Augmenting Node Information for Homogeneous Graphs

Jonathan Ling
Department of Management Science and Engineering
Stanford University
jonling@stanford.edu

Kevin Ji
Department of Computer Science
Stanford Univeristy
kevinji@stanford.edu

Wes Peisch
Department of Mechanical Engineering
Stanford University
wpeisch@stanford.edu

Abstract

Graph Neural Network (GNN) aggregation schemes involve recursively aggregating representations of network nodes, but popular methods such as mean-pooling in Graph Convolutional Networks (GCNs) and max-pooling in GraphSAGE are not able to adequately distinguish some basic graph structures, including symmetrical ones. In our paper, we use the link prediction task on the drug-drug interaction dataset from Open Graph Benchmark to analyze the impacts of adding different types of information. We investigate the usage of different random initializations of embeddings, which are updated across epochs, static node information that is constant across epochs, and pre-trained model embeddings. Compared to using a uniform distribution for the embeddings, we determine that a combination of binarized k-hop, cycle, and subgraph features with static Xavier initialization performs significantly better than the baseline. Further work with better initializations could significantly improve existing models.

1 Introduction

Graph Neural Network (GNN) aggregation schemes involve recursively aggregating representations of network nodes, but popular methods such as mean-pooling in Graph Convolutional Networks (GCNs) and max-pooling in GraphSAGE are not able to adequately distinguish some basic graph structures. Xu et al. [2019] showed that Graph Isomorphism Networks (GIN), a neural aggregation function, is provably most expressive among GNNs with no distinct node features. However, this class of GNNs is not able to distinguish between some simple non-isomorphic subgraphs, such as those with different cycle lengths.

To solve this, we propose three different types of information we can add. The first is randomly generated embeddings. Since the primary method of learning edge-detection models depends on gradient descent, it is possible to get stuck in local minima as analyzed in Lin et al. [2020]. As such, it is important to have good embedding initializations. We implement an architecture from Sato et al. [2020] which is innovative in its addition of random features to nodes in the network in order to distinguish nodes to help with tasks such as detecting cycles, which can be a difficult task for GNNs. We test numerous changes to this strategy– initializing embeddings using He et al. [2015] weight formula for embedding initialization and using the strategy outlined in Glorot and Bengio [2010] and setting node features as one-hot encoding, using node degree, the number of cycles it participates in and the number of subgraphs it belongs to.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

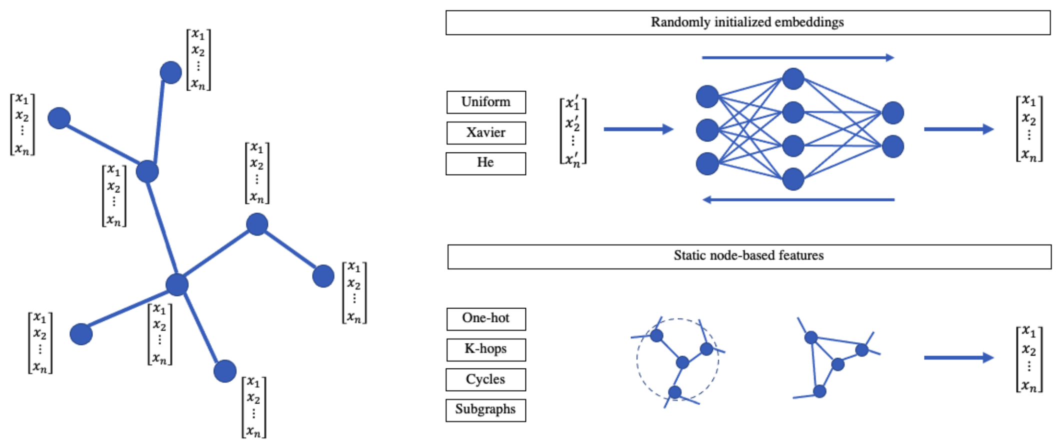


Figure 1: Initializing embeddings and node features using various randomization methods and node characteristics respectively.

We also tried adding static feature information to each node. This could potentially solve the issue of nodes being treated as symmetric. Two methods we propose are concatenating the binarization of the 1 and 2 hop neighbor counts, and concatenating the binarization of cycle counts.

Finally, one last method we used was to pretrain on a certain subset of nodes, save those embeddings, and use these to finetune the model on the rest of the training set. As seen in PAPER, pre-training has proven to be an effective method in allowing the model perform better.

To test all of these approaches, we used the drug-drug interaction dataset from the Open Graph Benchmark, and used the Hits@K metric for link prediction.

2 Existing Method

In the paper by Sato et al. [2020] the addition of random node features is used to make the model's results more accurate. It does so by allowing the model to distinguish between certain non-isomorphic graphs that GNNs have historically not been able to distinguish between. When testing the approach on GINs (graph isomorphism networks), rGINs (GINs with random features) have the random features added to each node at every iteration. The addition of random features allow the graph to detect cycles and also learn a randomized algorithm as opposed to a deterministic one, so that the output isn't exactly the same every time. This paper proves that rGIN output is close to the optimal solution for a graph of any size, which separates it from preceding papers like those by Dasoulas and Keriven, which bounded the graph size.

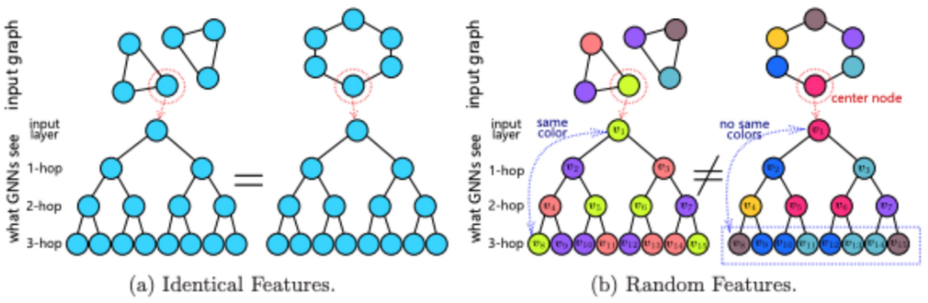


Figure 2: Existing method based on Sato et al. [2020]

As our chosen dataset, the drug-drug interaction dataset, has no node features, we will to generate random node features or embeddings, then apply a GNN architecture for aggregation. In addition to implementing this, we aim to build on this work by considering more informed methods of initializing node embeddings or features, such as with cycle length information and pre-training.

3 Method

The multiple ideas expressed in this paper fall into two categories: changing embedding initializations and changing the node feature assignment strategy.

3.1 Embedding Strategies

Our baseline model creates a random embedding matrix using a uniform distribution, with 256 columns per node. The objective of this is guaranteeing that unique embeddings are calculated on all parts of the graph regardless of structure. As a result, cycles are more easily identified than if those additional features had not been added. This model is referred to throughout the paper as either "baseline" or "uniform".

In our paper, we also try two other random embedding strategies discussed in Kocmi and Bojar [2017].

The first of which He initialization, from He et al. [2015], described below:

$$W \sim \mathcal{N}\left(0, \frac{2}{n_i}\right)$$

The second was Xavier initialization from Glorot and Bengio [2010], using the formula:

$$W \sim \left[\frac{\sqrt{6}}{\sqrt{n_i + n_o}}; \frac{\sqrt{6}}{\sqrt{n_i + n_o}}\right]$$

Kocmi and Bojar [2017] applied these initializations for word embeddings. As such, we believe that it would potentially be appropriate for us to use them for our node embeddings. They also concluded that Xavier initialization performed better than He initialization, which we also expect to occur here.

3.2 Node Feature strategies

We used four fundamental methods for generating node features from node characteristics: one-hot, k-hops, cycles and subgraphs.

The one-hot feature vector assigned each node a unique label in the graph. To have a more compact representation than the traditional one-hot encoding which would use a feature vector size of the number of nodes $|V|$, and to be consistent with the baseline embedding size of 256, we encoded each node with the binary representation of its (arbitrarily assigned but unique) index $i = 0, ..., |V| - 1$.

The k-hops feature vector counted the size of a k-radius neighborhood. We concatenated the binarized counts of 1-hop and 2-hop neighborhoods sizes. We chose these low number of hops as the graph was extremely dense and became fully connected (and therefore not informative) only after a handful of hops.

The cycles feature vector counted how many times each node appeared in a cycle of length 2, 3, 5 and 7, then concatenated the binarized representation of these counts. We chose cycles of prime length for quick computation using powers of the adjacency matrix, without the need to subtract out smaller cycle counts within larger cycles when using this matrix.

The subgraphs feature vector counted the number of undirected, anchored subgraphs of size 2 and 3. Specifically, it counted the four such subgraphs: size-2 links, size-3 cycles, size-3 chains with the anchor at one end, and size-3 chains with the anchor in the middle. It then concatenated the binarized counts. We did not enumerate larger sizes due to the extreme computation time to do so.

Each model was run for 50 epochs, chosen due to the computational limits of Google Collab. Rather than run a few models for a longer training time which would have increased model performance, we opted to train many models for a shorter time so as to get a broad sense of the different options for optimal model design, which were also largely generalizable to longer training times.

3.3 Combination strategies

We ran two strategies that combined several of the previously modeled methods, concatenating their binarized versions as before: 1) k-hops, cycles and subgraphs, which together specify 232 of the 256 feature dimensions, and after which we used zeros for the remaining 24 dimensions, and 2) the same but with Xavier randomization for the remaining 24 dimensions.

3.4 Pre-Training

In addition to the previous two strategies, we investigated potential implications of pre-training. In Hu et al. [2020], the authors used pre-training on large-scale graphs, splitting the graph by nodes into pre-train, train, validation and test sets, with pre-training being significantly larger than the other splits. While the train, validation, and test sets were split for this graph based on drugs of different functions, we attempted to split our pre-training and training set based on an arbitrarily assigned node ID. Despite our best attempts, we were unable to fully implement this method due to the limitations of the Google Collab environment.

4 Experiments

Due to the nature of the current extenuating circumstances, we were unable to run all models that we generated to completion. Specifically, all these models took significant time to run, on the order of 6 hours per model, without even doing hyper-parameter tuning. As such, we opted to run each model only to 50 epochs and compare then. For consistency, all models were run with the dropout of 0.5, with 5 evaluation steps, 256 hidden channels, 1 log step, learning rate of 0.005, and 10 runs.

For our tests, the main model we decided to use is a GCN from Kipf and Welling [2017]. For our dataset, we used the drug-drug interaction dataset from the open graph benchmark (ogbl-ddi). The main metric we used was Hits@K, in particular, Hits@20, since it is used for the leaderboard in the Open Graph Benchmark.

Results as below are reported after training for 50 epochs for Hits@K with $K = \{10, 20, 30\}$.

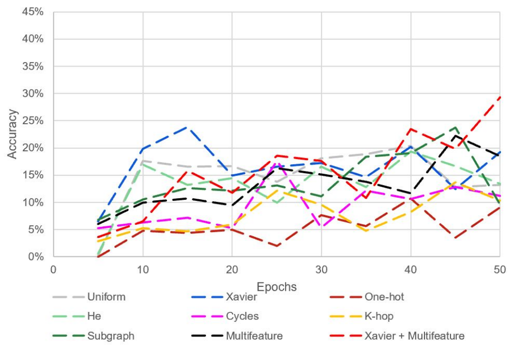
	Hits@10		Hits@20		Hits@30	
	Validation	Test	Validation	Test	Validation	Test
Embedding experiments						
Baseline (Uniform)	15.27	13.23	18.46	12.81	24.02	19.85
Xavier	21.82	**19.20**	18.26	26.18	31.26	23.89
He	15.42	13.42	24.13	14.38	27.32	19.48
Feature experiments						
One-hot	10.27	9.05	11.35	13.26	13.71	15.05
K-hop	10.36	13.02	14.77	15.52	17.19	17.95
Cycles	11.20	11.66	15.44	15.08	18.95	17.57
Subgraphs	9.53	14.00	17.64	17.50	19.91	21.03
K-hop, Cycles, Subgraphs	18.43	12.00	24.22	20.96	26.62	24.54
K-hop, Cycles, Subgraphs + Xavier	**29.25**	13.35	**32.12**	**28.52**	**34.17**	**31.71**

Table 1: Accuracy after 50 epochs for various models

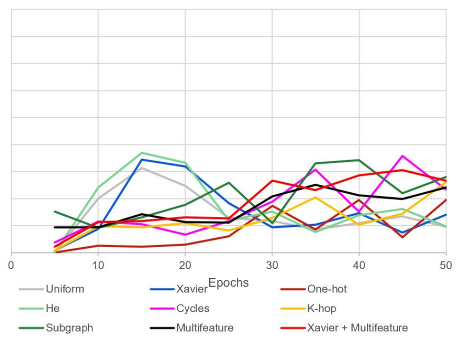
Based on our experiments, all initialization methods perform better than the baseline for the test set for Hits@20. However, for the validation set, it appears that only He, the multifeature combination (k-hops, cycles and subgraphs) and the multifeature combination with Xavier, perform better than the baseline. One explanation for this is that based on the way the ogbl-ddi dataset splits the train, validation, and test sets, there are different types of drugs in each section with some harder or easier to generalize to, so we should be looking for models that generalize the best to drugs of all types.

In general, it appears that Xavier and He are better embedding initializations than the uniform random initialization, with Xavier performing better. This is the same result as Kocmi and Bojar [2017] found for word embedding initialization.

In terms of feature experiments, one-hot, k-hop and cycles all performed worse than the baseline for the validation set, but higher than the baseline for the test set. However, this is likely due to the fact that we ran these experiments for so few epochs. In epoch 15 of the test set, we see that the baseline

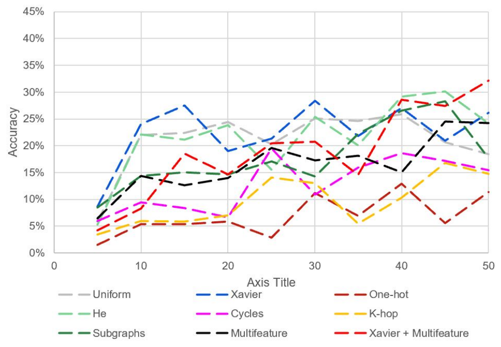


(a) Validation

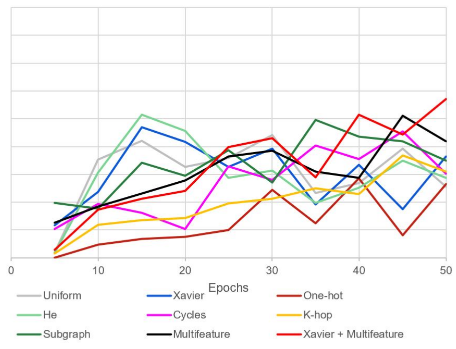


(b) Test

Figure 3: Hits@10

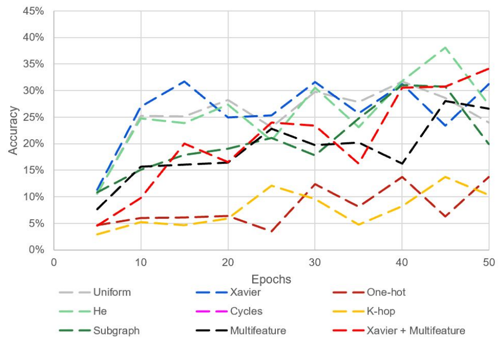


(a) Validation

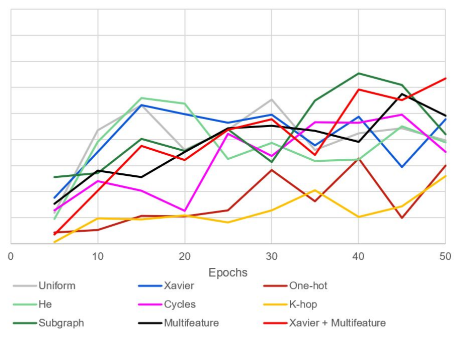


(b) Test

Figure 4: Hits@20



(a) Validation



(b) Test

Figure 5: Hits@30

does have a much higher accuracy, suggesting that it is possible for the baseline to perform better than these models in a longer experiment. One explanation for why these feature experiments performed worse could be because of how sparse they are. Based on our methodology of binarization, many of the dimensions in the feature vectors were 0. This was the case even for cycles, which had the largest number of non-zero dimensions, but only filled up 134 of the 256 dimensions in the hidden layer. With about half of the features being 0 for all nodes, it is not surprising that these models performed worse than the baseline, which had some form of random noise in all values.

Our combination model of k-hop, cycles and subgraphs solves most of this problem, as 232 dimensions were filled. As such, it performed better than the baseline for Hits@20 and almost all other Hits@ values that were being measured. However, we still had 24 dimensions that were zeros. Our final combination model fills in these values with statically-initialized Xavier values. As shown above, this model consistently performs better across almost all tests as compared to the baseline and other models.

While these experiments were only run for 50 epochs, we see that the 'all' combination model performs generally better on epoch 25 and beyond, suggesting that this model will be robust for the 200 epoch-standard that is used in Kipf and Welling [2017].

5 Conclusion

Initialization of parameters can make or break a model, especially for graphs with relatively little information. In this paper, we presented different embedding and feature initialization strategies. In particular, a combination of binarized k-hop, cycle, and subgraph features with static Xavier initialization performed significantly better than the baseline uniform initialization. We believe that this form of augmenting information to graphs with little information could significantly improve all existing models. Further work in more complex augmentation of information, such as pre-training, could yield breakthroughs in link prediction tasks.

References

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Journal of Machine Learning Research*, 2010.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015. ISBN 9781467383912. doi: 10.1109/ICCV.2015.123.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, Kai Wei Chang, and Yizhou Sun. GPT-GNN: Generative Pre-Training of Graph Neural Networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020. ISBN 9781450379984. doi: 10.1145/3394486.3403237.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017.

Tom Kocmi and Ondřej Bojar. An exploration of word embedding initialization in deep-learning tasks, 2017. ISSN 23318422.

Wenqing Lin, Feng He, Faqiang Zhang, Xu Cheng, and Hongyun Cai. Initialization for network embedding: A graph partition approach. In *WSDM 2020 - Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020. ISBN 9781450368223. doi: 10.1145/3336191.3371781.

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. Random features strengthen graph neural networks, 2020. ISSN 23318422.

Keyulu Xu, Stefanie Jegelka, Weihua Hu, and Jure Leskovec. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019*, 2019.