Effect of Combining Graph Convolutions

Samuel Bortolin, 221245
DISI, University of Trento
samuel.bortolin@studenti.unitn.it

Abstract

The combination of different approaches recently demonstrated a very used strategy on the OGB datasets for seeking to improve the performance by using already existing approaches and finding the optimal way of letting these work together. In this work, I proposed some strategies for combining different graph convolution layers in different ways. I tested out the different strategies with different number of layers on the arXiv dataset. As a result, I obtained some improvements by combining the approaches with respect to the ones alone considering the same number of parameters.

1. Introduction

The Open Graph Benchmark (OGB) is a collection of realistic, large-scale, and diverse benchmark datasets for machine learning on graphs [6]. OGB provides a Python library[1] that allows downloading, processing, and splitting datasets using the OGB Data Loader. The model performance can be easily evaluated using the OGB Evaluator. OGB supports multiple task categories, in particular, the three fundamental graph machine learning task categories are covered: predicting the properties of nodes, links, and graphs.

I chose the Node Property Prediction as task typology, i.e., the task to predict properties of single nodes. As the node property prediction dataset, I chose the ogbn-arxiv dataset since it is small enough not to require powerful GPUs. It is a directed graph, representing the citation network between all Computer Science arXiv papers indexed by Microsoft Academic Graph (MAG) [14]. Each node is an arXiv paper and each directed edge indicates that one paper cites another. Each paper is described with a 128-dimensional feature vector obtained by averaging the embeddings of words in its title and abstract. The embeddings of individual words are computed by running the skip-gram model [9] over the MAG corpus. In addition, all papers are also associated with the year that the corresponding paper was published. This information is used for splitting data as follows:

- Train set: composed of the papers published until 2017;
- Validation set: composed of the papers published in 2018;
- Test set: composed of the papers published since 2019.

The relevant data details are the following:

- Number of nodes: 169343;
- Features per node: 128;
- Number of edges: 2315598;

[1] https://pypi.org/project/ogb/

- Average node degree: 13.674;
- The graph does not contain isolated nodes;
- The graph does not contain self-loops;
- During the processing I transformed it as an undirected graph using the *to_symmetric* method on the adjacency matrix, this allows to gain information from both papers that are cited in the current paper and the ones that cite it;
- Number of training nodes: 90941 (training node label rate: 0.537);
- Number of validation nodes: 29799 (validation node label rate: 0.176);
- Number of testing nodes: 48603 (test node label rate: 0.287).

The specific task on this dataset consists in predicting the 40 subject areas of arXiv computer science papers, e.g., cs.AI, cs.LG, and cs.OS, which are manually labeled by the paper's authors and arXiv moderators. This is a very important application since the volume of scientific publications was doubling every 12 years over the past century. It is practically important to automatically classify each publication's areas and topics. Formally, the task is to predict the primary categories of the arXiv papers, which is formulated as a 40-class classification problem. As an evaluation metric, accuracy is proposed. The general setting in which ML models are used is that they are trained on existing papers and then used to predict the subject areas of newly-published papers helping the arXiv moderators.

2. State of the art

Recently, many studies were proposed for extending deep learning approaches for graph data. The standard and most basic approaches used to solve node classification, which I also used in my analyses, are the Graph Convolutional Network (GCN) [7] [8] and the GraphSAGE framework (SAmple and aggreGatE) [5].
GCN kicked off geometric deep learning with the introduction of the GCNConv layers[2], where the convolution operation is applied to the graph adjacency matrix and the node features to extract features from the graph structure. Instead, the GraphSAGE framework was proposed for inductive node embedding by using SAGEConv layers[3], the convolution operation is performed by aggregating the node's neighborhood information by applying a set of transformation functions.

Many relevant improvements over them have been proposed. The Residual Gated Graph ConvNet [1] introduced gated edges and residuality. The Graph Attention Network [13] [2] introduced the attention mechanism to GCN proposing masked self-attentional layers. The Graph Transformer Network [11] adapted the concept of the transformer to work with graph data. Also networks with a strong theoretical background were proposed, such as the Graph Isomorphism Network [15] which discriminative/representational power is theoretically validated to be equal to the power of the WL test. There are not only architectural advancements but also some related to the way of aggregating information. For example, the Principal Neighborhood Aggregation Network [3] aimed to combine multiple aggregation strategies to improve the performance of the GNNs. Furthermore, the Learning Aggregation Function Network [10] aimed to learn the best aggregation function for the task rather than adopting the widely used aggregators.

[2] https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.conv.GCNConv

[3] https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.conv.SAGEConv

The combination of different approaches recently demonstrated a very used strategy on the OGB datasets for seeking to improve the performance by using already existing approaches and finding the optimal way of letting these work together. In fact, currently the state of the art and the most performing network according to the arXiv leaderboard on OGB is a combination of EnGCN (Ensembling GCN) [4] that integrates also the Self Label Enhanced (SLE) Training technique [12], with the GLEM framework (Graph and Language Learning by Expectation Maximization) [16].

3. Methodology

In this chapter are described the proposed approaches for combining the different graph convolution layers.

The first idea is MixedSeriesGCN, this combination method uses different graph convolution blocks in series, one block after the other in the same network. The schematic representation of this combination mechanism is illustrated in figure 1.

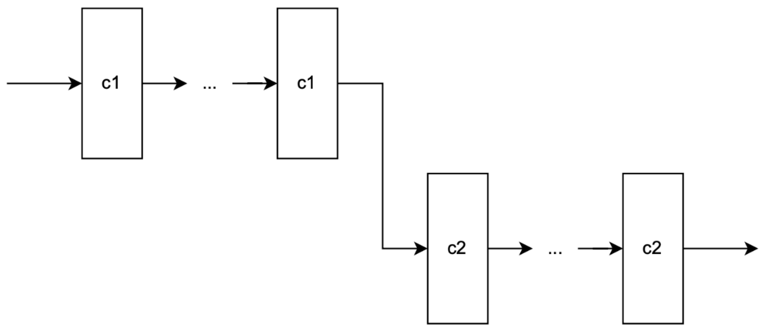

Figure 1: Schematic representation of the MixedSeriesGCN combination mechanism.

Dually, SwitchedMixedSeriesGCN is obtained by switching the order of the c1 and c2 blocks in the MixedSeriesGCN network.

Instead of combining graph convolution blocks, SwitchGCN continuously alternates the layers of the approaches. The schematic representation of this combination mechanism is illustrated in figure 2.

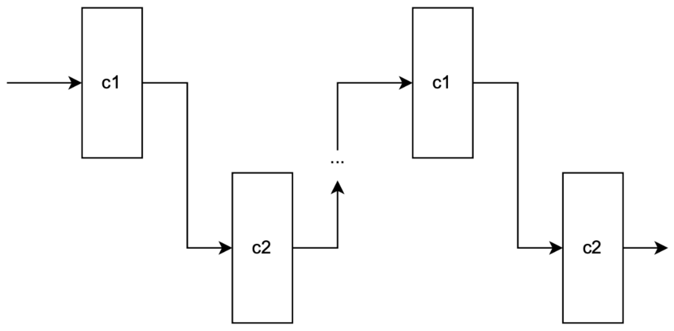

Figure 2: Schematic representation of the SwitchGCN combination mechanism.

Dually, SwitchedSwitchGCN is obtained by switching the order of the c1 and c2 layers in the SwitchGCN.

Trying out a parallel combination, MixedParallelGCN combines the outputs of the different approaches by averaging the outputs of the single approaches. The schematic representation of this combination mechanism is illustrated in figure 3.

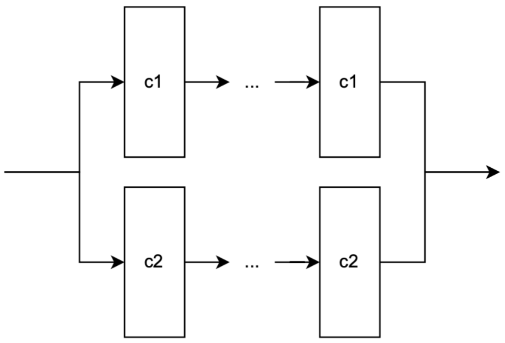

Figure 3: Schematic representation of the MixedParallelGCN combination mechanism.

Instead of combining the outputs just at the end, MixedGCN continuously combines the outputs of the approaches at each layer by averaging the outputs of the single approaches. The schematic representation of this combination mechanism is illustrated in figure 4.

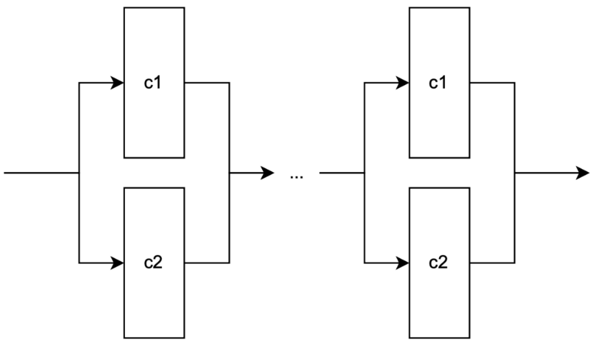

Figure 4: Schematic representation of the MixedGCN combination mechanism.

For the analyses, I used the simplest possible graph convolutions, just for the purpose of demonstrating the effectiveness of the different approaches combinations. As c1 I used GCNConv from torch_geometric.nn (improved and not cached version) and SAGEConv from torch_geometric.nn (with mean aggregation and without normalization) as c2.
With these analyses, it is interesting to understand whether it is possible to combine the advantages given by each of the considered approaches. In the considered case, GCNs can effectively extract global graph features, but they have a high computational cost and are prone to over-smoothing. On the other hand, GraphSAGE performs well with lower computational costs but can miss some important information from the graph structure.

The main objective of the analyses is to study the effect of the different combination mechanisms and compare the obtained results. For a fair comparison, the analyses are made on the same number of parameters, this allows us to understand which combination mechanism can produce the best performance. In particular, there is to notice that for the series combinations (MixedSeriesGCN, SwitchedMixedSeriesGCN, SwitchGCN, SwitchedSwitchGCN) neighbors at more hops will be considered by one or the other approach. Instead, for the parallel combinations (MixedParallelGCN, MixedGCN), neighbors at fewer hops will be considered, but they are considered in different ways by both approaches.

4. Results

I started my analyses by testing out some networks using the basic and most simple approaches: GCN and GraphSAGE. The results of these approaches obtained over 5 different runs and by varying the number of hidden layers are reported in table 1.

Approach	Hidden layers	Input fc layer	Params	Train	Validation	Test
GCN	6	Yes	121512	79.28%	72.91%	71.63%
		No	105768	78.94%	73.00%	71.94%
	12	Yes	222888	76.87%	72.26%	71.20%
		No	206376	77.04%	72.17%	71.09%
SAGEGCN	3	Yes	121128	79.74%	71.52%	70.36%
		No	104616	83.12%	72.15%	70.87%
	6	Yes	220584	77.44%	72.30%	71.34%
		No	204072	80.72%	72.83%	71.39%

Table 1: The results obtained by the basic approaches changing the number of hidden layers.

The results suggest that GCN tends to work better with few hidden layers and instead, SAGEGCN with more hidden layers, which actually is the same number but this equals twice the number of parameters.

The results of the proposed approaches for combining the graph convolutions obtained over 5 different runs and by varying the number of hidden layers are reported in table 2.

The best combination technique in training is also the worst in testing: the MixedParallelGCN, this suggests that considering hops in different ways does not generalize well on validation and test sets compared to the other combination strategies, but tends more to overfit the training data.
The best combination technique in testing turns out to be the SwitchGCN, which actually is also the one with the worst training accuracy.

The evolution of validation and test accuracies of these networks looks a bit noisy, probably due to the input fc layer. An example of this can be seen in figure 5 where these are reported for SwitchGCN with 4 hidden layers.

Approach	Hidden layers	Params	Train	Validation	Test
MixedSeriesGCN	2	121512	79.85%	73.09%	71.84%
	4	221352	78.22%	72.77%	71.45%
	6	321192	76.09%	72.30%	71.24%
SwitchedMixedSeriesGCN	2	121512	78.36%	72.20%	71.59%
	4	221352	79.15%	72.22%	71.22%
	6	321192	77.88%	71.81%	70.52%
SwitchGCN	2	121512	77.57%	73.25%	71.98%
	4	221352	77.04%	73.04%	71.68%
	6	321192	74.95%	71.94%	71.14%
SwitchedSwitchGCN	2	121512	77.50%	72.24%	71.46%
	4	221352	77.54%	72.41%	71.43%
	6	321192	76.62%	72.18%	71.21%
MixedParallelGCN	2	121512	76.57%	68.38%	67.45%
	4	221352	85.03%	70.75%	69.38%
	6	321192	84.28%	70.55%	69.18%
MixedGCN	2	121512	77.78%	68.95%	67.96%
	4	221352	81.54%	72.42%	71.32%
	6	321192	83.80%	72.75%	71.29%

Table 2: The results obtained by the proposed approaches changing the number of hidden layers.

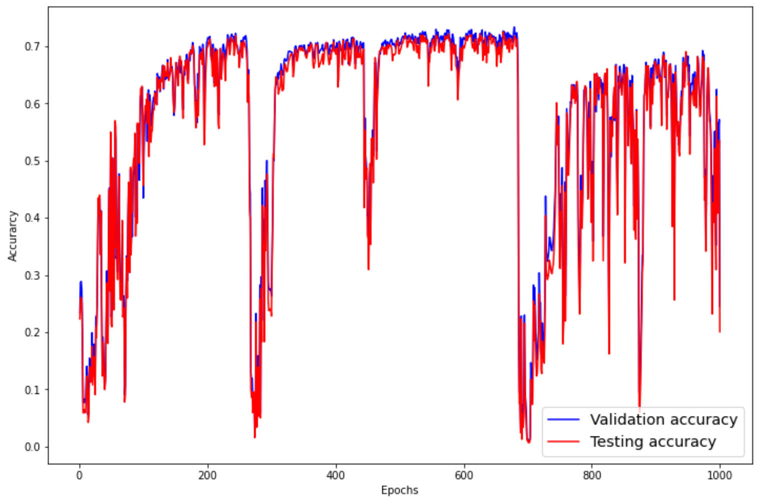

Figure 5: The evolution of the validation and test accuracies for SwitchGCN with 4 hidden layers.

The results of the proposed approaches removing the input fc layer obtained over 5 different runs and by varying the number of hidden layers are reported in table 3.

Approach	Hidden layers	Params	Train	Validation	Test
MixedSeriesGCN	2	105000	79.19%	73.09%	71.69%
	4	204840	79.18%	73.04%	71.40%
	6	304680	76.92%	72.39%	71.54%
SwitchedMixedSeriesGCN	2	105000	80.70%	72.83%	71.65%
	4	204840	80.68%	72.73%	71.63%
	6	304680	79.15%	72.48%	71.01%
SwitchGCN	2	105000	79.99%	73.23%	72.04%
	4	204840	78.86%	73.13%	71.77%
	6	304680	77.00%	72.85%	71.66%
SwitchedSwitchGCN	2	105000	80.54%	72.95%	71.85%
	4	204840	79.56%	73.05%	71.95%
	6	304680	78.97%	72.60%	71.37%
MixedParallelGCN	2	105000	78.22%	68.81%	67.78%
	4	204840	82.34%	70.74%	69.76%
	6	304680	83.17%	70.51%	69.05%
MixedGCN	2	105000	75.91%	68.88%	67.94%
	4	204840	82.38%	72.75%	71.53%
	6	304680	82.15%	72.82%	71.68%

Table 3: The results obtained by the proposed approaches removing the input fc layer changing the number of hidden layers.

In most of the approaches, it is possible to notice that the validation and test accuracies seem very stable after the initial transient of around 200 epochs. An example of the evolution of these accuracies for SwitchGCN with 2 layers is reported in figure 6.

Also in this case, MixedParallelGCN is very good in training but is the worst in testing, this seems to confirm what was analyzed with the input layer.
The best combination technique in testing turns out to be the SwitchedSwitchGCN, suggesting that the idea of switching the approaches can be a good idea.

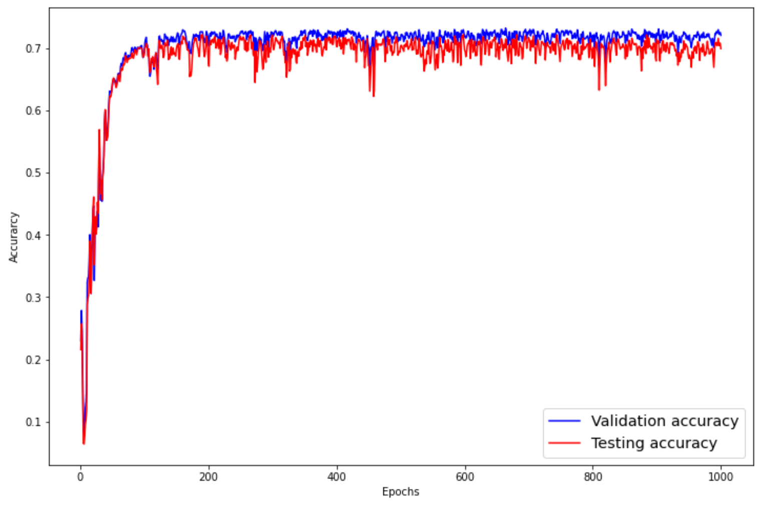

Figure 6: The evolution of the validation and test accuracies for SwitchGCN with 2 hidden layers and without input fc layer.

SwitchGCN with 2 layers showed the best validation and test performance:

- 73.25% on the validation set and 71.98% on the test set (with input fc layer);
- 73.23% on the validation set and 72.04% on the test set (without input fc layer).

The best proposed solution in testing achieves 72.04% of accuracy, while the best proposed on the OGB leaderboard scores 79.66%, for doing a comparison with the leaderboard, this model should obtain the 50th place among the loaded models.

In general increasing parameters seems to decrease the performance (with some exceptions) and few layers are sufficient for obtaining reasonable results and slightly better accuracy compared to the ones alone. We can deduce that the models do not need more expressive power to perform better and instead they will tend to overfit the training data due to the curse of dimensionality,
The results without input fc are very similar to the ones with it, but they are a bit better in most of the networks and produce smoother validation and test accuracy curves. We can infer that an initial transformation of the data in general does not improve results, and using only convolutions may provide better results.

5. Conclusions

I proposed some strategies for combining different graph convolution layers in different ways. I tested out the different strategies with different number of layers on the arXiv dataset. The combination of the approaches showed some improvements with respect to the ones alone considering the same number of parameters.
As demonstrated also by the different OGB leaderboards, the combination of different strategies can be a good idea for seeking to improve the performance by using already existing approaches and finding the optimal way of letting these work together.
In fact, also combining simple graph convolutions can lead to better results compared to using the approaches in isolation.

References

[1] Bresson, Xavier, and Thomas Laurent. “Residual Gated Graph ConvNets.” *arXiv*, 20 Nov 2017, https://doi.org/10.48550/arXiv.1711.07553.

[2] Brody, Shaked, et al. “How Attentive are Graph Attention Networks?” *arXiv*, 30 May 2021, https://doi.org/10.48550/arXiv.2105.14491.

[3] Corso, Gabriele, et al. “Principal Neighbourhood Aggregation for Graph Nets.” *arXiv*, 12 Apr 2020, https://doi.org/10.48550/arXiv.2004.05718.

[4] Duan, Keyu, et al. “A Comprehensive Study on Large-Scale Graph Training: Benchmarking and Rethinking.” *arXiv*, 14 Oct 2022, https://doi.org/10.48550/arXiv.2210.07494.

[5] Hamilton, William L., et al. “Inductive Representation Learning on Large Graphs.” *arXiv*, 7 Jun 2017, https://doi.org/10.48550/arXiv.1706.02216.

[6] Hu, Weihua, et al. “Open Graph Benchmark: Datasets for Machine Learning on Graphs.” *arXiv*, 2 May 2020, https://doi.org/10.48550/arXiv.2005.00687.

[7] Kipf, Thomas N., and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks.” *arXiv*, 9 Sep 2016, https://doi.org/10.48550/arXiv.1609.02907.

[8] Li, Guohao, et al. “DeeperGCN: All You Need to Train Deeper GCNs.” *arXiv*, 13 Jun 2020, https://doi.org/10.48550/arXiv.2006.07739.

[9] Mikolov, Tomas, et al. “Distributed Representations of Words and Phrases and their Compositionality.” *Advances in Neural Information Processing Systems*, 2013, https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[10] Pellegrini, Giovanni, et al. “Learning Aggregation Functions.” *arXiv*, 15 Dec 2020, https://doi.org/10.48550/arXiv.2012.08482.

[11] Shi, Yunsheng, et al. “Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification.” *arXiv*, 8 Sep 2020, https://doi.org/10.48550/arXiv.2009.03509.

[12] Sun, Chuxiong, et al. “Scalable and Adaptive Graph Neural Networks with Self-Label-Enhanced Training.” *arXiv*, 19 Apr 2021, https://doi.org/10.48550/arXiv.2104.09376.

[13] Veličković, Petar, et al. “Graph Attention Networks.” arXiv, 30 Oct 2017, https://doi.org/10.48550/arXiv.1710.10903.

[14] Wang, Kuansan, et al. “Microsoft Academic Graph: When experts are not enough.” *Quantitative Science Studies*, 2020, https://doi.org/10.1162/qss_a_00021.

[15] Xu, Keyulu, et al. “How Powerful are Graph Neural Networks?” *arXiv*, 1 Oct 2018, https://doi.org/10.48550/arXiv.1810.00826.

[16] Zhao, Jianan, et al. “Learning on Large-scale Text-attributed Graphs via Variational Inference.” arXiv, 26 Oct 2022, https://doi.org/10.48550/arXiv.2210.14709.