved results close to the original findings using at least one combination of optimizers and learning rates. Furthermore, for the RMSProp optimizer on the PubMed dataset, I decided to try a learning rate of 0.1 and received an accuracy of 0.792, which slightly edges out the original results for this dataset using the AdamOptimizer. This shows that the optimal learning rates for each optimizer may lie between 0.01 and 0.99, so another step we can try in the future is to do a grid search to find the best optimizer and learning rate combination, which may also depend on dataset. We would want the most general one. Overall, modifying the optimizers appears to be a promising direction for further improvement.

Table 4 on page 3 shows the result of the experiments with different optimizers.

Optimizer (LR)	**Cora**	**Citeseer**	**PubMed**
Adam (0.01)	0.818	0.706	0.794
Adam (0.99)	0.718	0.679	0.666
Adadelta (0.01)	0.145	0.268	0.327
Adadelta (0.99)	0.388	0.433	0.608
RMSProp (0.01)	0.760	0.660	0.774
RMSProp (0.99)	0.729	0.697	0.765
Gradient Descent (0.01)	0.144	0.269	0.328
Gradient Descent (0.99)	0.444	0.445	0.547

Table 4: Accuracies for different optimizers, learning rates, and datasets

4.3.3 Exploration with Different Number of GCN Layers

The third experiment we conducted involved adding and removing a layer from the GCN model. Specifically, we tested the model with a single GCN layer, double GCN layers, and triple GCN layers. Although the performance of these models is not comparable to the original two-layer model, the model with three GCN layers showed similar performance to the original model that had two GCN layers. Furthermore, the model with one GCN layer showed worse performance than the original model. This could be because the model with one GCN layer is not deep enough to learn the features of the dataset. Overall, simply modifying the number of GCN layers in the model without changing other parameters did not provide promising results.

Table 5 on page 4 shows the result of the experiments with different number of GCN layers.

Number of Layers	**Cora**	**Citeseer**	**PubMed**
2 (original setup)	0.817	0.706	0.794
1 (2nd layer removed)	0.742	0.652	0.724
3 (added ReLU layer)	0.796	0.670	0.747

Table 5: Accuracies for different number of layers and datasets

5 Discussion

Our experiments are successful in demonstrating that the original paper by Kipf and Welling is indeed reproducible. By following the methodology in the paper, we were able to closely replicate their results, confirming the validity of their work. The consistency between our results and those of the original authors not only supports the conclusions of the paper but also suggests potential improvements due to advances in hardware since the original study. These improvements may lead to more efficient training and better performance of the GCN model in certain cases. Our findings provide a strong basis for further studies in graph convolutional networks, as researchers can be confident in the reproducibility of the original paper's methods and results. Additional modifications and optimizations should be explored, contributing to more effective applications of GCNs in the real world.

Reproducing the original study was made easier by the availability of the datasets on the authors' GitHub repository, which assisted greatly in the process of acquiring the necessary datasets. Additionally, the provided code was a solid starting point (not counting the issues we faced with the TensorFlow library setup due to the authors using more legacy functionality), enabling us to efficiently obtain initial results that matched those of the original authors. Furthermore, the practicality of the GCN model, along with its resemblance to convolutional neural networks, contributed to the ease of application of the method.

Some challenges we encountered during the reproduction process included understanding the theoretical and mathematical background of graph convolutional networks as well as previous papers that the new method relied on. Furthermore, it was not simple to apply the concepts of convolution to a more abstract structure, such as graphs. These aspects required a deeper level of understanding to effectively implement and adapt the GCN model. Additionally, we experienced technical difficulties when attempting to run the code initially, which necessitated the use of an older, compatible version of TensorFlow.

For the original authors and others working on this area, we would recommend that they spend time on expanding the types of datasets to more than mostly citation networks and further optimize the hyperparameters that were covered in our reproduction such as learning rate. This would allow them to test the generalizability of their methods on a wider range of graph examples. Hopefully, this helps make the next iteration even more successful on diverse types of data.

References

Thomas Kipf. 2016. Gcn: Implementation of graph convolutional networks in tensorflow. https://github.com/tkipf/gcn.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. 2008. Collective classification in network data. *AI Magazine*, 29(3):93.