Global and Local Context-aware Graph Convolution Networks

Yichen Yang
Department of Statistics
Stanford University
yyang9@stanford.edu

Lingjue Xie
Department of Statistics
Stanford University
ljxie@stanford.edu

Fangchen Li
Department of Statistics
Stanford University
fcli1997@stanford.edu

1 Introduction

1.1 Dataset: ogbl-ddi

ogbl-ddi is a drug-drug interaction network [1] provided by OPEN GRAPH BENCHMARK (OGB)[2], a set of benchmark datasets covering a variety of real-world applications and several important domains. ogbl-ddi is a homogeneous, unweighted, undirected graph in which each node represents FDA-approved or experimental drug, while each edge represents interactions between the two linked drugs. Drug-drug interactions occur when two or more drugs react with each other, which may make the drug less effective, cause unexpected side effects, or increase the effect of a particular drug. The study on drug-drug interactions is an important phase in drug development and has a significant effect on people's health. In addition, the protein-specific data split provided by OGB brings unique out-of-distribution prediction challenges. ogbl-ddi has 4,267 nodes with 1,334,889 edges. The average node degree is about 500 and the average cluster coefficient is about 0.5. Compared to graphs in social networks, knowledge graphs or recommender systems, it is a much denser graph, suggesting that techniques specialized for sparse link prediction may not applicable for this dataset.

1.2 Problem settings

Our task is to predict drug-drug interactions given information on already known joint effects between drugs using state-of-the-art graph neural networks. For ogbl-ddi, OGB also offers a protein-specific data split, meaning that the drug edges are split according to what proteins those drugs target in the body. The drugs in the test set target on different proteins from the drugs in the train set, which allows us to evaluate the capabilities of our model to generate practically useful predictions on unknown medications. Technically speaking, the links in the test set and the ones in the train set are from different distributions. The out-of-distribution predictions increase the task difficulties.

The metric we are going to use for evaluating our algorithm is Hits$@K$, which is calculated by the proportion of positive links that are ranked higher than the K-th negative link. According to the preliminary experiments provided by OGB, K = 20 is an ideal threshold for this dataset.

1.3 Existing method

From the initial benchmarking results provided by OGB[2], among GCN, GraphSAGE, Node2vec and matrix factorization approaches, only GNN models are able to achieve good performance on the test set, suggesting the relational information is essential for the model to generalize to new interactions in our dataset. Thus, we first implement a GCN model as our baseline model. However, it has been widely recognized that the power of many existing GNNs is upper-bounded by the 1-Weisfeiler-Lehman(WL) test[3]. We focus on Distance encoding(DE) proposed by Li et al.[4], which is one of the methods trying to supersede such upper bound of power. It intends to use graph distance measures such as shortest path distance(SPD) between the node set whose representation to

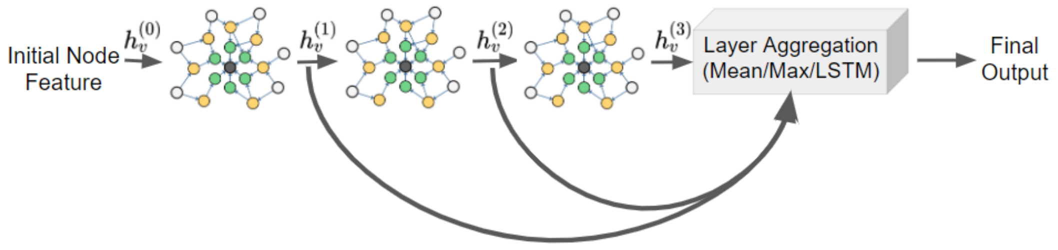

Figure 1: Illustration of our model's framework. The initial node features include trainable node embedding, node statistics and distance encoding. In the intermediate steps, multi-hop aggregation is performed(up to 2-hop in the graph). The center node(in dark grey) aggregates message from itself, 1-hop neighbors(in green), and 2-hop neighbors(in yellow), but not from further nodes(in white). The output of each hidden layer will be further aggregated at the end of all hidden layers with a chosen method of aggregation (mean pooling, max pooling, LSTM attention, etc.)

be learned and each node in the graph, as extra node features. DE can be further extended to control the aggregation step of GNNs.

In our implementation, we combine DE with GCN architecture in two effective ways suggested in the paper: (1)DE-GNN: use DE as extra node features, and (2)DEA-GNN: use DEs to control the process of message aggregations on top of DE-GNN. For the choice of distance measure, we followed the suggestions from the authors, and tried two distance measurements: shortest-path-distance (SPD) and landing probability(LP).

1.4 Our method

As our dataset does not have features, besides the features generated by DE, we also include four node statistics as additional node features: node degree, page rank, clustering coefficient and closeness centrality.

Inspired by [5], we add batch normalization after GCN layers because empirical studies show that adding batch normalization helps the back propagation process and generally improves GNN performance.

Then as our graph has diverse subgraph structures, we implement JKNet framework[6], which enables the model to learn adaptive, structure-aware representations. JKNet learns to selectively exploit information from neighborhoods of the differing locality by two architectural changes - jump connections and a following adaptive aggregation mechanism, which we will discuss in detail in section 3.2. Combining the JKNet framework with Distance Encoding (Figure 1) gives a boost on the model performance, resulting in our best model's test Hits@20 exceeds the best test Hits@20 on the leaderboard by 9.68% and reduces the standard deviation by 0.0164.

2 Existing Method

Recently, Graph neural networks(GNNs) have become the most prevalently used method to obtain the learned representation of the underlying graph structure. GNNs apply message passing and aggregation iteratively layer-by-layer from an initialized set of node features, and readout the embedding from a final readout function. However, it has been widely recognized that the power of many existing GNNs is upper-bounded by the 1-Weisfeiler-Lehman(WL) test[3]. Efforts have been made to surmount such upper bound. Some proposed frameworks, including PPGN[7], focus to surpass the bound by focusing on the representation of the entire graph.

We focus on Distance encoding(DE) proposed by Li et al.[4], a provably more powerful and expressive GNN structure. It is one of the methods trying to supersede such upper bound of power by using graph distance measures to augment node features, and can be further extended to control the aggregation step of GNNs. **The DE method is not on the current leaderboard.** A rigorous mathematical definition of Distance Encoding goes as the following: Let $G = (V, E)$ be a given undirected graph with information stored in an adjacency matrix A. Consider a set of nodes $S \subseteq V$, a $k-$distance encoding is defined as a function

$$\zeta(\cdot|S, A) : V \to \mathbb{R}^k$$

, where the choices of ζ are those from the set of permutation invariant functions. In the original literature, for a given node $u \in V$, the authors choose DE as a concatenation of $\zeta(u|v)$ with $v \in S$; that is,

$$\zeta(u|S) = \text{Concat}(\{\zeta(u|v)|v \in S\})$$

The authors first use DE $\zeta(u|S)$ as extra node features, which is termed as DE-GNN. The authors further proposed DE-Aggregation-GNN (DEA-GNN) based on DE-GNN. With the structural features generated by DE-GNN, the DEA-GNN framework also uses them to control the aggregation step. Specifically, the aggregation step is modified by

$$\text{AGG}(\{f(h_v^{(l)})\}_{v \in \mathcal{N}_u}) \to \text{AGG}(\{f(h_v^{(l)})|\zeta(v|u) \in \mathcal{C}\}_{v \in V})$$

, where $\mathcal{C}$ is some criteria associated with the function ζ.

We consider Distance Encoding as the ideal choice for our dataset for various reasons. First, the ogbl-ddi dataset does not have any existing node features associated with nodes; therefore, Distance Encoding can be used alongside node embeddings to generate useful structural features that can be utilized in the subsequent GNNs. Second, ogbl-ddi forms an extremely dense network, so using Distance Encoding would be greatly helpful to deal with problems like over-smoothing by extending the local aggregation to further neighbors.

2.1 DE-GNN—Distance Encodings as Node Features

In the first application of Distance Encoding termed as DE-GNN, where given a set of nodes S, DE-GNN computes $\zeta(u|S)$ and concatenates it to the original feature of node u. Thus, the initial input for node u can be written as $h_u^{(0)} \oplus \zeta(u|S)$. In our case, as the original dataset does not contain any node feature, we generate node embeddings as our $h_u^{(0)}$. We implemented two choices for the function ζ: Shortest-path-distance (SPD) and landing probability(LP).

- **DE-GNN-SPD**
  In DE-GNN-SPD, for $u \in V$ and $v \in S$, $\zeta(u|v)$ is a one-hot vector of the truncated shortest-path-distance between u and v. We can denote it as

  $$\zeta_{SPD}(u|v) = \text{one hot}(\min(d_{SPD}(u, v), d_{max}))$$

  , where d_{max} is a chosen constant as the maximum distance that can be encoded. d_{max} can help prevent overfitting the noise in an overly large neighborhood.
- **DE-GNN-LP**
  This variant sets $\zeta(u|v)$, $u \in V$ and $v \in S$ as landing probabilities of random walks with various number of steps starting from v to u:

  $$\zeta_{lp}(u|v) = ((W^{(0)})_{vu}, (W^{(1)})_{vu}, (W^{(2)})_{vu}, ..., (W^{(d_{rw})})_{vu})$$

  where $W^{(k)} = (AD^{-1})^k$ is the $k-$step random walk matrix. d_{rw} is a chosen constant as the max number of steps of random walks. The information contained in $\zeta_{lp}(u|v)$ encodes the distance information that is stronger than $\zeta_{SPD}(u|S)$, sincec $d_{SPD}(u, v)$ can be obtained from $\zeta_{lp}(u|v)$ by computing the number of steps required to get non-zero random walk landing probability for the first time.

In both two distance measures, we finally compute

$$\zeta(u|S) = \frac{1}{|S|} \sum_{v \in S} \zeta(u|v)$$

2.2 DEA-GNN—Distance Encodings as Controllers of the Message Aggregation

As previously introduced, DEA-GNN applies the augmented features for controlling the aggregation step. DEA-GNN allows message aggregations to go beyond the traditional $1-$hop procedure and aggregate messages from $K-$hop neighbors or unusual neighborhood structures. One of the choices which the authors proposed is Shortest Path Distance(SPD). A formal exposition of the framework follows:

- **DEA-GNN-SPD**
  For a given node u, DEA-GNN-SPD aggregates the information as indicated below:
  $$\text{AGG}(\{f(h_v^{(l)})\}_{v\in\mathcal{N}_u}) \rightarrow \text{AGG}(\{f(h_v^{(l)})|d_{SPD}(v,u) \leq K\}_{v\in V})$$
  , which essentially achieves a $K-$hop neighbor aggregation.
  Thus, in practice, the aggregation step for the layer l can be written as
  $$h_u^{(l+1)} = \sum_{k=1}^{K} \text{Relu}(\frac{1}{|S_{u,k}|+1}(h_u^{(l)} + \sum_{v\in S_{u,k}} h_v^{(l)}\Theta^{lk})), S_{v,k} = \{v|d_{SPD}(v,u) = k\}.$$
  where $\Theta^{(lk)}$ is a weight matrix which would be updated for each k.

3 Method

3.1 Node Statistics as Node Feature

Our final input features have three components: node embeddings, node statistics, and DE. For node statistics, we use four measures: node degree, page rank, clustering coefficient and closeness centrality, as well as node embeddings. For DE, we test the performance of both SPD and LP.

3.2 Incorporating Jumping knowledge networks[6]

During our implementation of DE encoding, we realized that a limitation of its neighborhood aggregation scheme is that it has a fixed but structure-dependent influence radius size which might not be the best representation. For example, 5 hops from a node within the core include almost the entire graph. On the contrary, 5 hops starting at a marginal node might include only a very small fraction of all nodes. The biological networks that we are working with possess locally strongly changing structure; the majority of the nodes have many connections, whereas some nodes are connected to only a few other nodes. If we use a fixed radius size for information aggregation for each node, then for the central node, the aggregation is too broad and therefore losses information. In contrast, the marginal nodes do not aggregate information from sufficient neighborhoods, which might cause predictions to be unstable.

Thus, to further improve our model performance, we adopt the idea from jumping knowledge (JK) networks, which flexibly aggregates, for each node, information from different neighborhood ranges to train better structure-aware representations. Figure 1 illustrates the main idea of how we incorporate the JK network. Instead of pass the information layer by layer, the JK network allows information from the middle layers to "jump" to the last layer. For each node, it is able to gain information through such architecture independently, then the network can learn the effective neighborhood radius for each node as needed. Besides the jump connections, JK networks also propose a subsequent selective but adaptive aggregation mechanism. Let $h_v^{(1)}, \ldots, h_v^{(k)}$ be the jumping representations of node v as shown in figure. In the paper, the authors propose three layer-aggregation mechanisms: concatenation, max-pooling and lstm-attention. We implement all three proposed approaches and also explore one additional possibility, mean aggregation.

- **Concatenation**
  The concatenation aggregation is realized by concatenate the k different jumping representation; ie $[h_v^{(1)}, \ldots, h_v^{(k)}]$, after which, the JKNet performs a linear transformation to get $h_v^{(final)}$. The weights in linear transformation are optimized with subgraph features into account so that it works best for the entire dataset. Through our implementation, we found that the weight-sharing in concatenation makes this aggregation mechanism not flexible enough to learn the varying subgraph structure in our dataset.

- **Max-pooling**
  For each feature coordinate, the model selects the most informative element; i.e. $h_v^{(\text{final})}$ = max $(h_v^{(1)}, \ldots, h_v^{(k)})$. Max-pooling is adaptive by nature and it does not require any additional parameters to train.
- **LSTM-attention**
  For the LSTM attention, we first input $h_v^{(1)}, \ldots, h_v^{(k)}$ into a bi-directional LSTM[8], which outputs hidden features $f_v^{(l)}$ from forward-LSTM and $b_v^{(l)}$ from backward-LSTM. Then we apply a linear transformation on the concatenated hidden states $[f_v^{(l)}, b_v^{(l)}]$ to produce a scalar value s_v^l. Subsequently, we applied a softmax layer to $\{s_v^l\}_{l=1}^k$, which generates the attention of node v to the features learned in previous layers. Finally, the $h_v^{(\text{final})}$ is calculated as $\sum_l \text{softmax}(\{s_v^{(l)}\}_{l=1}^k)[f_v^{(l)}, b_v^{(l)}]$. The attention mechanism effectively identify the most important neighborhood range for each node via the assignment of attention score to different layers
- **Mean**
  Similar to max pooling, but now we replace the max operation with mean for each feature coordinate, i.e. $h_v^{(final)}$ = mean $(h_v^{(1)}, \ldots, h_v^{(k)})$. The model averages all the information from previous layers. Mean is adaptive by nature and it does not introduce any additional parameters to train.

4 Experiments

Extensive experiments have been conducted on the baseline, the existing model of our concern, and different variants of our proposed framework that incorporates both JKNet and DEA-GNN.

4.1 Dataset and Training

We conduct the link prediction task on the ogbl-ddi dataset. The dataset is split into train, validation, and test dataset according to the ratio 80/10/10. All the models are trained using Adam[9] with a learning rate 0.005 until loss converges or reaching a maximum of 400 epochs. We use a batch size of 1024×64 and the hidden dimension size is set to be 256 for all settings. All these hyperparameters choices except the number of epochs follow from OGB baseline methods (GCN, GraphSage, etc.) The testing performance of the best model on the validation set is reported. For all following experiment settings, we repeat each experiment 5 times with different random seeds.

4.2 Baseline

We choose GCN as our fundamental baseline model. The baseline framework has 2 GCN layers. We take the element-wise product of the final embeddings of the two nodes to represent the edge between them. Then the element-wise product is fed into a 2-layer perceptron to generate the final score. Batch normalization, Relu nonlinear activation function and dropout with probability 0.5 are applied to the output of each hidden layer. Since there are no existing node features, we use trainable node embeddings as the node features fed into the GCN framework.

4.3 Distance-Encoding

We stick with GCN as the base of our proposed framework and apply variants of DE-GNN over it. We use shortest path distance(SPD) and landing probability(LP), whose detailed implementations are mentioned in section 2.1, separately as extra initial features along with the embedding matrix. To better compare the performance with the baseline model, we also implement common choices for feature augmentation, including centrality, Page Rank, degree, and clustering coefficients. We further construct a model that incorporates all previously mentioned features.

For DEA-GNN, we implement the variant, DEA-GNN-SPD, upon the basic framework GCN. As previously introduced, this type of model enables us to perform multi-hop aggregation. We experiment

Method	Validation Hits@20	Test Hits@20
NODE2VEC	0.3292$\pm$ 0.0121	0.2326 $\pm$ 0.0209
GCN	0.5550 $\pm$ 0.0208	0.3707 $\pm$ 0.0507
GRAPHSAGE	0.6262 $\pm$ 0.0037	0.5390 $\pm$ 0.0474
MATRIXFACTORIZATION	0.3370 $\pm$ 0.0264	0.1368 $\pm$ 0.4750

(a) OGB baseline models[2]

Method	Validation Hits@20	Test Hits@20
GCN	0.6680 $\pm$ 0.0085	0.3895 $\pm$ 0.1360
GCN + Augmentation	0.6716 $\pm$ 0.0067	0.4323 $\pm$ 0.1553
GCN + SPD	**0.6734** $\pm$ **0.0038**	0.4669 $\pm$ 0.1183
GCN + LP	0.6725 $\pm$ 0.0022	**0.5261** $\pm$ **0.1019**
GCN + SPD + LP + Augmentation	0.6720 $\pm$ 0.0046	0.4634 $\pm$ 0.0846

(b) Comparison of baseline, common feature augmentation, and DE-GNN features

Method	Validation Hits@20	Test Hits@20
DEA-GNN-SPD, K=2	**0.7128** $\pm$ **0.0032**	**0.6242** $\pm$ **0.0708**
DEA-GNN-SPD, K=3	0.7105 $\pm$ 0.0034	0.6231 $\pm$ 0.1101

(c) Comparison of different ranges of aggregation in DEA-GNN-SPD ($K = 2, 3$)

Method	Validation Hits@20	Test Hits@20
DEA-JK-Mean (2 layers)	0.6722 $\pm$ 0.0086	-
DEA-JK-Mean (3 layers)	**0.6885** $\pm$ **0.0048**	0.7133 $\pm$ 0.1519
DEA-JK-Mean (4 layers)	0.6813 $\pm$ 0.0050	-
DEA-JK-LSTM (2 layers)	0.6676 $\pm$ 0.0073	-
DEA-JK-LSTM (3 layers)	**0.6900** $\pm$ **0.0039**	0.7662 $\pm$ 0.0681
DEA-JK-LSTM (4 layers)	0.6751 $\pm$ 0.0075	-
DEA-JK-Max (2 layers)	0.6421 $\pm$ 0.0113	-
DEA-JK-Max (3 layers)	**0.6770** $\pm$ **0.0037**	**0.7749** $\pm$ **0.0130**
DEA-JK-Max (4 layers)	0.6342 $\pm$ 0.0151	-

(d) Comparison of performance of DEA-JK with various aggregation methods and different numbers of DEA-GNN-SPD layers

Table 1: Hits@20 results of different models

with $K = 2, 3$, which is the max distance for aggregation, and the one with better performance would be chosen in the subsequent experiments.

4.4 JKNet

On top of DEA-GNN-SPD, we incorporate JKNet with fours different aggregation methods across layers including concatenation, mean pooling, max pooling and LSTM attention. For the concatenation aggregation, we find that the weight-sharing in this mechanism is not flexible enough to learn the varying subgraph structures. Thus, it is not included in our final experiments shown in 1d. Based on the result from DEA-GNN-SPD, we fix $K = 2$, which allows up to 2-hop aggregation. We compare the performance of models with $2, 3$, and 4 DEA-GNN-SPD layers in the DEA-JK setting.

4.5 Result and Discussion

In Table 1b, our baseline GCN model performs slightly better than the GCN baseline model provided by OGB on the test set, shown in 1a. It can caused by the batch normalization added after each GCN layer, which can be helpful in the back propagation process. In Table 1b, we observe that using different input features results in comparable validation scores; however, their testing performances differ a lot. A credible explanation is that, because the data splits provided by OGB is protein-specific, the edges are not identically distributed among test set and training/validation sets. Therefore, the performance on the validation set might not be quite useful in evaluating the model's performance on the test set. This also explains why the variance of test hits is large. The results also suggest that using extra untrainable features, such as node statistics, SPD, and LP, can reduce overfitting and generalize the model to unseen structures. In the following experiments, we choose to add all of node statistics, SPD, and LP as the input features.

In Table 1c, we notice that DEA-GNN-SPD significantly improves upon the best method in Table 1b. The results of 2-hop and 3-hop DEA-GNN-SPD are similar, probably due to the fact that the ogbl-ddi graph is dense; therefore, 3-hop aggregation does not bring much additional information than the 2-hop aggregation. Since the results are comparable, for simplicity, we choose to work with 2-hop DEA-GNN-SPD in the following experiments.

In Table 1d, for all of the three aggregation methods (Mean pooling, LSTM attention, and Max pooling), we first use the validation set to tune the number of layers, and then report the best model's performance on test set. For all three aggregation methods, the 3-layer structure has the best performance. We find that although the max pooling is not the best model on the validation set, it outperforms mean pooling and LSTM attention by a larger mean test Hits@20 and much smaller variance. The current top performer on the leaderboard MAD learning has Hits@20 of 0.6781 ± 0.0294 on test set. Our best model, the 3-layer DEA-JK-Max model, outperforms it by an increase of 0.0968 in test Hits@20 and a decrease of 0.0164 in standard deviation.

We presume that using max aggregator can be a good method to select an appropriate neighborhood radius for message aggregation; especially in a dense graph like the ogbl-ddi, we may have a problem with oversmoothing or overfitting if the choice of neighborhood radius is too large or too small. Max pooling serves as a simple but powerful technique to allow our model to gather information from multiple layers and extract only useful one from them.

5 Conclusion

In this project, we first compute and compare the performance of using node embeddings, node statistics, and Distance Encoding in the baseline GCN. We then go beyond the traditional 1-hop aggregation structure and move to K-hop aggregation as DEA-GNN-SPD. Finally, we incorporate JK networks on the top of the DEA structure. Although DEA alone or JKNet alone does not perform well, our innovative combination effectively improves the learned representation of the nodes in the graph. On the one hand, DEA aggregates multi-hop information at each layer, so it can capture global information within just a few layers; on the other hand, JKNet with max pooling can select useful information from each layer, so the model can learn adaptive representations that are aware of both local and global information. Results show that our final model outperforms the top model on the leaderboard by approximately 10% in mean test Hits@20 on the link prediction task on the ogbl-ddi graph.

In the future, we can test our model performance on different tasks. For example, we could conduct experiments on larger datasets if computation resource permits us to test our model's generalizability. We can also continue to work on ogbl-ddi dataset and improve our model's performance by finding out the best design choice, such as hidden dimensions, activation functions and learning rate.

References

[1] David S Wishart, Yannick D Feunang, An C Guo, Elvis J Lo, Ana Marcu, Jason R Grant, Tanvir Sajed, Daniel Johnson, Carin Li, Zinat Sayeeda, et al. Drugbank 5.0: a major update to the drugbank database for 2018. *Nucleic acids research*, 46(D1):D1074–D1082, 2018.

[2] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs, 05 2020.

[3] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks?, 2019.

[4] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning, 2020.

[5] Jiaxuan You, Rex Ying, and Jure Leskovec. Design space for graph neural networks. *arXiv preprint arXiv:2011.08843*, 2020.

[6] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks, 2018.

[7] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks, 2020.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.