Experimenting with Graph Convolutional Neural Networks for Node Classification

Sohail Nizam

September 20, 2020

1 Graph Mining Task and Data Source

In this paper we discuss the performance of various graph neural network architectures on a node classification task. In particular, we consider a graph with scientific documents as nodes and citation links as edges (Kipf and Welling, 2016). The documents are publications found on PubMed. PubMed is a free search engine that can be used to find papers on the life sciences and biomedical information. The PubMed dataset is available as part of the Planetoid class in Pytorch Geometric. The PubMed dataset consists of 19,717 documents and 44,338 edges. There are 3 node classes.

The documents are each represented as a bag of words. In a bag of words document representation, the individual vector elements are counts of the number of times a given word appears in that document. The vector has an entry for each unique word, and there is one vector for each document.

2 Basic GNN Structure

First, a very basic graph neural network structure was used for node classification. This GNN consisted of only a single convolutional layer followed by a softmax to allow for class prediction. The convolutional layer was the GCNConv layer detailed by Kipf and Welling (2016). Note that no ReLU layer was used following the convolutional layer. Negative-log-likelihood loss was used.

The reasoning behind this model is that it should act as a good baseline to compare performance against. More sophisticated architectures should provide improvement (like the addition of ReLU layers with dropout as well as more convolutional layers, some fully connected layers, and different types of convolutions). However, it still has the basic architecture needed to perform node classification.

3 GNN Structure Improvements

Six different GNN architectures were employed to try and improve the node classification performance. The first model included two additional convolutional layers (for a total of 3). The second model kept the three convolutional layers and included ReLU and dropout layers after each. The third model kept the three convolutional layers with ReLU and dropout and added two fully connected layers after. The fourth model added a ReLU layer inbetween the two final fully connected layers. The fifth model substituted the SAGEConv convolutional layer (Hamilton et al., 2017) instead of the GCNConv layer used previously. The sixth and final model again changed the

convolutional layer, this time to the GINConv (graph isomorphism convolutional layer) (Xu et al., 2018).

The reasoning for these changes is to implement small increases in complexity. Not every change was meant to result in an optimal model. Rather, the goal here was experimentation to observe performance with models even when we expect them to have serious flaws. Many state of the art convolutional network models (in and out of the GNN realm) involve multiple convolutional layers. Typically, each convolutional layer is followed by an activation function. AlexNet (Krizhevsky et al., 2017) (designed for computer vision tasks) is a key example of this. AlexNet's architecture includes several convolutional layers each followed by a ReLU layer. ReLU layers (and activation function layers in general) can provide increases in performance by applying non-linear transformations to the linear transformations provided by the other layers in the network. Furthermore, the ReLU activation function layer solves the problem of vanishing gradients. Furthermore, adding dropout an prevent overfitting in a model. Units are chosen at random to drop out of the network. Models 1 and 2 were implemented to observe the effect of adding more convolutional layers alone and adding them with ReLU activation and dropout. Models 3 and 4 were, again, attempts to increase the complexity of the network. More fully connected layers allows the newtwork to estimate more complicated functions. Finally, in models 5 and 6 we experimented with SAGEConv and GINConv. SAGEConv is an inductive framework that uses a given node's features as well as the features of nodes in a surrounding neighborhood to learn a function for creating embeddings. This means that node embeddings can be generated for previously unseen nodes. In our node classification task, we have bag of word representation features. Thus we use SAGEConv to see if these different method for node embeddings can improve performance. GINConv provides yet another framework for graph representation. We employ it to see if it provides any improvement.

Model	Architecture	Test Accuracy
Baseline	1xGCNConv	.741
1	3xGCNConv	.722
2	3x(GCNConv+ReLU+Drop)	.749
3	3x(GCNConv+ReLU+Drop) + 2xLinear	.750
4	3x(GCNConv+ReLU+Drop) + Linear+ReLU+Linear	.742
5	3x(SAGEConv+ReLU+Drop) + 2xLinear	.739
6	3x(GINConv+ReLU+Drop) + 2xLinear	.772

Table 1: GNN architectures and testing accuracies.

4 Results

For each model (including the basic architecture), the full graph was used for training. 200 epochs were used. Figure 1 shows training accuracies for at each epoch for each of the seven total models. The baseline model and model 1 appear to have the most stable trajectories without much variation from epoch to epoch. These are fairly simple models so this makes sense. Model 2 peaks at an early epoch and then decreases with little variation after. Model 6 (GINConv) shows the clear best training trajectory. Around epoch 100, it rises above the other models and stays there. To further evaluate each model, the full graph was used as a test set with nodes masked at random and predicted. Table 1 shows testing accuracies for each model. The superior performance of Model 6 is also reflected here as it has the highest test accuracy. Interestingly, the baseline model and Models and 2 are superior to models 4 and 5 which show instability across epochs. This suggest

that, while the GINConv layers with ReLU and dropout plus 2 fully connected layers offers clear improvement, using SAGEConv or adding an extra ReLU layer between the fully connected layers leads to unreliable models. Model 2 had the lowest testing accuracy but offered more stability.

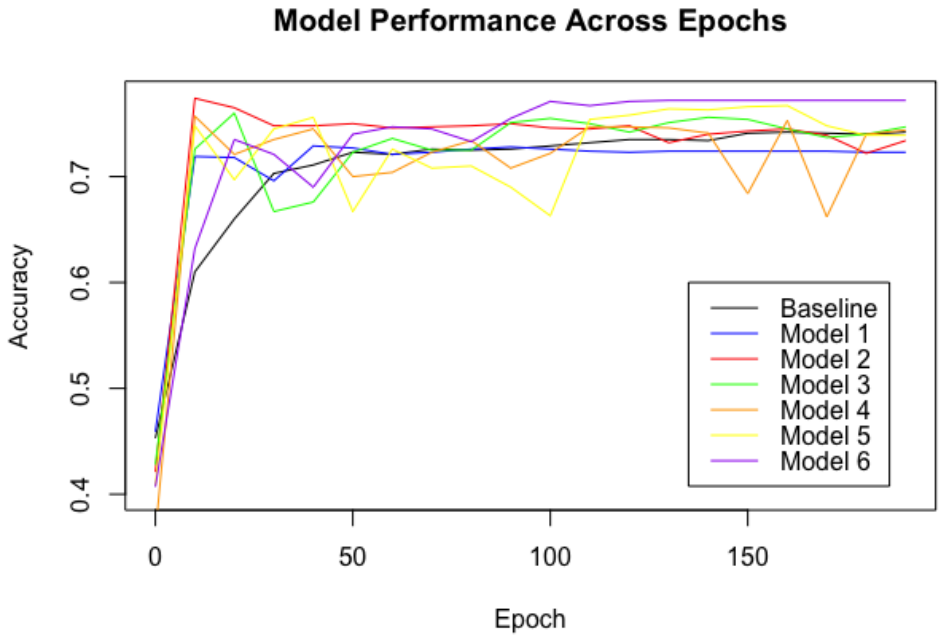
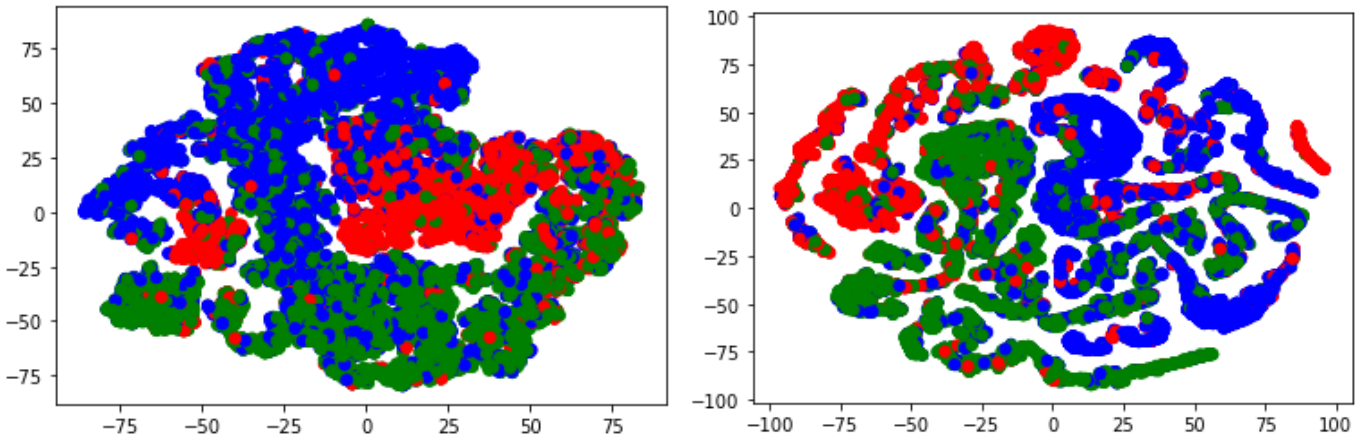


Figure 1: Model performance trajectory during training.

Finally, Models 2 and 7 were visualized. Node embeddings were extracted from the final convolutional layer of each and plotted using TSNE (a tool for visualizing high dimensional data). Figure 2 shows side by side comparisons of these visualizations. Models 2 and 7 were chosen because they represent the worst and best testing performances respectively. It is of interest to compare their node embedding visualizations. Both plots show a good level of segregation between the colors, but the higher level of segregation in the figure representing Model 6 is in line with the higher complexity and better performance of the model.

5 Discussion

Based on these analyses, it is clear that GNN architecture can have an impact on performance. Not only does increasing the overall complexity of the model (i.e. including more layers) improve the model, but using other types of convolution besides the standard GCN framework can have a real impact. It must be noted, however, that simply increasing complexity without discrimination does not necessarily mean better performance. One unexpected element of this analysis was the seeming instability of certain models that jump up and down in accuracy from epoch to epoch. In the future it would be interesting to experiment with different convolutional layers and more varying levels of complexity in the subsequent layers.



(a) Model 2 node embedding visualization

(b) Model 6 node embedding visualization

Figure 2: TSNE node embedding visualizations for model 2 (left) and model 6 (right).

6 Link to Code

All preprocessing and analysis code can be found at the following link:
https://github.com/SohailNizam/GraphDataMining-MP2

References

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034.

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. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.