Reddit GCNN - Implementation of Graph Convolutional Neural Networks in Graph Classification problem

Luka Nedimović
Faculty of Sciences, University of Novi Sad, Novi Sad
Faculty of Computing, Union University, Belgrade

Abstract

In this short paper, we aim to explore the application of graph convolutional networks (**GCN**) [2] on the problem of graph classification. We implement a straightforward architecture, consisting of several GCN layers, followed by a multi-layer perceptron (**MLP**). In the end a comparative analysis, based on type of graph convolution, embedding dimensions and activation functions is conducted.

1 Introduction

Graphs are useful structures to represent relations among data. Although different architectures can be used for inference, some classes of data can benefit of understanding and encoding relational information, such as social networks, molecules, road networks and so on.
Formally, a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ consists of a set of vertices $\mathbf{V}$ and a set of edges $\mathbf{E}$, where each edge is an unordered pair of distinct vertices from $\mathbf{V}$. Additionally, the *neighborhood* of a node $\mathbf{v}$ in a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, denoted by $\mathcal{N}_{\mathbf{v}}$, is the set of all nodes that are adjacent to $\mathbf{v}$; that is, $\mathcal{N}_{\mathbf{v}} = \{\mathbf{u} \in \mathbf{V} \mid (\mathbf{v}, \mathbf{u}) \in \mathbf{E}\}$. Intuitively, by **aggregation** of information from neighbor vertices, we could **update** representation of each vertex and learn useful features of a complete graph, using them to create a new representation of the same graph more suited for the task at hand.

Graph convolutional networks do exactly that. By applying convolution operation on nodes in immediate neighborhood, neural network learns new embeddings, that can be used for the following:

- **Node-related problems**, such as **node regression** (e.g. income prediction in social network)
- **Edge-related problems**, such as **link prediction** (e.g. in recommendation systems to create new recommendations)
- **Graph-related problems**, such as **graph classification** (e.g. in drug testing)

2 Dataset

The dataset used is the **Reddit Threads** [3] - a simple dataset, where vertices represent users and edges between them represent replies. Complete dataset contains total of $203,088$ unique threads.

Below is an example entry:

edge_index	y	num_nodes
$[[0,1,2,2,2,2,2,2,2,2,3,4,5,5,6,7,8,8,9,10],$ $[2,5,0,4,5,6,7,8,9,10,8,2,1,2,2,2,2,3,2,2]]$	$[0]$	11

Table 1: Example entry in the **Reddit Threads** dataset

This entry directly corresponds to the following graph:

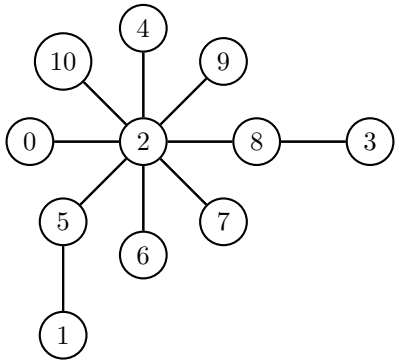

Figure 1: Example entry in **Reddit Threads** dataset

This thread is not labeled as a discussion, and we can conclude so by inspecting the degree of the node **2** - the thread is centered around a single person, while discussion would foster more diverse connections.

The minimum number of nodes of a single graph in dataset is $\mathbf{11}$, while the maximum is $\mathbf{97}$, indicating, we are handling relatively small graphs.

3 Proposed Architecture

We propose an architecture that consists of two parts - **GCN Block**, followed up by **MLP Block**.

GCN Block. We experiment with two kinds of convolutions - $\mathbf{GCNConv}$, proposed in [2] and $\mathbf{SAGEConv}$, proposed in [1].

The $\mathbf{GCNConv}$ operator, proposed in [2], is formulated as:

$$\mathbf{H}^{(l+1)} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(l)}\mathbf{W}^{(l)}$$

where:

- $\mathbf{H}^{(l)}$ is the matrix of activations in the l-th layer ($\mathbf{H}^{(0)} = \mathbf{X}$, the feature matrix).
- $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix of the graph with added self-connections (identity matrix $\mathbf{I}$).
- $\tilde{\mathbf{D}}$ is the degree matrix of $\tilde{\mathbf{A}}$.
- $\mathbf{W}^{(l)}$ is the trainable weight matrix of the l-th layer.

The $\mathbf{SAGEConv}$ operator, proposed in [1] (as the $\mathbf{GraphSAGE}$), is formulated as:

$$\mathbf{x}_i^{'} = \mathbf{W}_1\mathbf{x}_i + \mathbf{W}_2 \cdot \text{mean}_{j\in\mathcal{N}_i}\mathbf{x}_j$$

where:

- $\mathbf{x}_i^{'}$ is the updated feature vector of node i.
- $\mathbf{W}_1$ is a trainable weight matrix that transforms the feature vector of node i.
- $\mathbf{x}_i$ is the original feature vector of node i.
- $\mathbf{W}_2$ is a trainable weight matrix that transforms the mean of the feature vectors of the neighboring nodes of i.
- $\text{mean}_{j\in\mathcal{N}_i}\mathbf{x}_j$ is the mean of the feature vectors of the neighboring nodes of i, denoted by $\mathcal{N}_i$.

Between each GCN layer there is an activation function (discussed in 4.2), and a dropout layer (30%) at the end of it.

MLP Block. We make use of standard **Multi-layer Perceptron Architecture**, with varying dimensionality and ReLU activation function between each layer. In the end, a Sigmoid function is applied due to the binary-classification nature of the task.

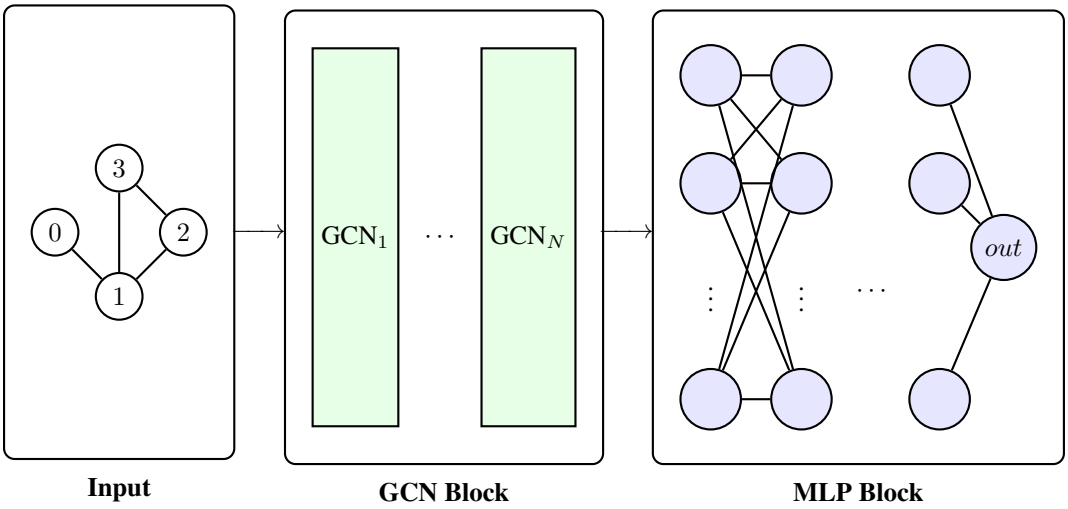

Figure 2: Model Architecture

4 Experiments

4.1 Experimental Settings

In the following table reside the general experimental settings, for reproduction purposes: Each

Component	**Details**
GPU	NVIDIA GeForce RTX 3060 (6GB)
CPU	Intel i7-12650H (16) @ 4.600GHz
RAM	16 GB DDR4
Python Version	Python 3.10.12
Learning Rate	0.001
Epochs	20

Table 2: Experimental Settings

experiment is conducted through respective script, where script's name comes in the format `{gcn_type}_{model_size}_{gcn_activation}.sh`. Result of each experiment is stored within `training_data.csv` file.

4.2 Experiment Descriptions

In this paper, along with **2** different graph convolutions, we experiment with **3** network sizes and **4** different activation functions.

Activation functions. Inside **GCN Block**, we experiments with $\mathbf{ReLU}$, $\mathbf{LeakyReLU}$, $\mathbf{Sigmoid}$ and $\mathbf{Tanh}$. Inside **MLP Block**, we apply $\mathbf{ReLU}$ in between each **FCL**, with a $\mathbf{Sigmoid}$ applied at the end.

Model sizes. The following table contains network sizes (**S** - small, **M** - medium, **L** - large):

Size	Graph Convolution Dimensions	MLP Dimensions
S	(16, 8, 4)	(4, 2, 1)
M	(256, 128, 128, 128, 64)	(64, 32, 16, 16, 1)
L	(2048, 2048, 2048, 2048, 1024, 1024, 512)	(512, 256, 128, 1)

Table 3: Architecture Sizes

4.3 Experiments Results

GCN Type	Size	GCN Activation	Train Accuracy	Test Accuracy
GCNConv	**S**	**ReLU**	51.2%	50.2%
		LeakyReLU	51.2%	50.2%
		Sigmoid	51.2%	50.2%
		Tanh	69.2%	62.6%
	M	**ReLU**	**92.4%**	**66.6%**
		LeakyReLU	91.4%	69.4%
		Sigmoid	80.0%	72.2%
		Tanh	88.2%	66.2%
	L	**ReLU**	89.2%	63.0%
		LeakyReLU	87.6%	65.4%
		Sigmoid	79.6%	71.0%
		Tanh	72.2%	66.0%
SAGEConv	**S**	**ReLU**	80.0%	68.6%
		LeakyReLU	79.0%	68.0%
		Sigmoid	51.2%	50.2%
		Tanh	79.0%	64.8%
	M	**ReLU**	84.8%	67.2%
		LeakyReLU	86.8%	68.8%
		Sigmoid	78.4%	64.2%
		Tanh	77.8%	64.8%
	L	**ReLU**	84.4%	67.8%
		LeakyReLU	76.4%	65.6%
		Sigmoid	51.2%	50.2%
		Tanh	**82.0%**	**72.2%**

Table 4: Experiment Results. We highlight the best performing model at train time (**GCNConv-M-ReLU**), and the best performing model at test time (**SAGEConv-L-Tanh**)

4.4 Ablation Study

Type of graph convolution. On average, **GCNConv** has achieved **75.28%** accuracy, while **SAGEConv** has achieved **75.91%** accuracy on **training data**. On **testing data**, **GCNConv** has achieved **62.75%**, while the **SAGEConv** has achieved **64.36%** average accuracy. We see diminishing returns considering only the type of graph convolution, with **SAGEConv** performing slightly better in both cases (i.e. by **0.63%** on training and **1.61%** on testing data).

Model size. **S-models** (i.e. "small" models), on average, learn a lot better with **SAGEConv**, achieving **16.6%** difference on training, and **9.6%** difference on testing data.
M-models, on average, learn better with **GCNConv**, by **6.05%** on training, and **2.35%** difference on testing data. **L-models**, on average, learn better with **GCNConv**, by **8.65%** on training, and **2.4%** difference on testing data.
However, **S-models** can be seen having trouble learning anything at all, capping at **51.2%** accuracy on training and **50.2%** accuracy on testing data, due to lack of dimensionality, from the very start of the training. **M-models**, paired with either graph convolution type, learn the best, as denoted in the highest average training and testing accuracies. **GCNConv-M** learns the best on average, with **88.0%** average training and **68.6%** testing accuracy.

Architecture	Average Training Accuracy	Average Testing Accuracy
GCNConv-S	55.7%	53.3%
GCNConv-M	**88.0%**	**68.6%**
GCNConv-L	82.1%	66.3%
SAGEConv-S	72.3%	62.9%
SAGEConv-M	81.9%	66.2%
SAGEConv-L	73.5%	63.9%

Table 5: Average training and testing accuracies for each model size

Type of GCN block activation. **Sigmoid** has shown the poorest performance, even radically collapsing the model (as in **SAGEConv-S** and **SAGEConv-L** cases). Other activation functions have achieved at least **70%** average training and **60%** average testing accuracy. **ReLU** learns the best on training data, with **80.33%** average accuracy, whilst **Tanh** generalizes the best, with **66.1%** average accuracy, on testing data. **Tanh**, also, seems to be capable of escaping low dimensionality, combined with **GCNConv**, by being the only one achieving more than **51.2%** training **/ 50.2%** testing accuracy. **ReLU** and **LeakyReLU** offer a very stable learning, with **ReLU** generally performing better on training, and **LeakyReLU** on testing. Check Appendix 5 for visualization.

GCN Block Activation	Average Training Accuracy	Average Testing Accuracy
ReLU	80.3%	63.9%
LeakyReLU	78.7%	64.6%
Sigmoid	65.2%	59.7%
Tanh	78.1%	66.1%

Table 6: Average training and testing accuracies for each GCN block activation

5 Conclusion

In this paper, we aimed to explore the effects of different types of graph convolution operations. We first introduced the model architecture and the experimental settings. After completing 24 experiments, we analyzed their synchronicity with **3** network sizes and **4** activation functions. Our findings include the following:

- Both **GCNConv** and **SAGEConv** produce results of negligible difference on small graphs
- Both graph convolution operations can be effectively used in a graph classification task, even on very limited data
- **Sigmoid** activation function can hinder the learning process, especially on small data, but can also collapse the model of higher dimensions
- Adding **Tanh** inbetween graph convolution layers, while increasing the dimensions of network, proved itself to be the best approach
- Relatively small graphs (i.e. maximum size being **97**) do not benefit much from small node embedding dimensions (i.e. dimensions being less than **256**)

Visualization of accuracies throughout training process can be found in Appendix 5.

References

[1] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *CoRR* abs/1706.02216 (2017). arXiv: 1706.02216. URL: http://arxiv.org/abs/1706.02216.

[2] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *CoRR* abs/1609.02907 (2016). arXiv: 1609.02907. URL: http://arxiv.org/abs/1609.02907.

[3] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar. "Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs". In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*. ACM. 2020, pp. 3125–3132.

Appendix

This appendix contains additional visualizations related to the study. These visualizations provide further insights into the performance and behavior of the models discussed in the main text. We suggest paying the attention of learning curve of models of small dimensions, and **Sigmoid** activation.

Training Plots

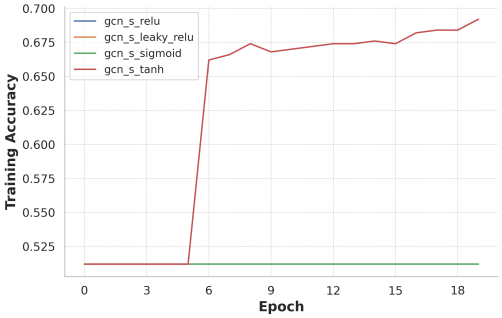

Figure 3: **GCNConv-S** training accuracy

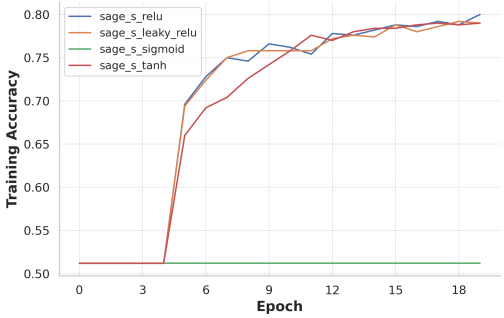

Figure 6: **SAGEConv-S** training accuracy

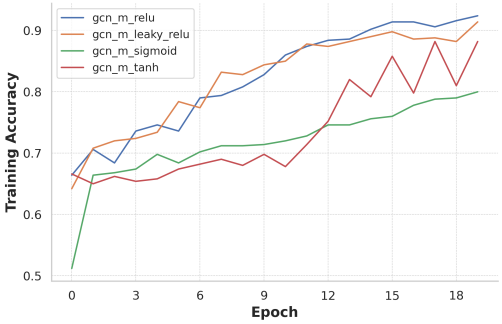

Figure 4: **GCNConv-M** training accuracy

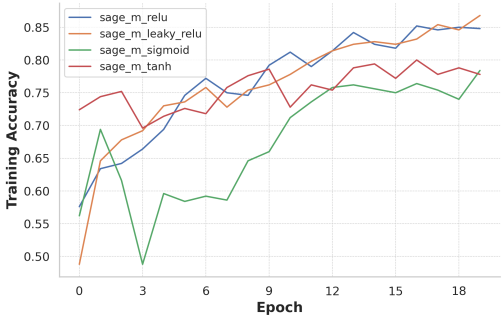

Figure 7: **SAGEConv-M** training accuracy

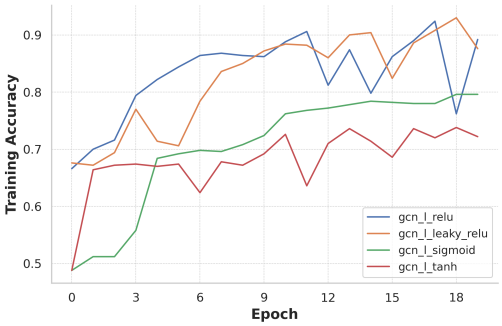

Figure 5: **GCNConv-L** training accuracy

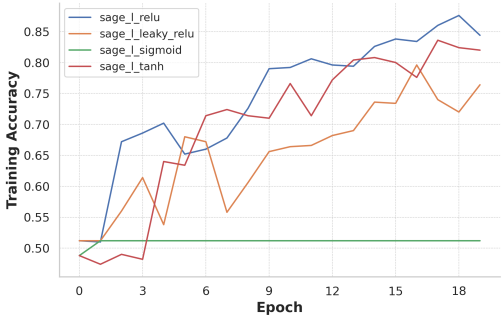

Figure 8: **SAGEConv-L** training accuracy

Testing Plots

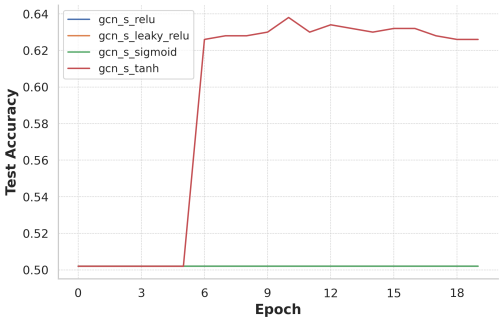

Figure 9: **GCNConv-S** testing accuracy

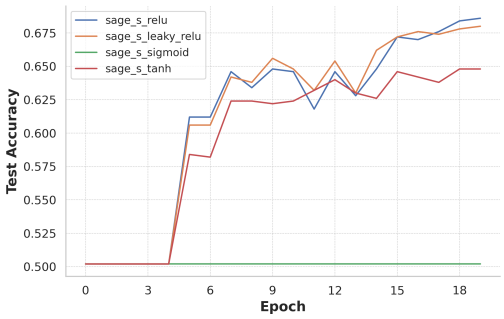

Figure 12: **SAGEConv-S** testing accuracy

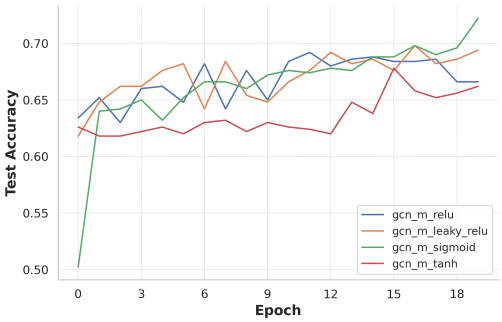

Figure 10: **GCNConv-M** testing accuracy

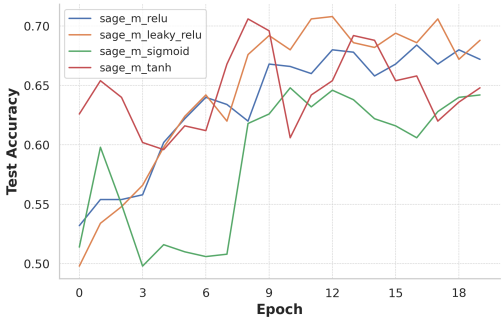

Figure 13: **SAGEConv-M** testing accuracy

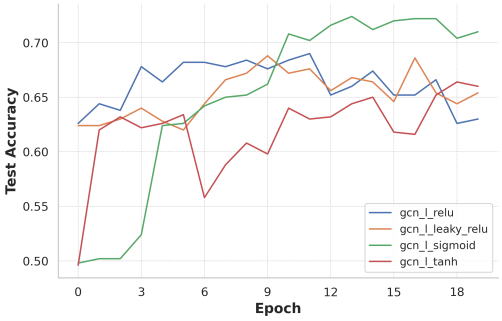

Figure 11: **GCNConv-L** testing accuracy

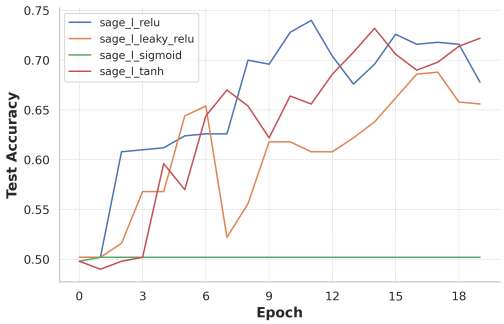

Figure 14: **SAGEConv-L** testing accuracy