Deep Learning Coursework: The COMP6248 Reproducibility Challenge

Aditi Dutta, Razan Alshareef, Yuxin Guo
University of Southampton

1 INTRODUCTION

This paper is a reproducibility report written in accordance with the ICLR reproducibility challenge as a requirement of module COMP6248. The report is accompanied by a GitHub repository that contains the code used to obtain the results in this paper. Our choice fell on a paper titled 'DropEdge: Towards Deep Graph Convolutional Networks on Node Classification', which proposes a novel method called DropEdge that overcomes two main obstacles of developing deep Graph Convolutional Networks (GCN): over-fitting and over-smoothing. DropEdge adapts a random approach to drop graph edges such that it achieves both data augmentation and reduction of message passing. The method is highly flexible and can be utilized with many backbones (e.g. GCN, ResGCN, and JKNet) to enhance their performance. The rest of this paper is organized as follows: section 2 describes the methodology and target questions. Next, different experiments and observations are illustrated in section 3. After that, section 4 provides an extensive discussion and a conclusion.

2 METHODOLOGY

Considering that DropEdge is a novel approach that was prodigal with paramount claims, we decided to adapt a hybrid methodology of a reproducibility analysis and a comparative analysis, achieving dual verification of results and claims by answering the question: Is this paper reproducible?

Reproducibility Analysis The aim of reproducibility analysis is validating an original experiment by replicating it by conducting an independent study and investigating the similarities of the results. The analysis was carried out through applying the authors' methodology along with investigating the choice of some hyperparameters (e.g. learning rate adjustment) and model-related claims (e.g. over-smoothing).

Comparative Analysis The aim of comparative analysis is to determine and quantify differences between two or more similar concepts through an identical set of variables. The comparison was conducted by exploring the performance of popular methods mentioned, such as, Dropout and DropNode on the same dataset to prove/disprove the authors' claims.

3 EXPERIMENTS

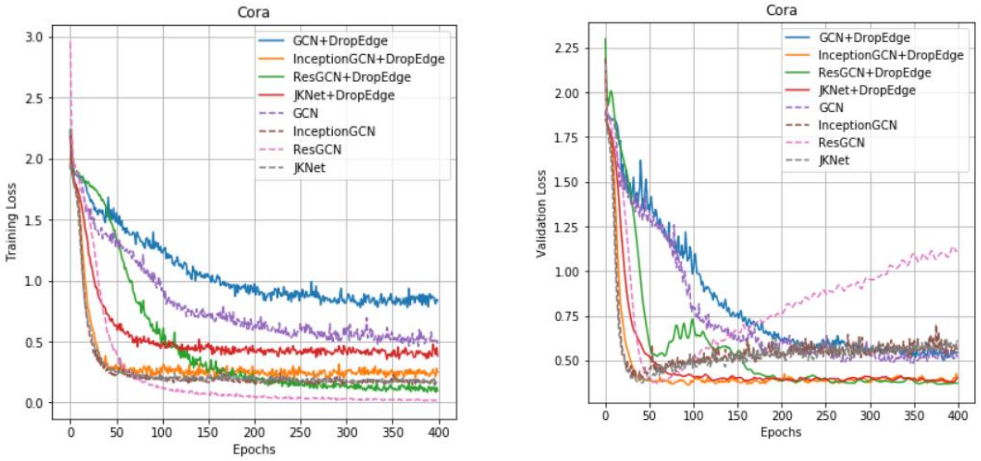
In this section, we present the experiments done precisely as described in the paper (Base Experiments) to reproduce the results and add an extra set of experiments (Additional Experiments) in order to gain a more in-depth comparison between the approaches used by the author and our experiments.

3.1 BASE EXPERIMENTS

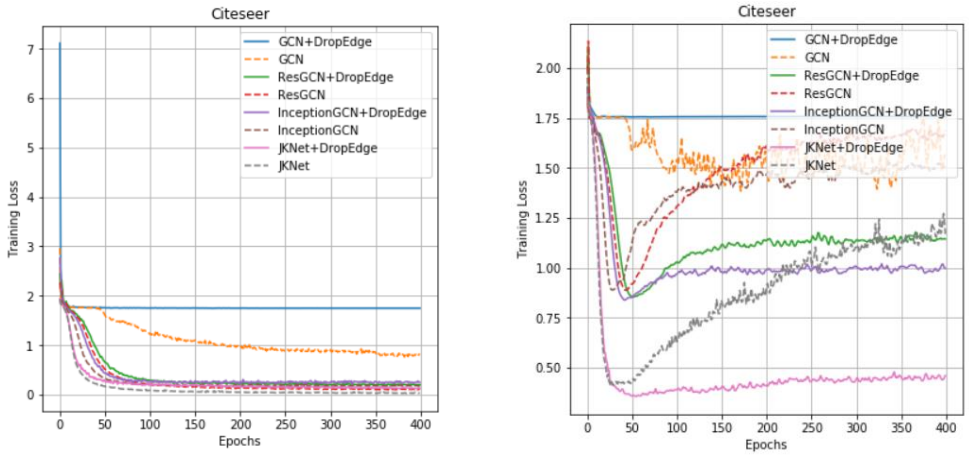
We have successfully managed to replicate the results from the DropEdge paper. For the purpose of inspection, we used all the backbones (replacement of the CNN layer with graph convolution layers) as used by the author, except for the GraphSAGE, because allthough it was there in the paper, the author's code only had the following backbones- GCN (Multilayer GCN), inception network (IncepGCN)[3], residual network

(ResGCN)[1] and dense network (JKNet)[2]. The layers in ResGCN, JKNet and InceptGCN are of at least 3 layers. The experiments were performed using the code provided by the paper's authors, however we had to make certain changes to that code to make it run properly and give the proper results. Some of the hyperparameters were already pre-set into the code with respect to each backbone, like the aggregation method, learning rate adjust, optimal loss function (nll_loss), task type (supervised learning method over semi-supervised), etc. Hyperparameters such as early-stopping and fast mode significantly factor in the amount of data a model utilizes, it is of the utmost importance that they be balanced such that the degree of their combined influence does not hinder the optimum result. One example is setting the value of earlystopping to 0 and enabling fast mode for larger graphs like ours, so that the model utilizes the full training set.

While experimenting, we found that the code they used for total layers to be ambiguous as it was not properly mentioned and likewise justified in the code. In fact, we tried the equation of total layers as stated by them in the code and worked it on IncepGCN with total number of layers as 2, and it ran perfectly fine (which should not have been the case). We usually conducted our observations by keeping the total number of layers to 6 (according to their equation). They have rightly mentioned the need for some of the hyperparameter tuning as we felt that it effected the results quite a bit. Also, the authors did the experiments using both Transductive (Cora, Citeseer, Pubmed) and Inductive datasets (Reddit). Although the datasets provided along with the code did not have Reddit in it, the code provided by the author had specific instructions written for it.



(a) Training and Validation losses of Cora

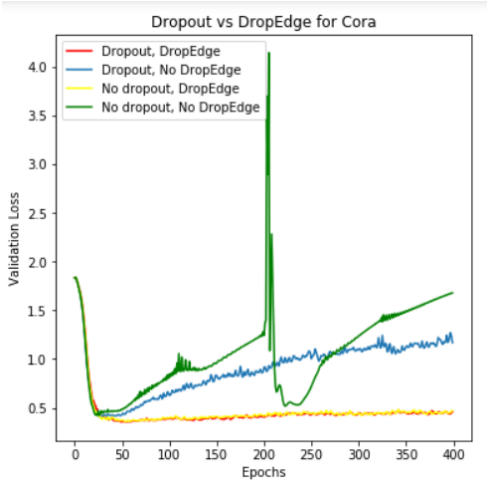


(b) Training and Validation losses of Citeseer

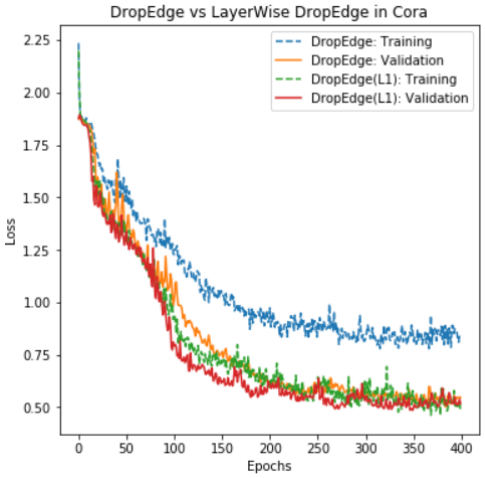
The authors fixed the number of epochs to 400, even though they worked with 200 and kept the default epochs to 800 in the code. While trying all that, it was found that the

experiments worked best with keeping the number of epochs at 400, showing the best accuracy of the three and some interesting observations beyond 200th epoch. Some work with the hidden dimensions stated that keeping it at 128 showed the best result. Tuning the hyperparameters for each datasets with each backbone showed us that the graph results differed a bit and optimum parameters for each backbone, with or without DropEdge tended to have different results than the ones as shown in the paper. Figure (a) and (b) shows the losses with two of the three datasets. We tried using different optimizers as well and it looked like Adam is the best suited one. Of all the four optimizers we used- Adam, SGD, SGD with momentum and Adagrad, Adagrad showed moderate performance with around 50% accuracy, while both the SGDs performed pretty bad with all our models and datasets. All our experiments were conducted in CPU, unlike that of the author's and initial performance with both showed that CPU was preferred more here as large graphs like ours do not fit in GPU memory and showed Out-of-memory(OOM) in most of the larger layers. In spite of that, Pubmed took the most time to compute.

Additionally, we tried experimenting with the author's claim on using dropout with and without dropEdge and we seem to get the same conclusion as theirs from the graph that with DropEdge, we get better results either way, and having dropout does not really effect much to the output (Fig (c)). We somehow see some changes with dropout without DropEdge after 200th epoch when at some point it does reach a low value, but yet continues to be above the value with DropEdge. Same for their other claim, which stated that having LayerWise DropEdge helped reduce the loss for both the validation and training set. We know that having a much lower training loss is helpful during experiment and using LayerWise DropEdge provides just that (Fig(d)).



(c) Dropout vs DropEdge



(d)DropEdge vs LayerWise DropEdge

Also, we performed an experiment with the author's claim showing that the distances of different intermediate layers had different edge dropping rates. And it did show a difference in dropping rates, some layers do need different dropping rates for smoothening. With DropNodes, it shows considerable difference with DropEdge (slightly better of the two), but removing edges seem more convenient as the DropEdge does not change the expectation of neighbor aggregation, which plays a crucial role in characterizing input graphs. Hence, the statistics of graph properties are still preserved.

3.2 ADDITIONAL EXPERIMENTS

Graph sparsification is the process of approximating a given graph by a sparser version of it-less edges- while keeping the loss of information at a minimum. The group of sparsification methods was described in the paper to "resort to a tedious optimization method to determine which edges to be deleted", however, while the random approach of DropEdge is one of its prominent attributes, no proof was made to the effect of it outperforming an optimized approach, such as graph sparsification.

In this section we highlight a few points that indicate, in principal, that adapting a graph sparsification method jointly with DropEdge has the potential of enhancing its performance.

Linear-time Sparsification Algorithms The latest publications in the field of graph sparsification show improved techniques with a running time of O(m) for unweighted graphs and Om+O(n/ϵ2) for unweighted graphs [4], hence, overcoming one of its least favored characteristics.

Epochs vs Sparsification Given the experiment results on the number of epochs above, it can be concluded that the random approach of DropEdge causes a significant loss of information that must be redeemed by an increased number of epochs, of course, this issue can potentially be mitigated by adapting an optimized graph sparsification approach.

4 DISCUSSION AND CONCLUSION

The reproducibility of the proposed model DropEdge was successful and the accompanying code ran successfully as well. However, a few issues were noticed. Most prominent of which is the significant discrepancies between results of loss and accuracy shown in different tables of the paper, compared to the results obtained by running the author's original code. With that being said, the model had a lot of advantageous aspects and equal amount of potential areas of improvements. Most importantly, the novel model showed considerable enhancement to some state-of-the-art (SOTA) models. Further work suggested would be an optimization of the random approach of dropping edges in the model, as that would ensure higher information value and even greater accuracy.

REFERENCES

[1] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In International Conference on Computer Vision, 2019.

[2] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. arXiv preprint arXiv:1806.03536, 2018b.

[3] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818–2826, 2016.

[4] Wai-Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi SIAM Journal on Computing 2019 48:4, 1196-1223.