Graph U-Nets variations

Alexandru Dochian[1][2776000]

Vrije Universiteit Amsterdam

Abstract. This study investigates variations the Graph U-Nets architecture. The research explores variations in pooling rates, depths, and introduces a trainable parameter for self-loops. The Encoder-Decoder blocks of Graph U-Nets are examined in terms of depths and stacking consecutive such blocks. We experimented solely on Cora citation graph in a semi-supervised learning setup.

Keywords: GCN · Grap U-Nets · Graph Pooling · Graph Unpooling · Trainable self loops · Semi-supervised Learning · Node Classification

1 Introduction

`https://github.com/alexandru-dochian/graph_u_nets_variations/`

Graph representation is an effective method to represent complex data. In this project we have worked only with citation graphs, where nodes represent articles and edges represent the citations from one paper to another. Given such data, we were tasked with finding a better vectorial representation for nodes such that different graph structures get embedded into the node itself. Having such an embedding for nodes allows for further downstream machine learning tasks like node classification.

In this project, we have set up a semi-supervised machine learning pipeline and experimented with different graph convolution methods and variations to learn better node representations which were further used for node classification tasks.

2 Related Work

Graph convolutional networks (GCNs) ([1]) achieved promising performance on graph classification tasks. They implement the convolution operation on graphs represented as matrices and manage to learn better node representations by aggregating data

Encoder-Decoder Architectures like U-Net ([2]) reached state-of the-art results in image segmentation through the usage of pooling and up-sampling operations. Such operations are natural for pixel-based inputs but cannot be directly applied on graph data which lack spatial locality information, given the adjacency representation.

Graph U-Nets ([3]) implements an encoder-decoder architecture on top of GCNs through the usage of gPool and gUnpool operations. These operations can only be used together to sample down a graph and then reconstruct it to the initial structure.

The sampling decision is based on a scoring function applied on the projection nodes features on the same trainable vector.

3 Methodology

We set up a semi-supervised machine learning task. The graph can be represented as a feature matrix X of size (num_nodes $\times$ num_features) and an adjacency matrix A of size (num_nodes $\times$ num_nodes). The dataset has been partitioned into training, validation, and testing using distinct masks, which identify specific subsets of nodes within the complete graph

Following blocks were used in the experiments:

1. GPoolBlock consists of a gPool layer followed by a GCN layer that operates on the sampled size
2. GUnpoolBlock consists of a gUnpool that reconstructs the initial graph structure followed by a GCN layer that executes the convolution on the expanded graph

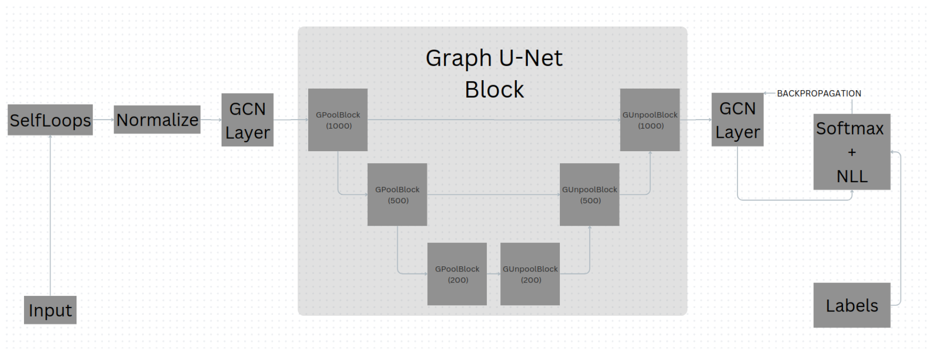

Fig. 1. Graph U-nets architecture

3.1 Trainable Self Loops

We have seen that in many previous experiments adding self loops to the adjacency matrix before the convolutions yielded promising results. In the Graph U-Nets work ([3]) 2 self loops were added $\hat{A} = A + 2I$. We have extended this feature by making it trainable for each node. A weight parameter with the size of the number of nodes was initialized with the values of 2, thus each node gets

2 self loops. During the backpropagation step, the amount of self loops added to each node will be individually updated with a custom learning rate different than standard learning rate used for the rest of the model parameters.

3.2 Depth Analysis

We can define multiple levels of pooling and unpooling. For instance, an encoder-decoder block defined by $k\_levels = [0.9, 0.5, 0.3]$ applied on a graph with 1000 nodes will mean that 900, 500 and 300 nodes are sampled in each **GPoolBlock**s and afterwards the **GUnpoolBlock**s will recreate the sampled graphs in the reverse order 500, 900 and 1000.

Besides the 'tallness' of the encoder-decoder block we have tested the concatenation of multiple such encoder-decoder blocks.

4 Results

We have used the Cora citation graph with following specifications:

- Nodes: 2708
- Degree: 4
- Features: 1433
- Classes: 7
- Training: 140
- Validation: 500
- Testing: 1000

All models were trained for 200 epochs with a learning rate of 0.01 for all parameters except the SelfLoops module which was updated with a learning rate of 0.5. A hidden layer size of 256 was used for all GCN layers with dropout rates of 0.2. For the GPool operations, a dropout rate of 0.1 was used.

With respect to the architecture presented in 3, if the number of Graph U-Net blocks is **0**, then the model is exactly a GCN with two GCN layers.

4.1 Trainable Self Loops

In table 1 we can see the comparison between models that had trainable or static self loops. With respect to the conducted experiments we observe that using a trainable SelfLoops operation slightly lowers the performance of the model.

Table 1. Self Loops Accuracy Results

Experiment Name	Epoch	Acc Train	Acc Test	Acc Val
GCN drop 0.2 not trainable self loops	200	0.73	0.70	0.70
GCN drop 0.5 not trainable self loops	200	0.69	0.66	0.66
GCN drop 0.2 trainable self loops	200	0.68	0.69	0.69
GCN drop 0.5 trainable self loops	200	0.66	0.64	0.64

4.2 Depth Analysis

Tallness of encoder-decoder blocks In this round of experiments we have played around with different a different number and pooling values of the GPool-Block - GUnpoolBlock pairs.

With respect to the experimental results presented in table 2 we observe that increasing the tallness of the encoder block slightly decreases performance.

Table 2. Tallness Accuracy Results

Experiment Name	Epoch	Acc Train	Acc Test	Acc Val
Graph U-nets drop 0.2 $k = [0.9]$	200	0.78	0.758	1.0
Graph U-nets drop 0.2 $k = [0.9, 0.7]$	200	0.767	0.754	1.0
Graph U-nets drop 0.2 $k = [0.9, 0.7, 0.5]$	200	0.712	0.708	0.86

Number of encoder-decoder blocks We have chosen the best performing Graph U-nets architecture from the previous batch of experiments and stacked multiple such blocks. Again, with respect to the conducted experiments we observe in 3 that stacking multiple Graph U-Net Encoder-Decoder blocks lowers performance.

Table 3. Number of Graph U-Net blocks Accuracy Results

Experiment Name	Number of Blocks	Epoch	Acc Train	Acc Test	Acc Val
Graph U-nets drop 0.2 $k = [0.9]$	1	200	0.78	0.758	1.0
Graph U-nets drop 0.2 $k = [0.9]$	2	200	0.723	0.714	0.86
Graph U-nets drop 0.2 $k = [0.9]$	3	200	0.452	0.452	0.57

5 Discussions and Conclusions

In this study, we have investigated various modifications to the Graph U-Nets architecture with the goal of improving its performance in capturing complex structures within citation graphs. The experiments involved testing different pooling rates, depths, and the introduction of trainable self-loops.

Generally, with respect to our conducted experiments, the proposed variations lowered performance but further experimentation is required to prove their ineffectiveness.

References

1. Berg, R. v. d., Kipf, T. N., & Welling, M. (2017). Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263.*

2. Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III*, pages 234–241. Springer.
3. Gao, H., & Ji, S. (2019). Graph u-nets. In *International Conference on Machine Learning*, pages 2083–2092. PMLR.