Edge representation learning with dual hypergraph transformation

Haixiao Wang
Department of Mathematics
University of California, San Diego
h9wang@ucsd.edu

Zhichao Wang
Department of Mathematics
University of California, San Diego
zhw036@ucsd.edu

Abstract

In this project, we aim to study how to utilize information from edges of the graphs in representation learning. We study the methods (Edge HyperGraph Neural Network) established in [1] and apply their methods in Graph Neural Network (GNN). This project focus on learning accurate edge representations for tasks such as node and edge classification where the importance of discriminative edge representations is often being neglected in most previous research. Following the idea of [1–3], we applied novel edge representation learning framework based on the Dual Hypergraph Transformation (DHT) introduced in [1]. We conducted extensive experiments on benchmarking dataset to validate the effectiveness of the methods in [1] on several important graph-based tasks. We found out applying GraphSage on the dual hypergraph is the best way for edge-level tasks. Also, we examined the efficiency and complexity of DHT for edge representation learning.

1 Introduction

Graphs are widely used for many complex systems of interacting objects, such as social networks, knowledge graphs, molecular graphs, biological networks, as well as for modeling 3D objects [4, 5]. Recently, significant methodological advances have been made, particularly GNNs, which have plenty of applications from diverse domains [5–9]. While GNNs recently have achieved impressive success on graph-related tasks, most of them only focus on learning node-level representations, with less focus on the edges. For instance, in both graph reconstruction and generation, accurately representing the edges of a graph is critical, as incorrectly reconstructing/generating edges may result in complete failure of the tasks [1]. Thus, we would like to study the edge-level approaches and obtain explicit representations for edges in GNNs by considering the efficiency and accuracy for these new edge representations in GNNs.

On the other hand, hypergraph has been employed in many computer vision tasks such as classification and retrieval tasks. Traditional hypergraph learning methods suffer from their high computation complexity and storage cost, which limits the wide application of hypergraph learning methods. Hence, hypergraph neural network itself is the second motivation of this project. In this problem, we will further analyze how to efficiently apply pooling techniques in hypergraph neural networks framework [10] after applying the DHT.

2 Related works

Edge-aware GNNs Graph representation learning has received great attention in recently since most real-world data in applications have graph structures [11]. While node-level representations achieve impressive success on graph-related tasks, especially in Graph neural networks (GNNs) [4–6, 8, 9], the edge-level representations are also important for capturing the graph features. There

Preprint. Under review.

are several works studied edge features while updating the node features. Edge-aware GNNs have been studied by [12, 7, 4, 13]. In particular, [14] provides an edge-level explanations for GNNs, further emphasizing the importance of edge-aware GNNs. In this project, we aim to develop GNNs which employ the edge representations. In fact, many advanced GNNs have considered edge features while updating the node features [15, 13]. Some of these edge-aware GNNs even explicitly represent edges by introducing edge-level GNN layers to improve node features [3, 16, 2, 17]. However, this project follows a new idea in [1] by introducing a new transformation of the graph to learn edge representations in a more explicit way.

Graph transformation We follow [1] using a graph transformation to recapture the edge-level representations. Graph transformations also appear in [18, 19], where they propose to transform the original graph into a typical graph structure, to apply graph convolution for learning the edge features. Besides, [20] also considered hypergraph duality for graph transformation. However, most of these transformations, comparing with [1], are not injective, not scalable, even loosing node information in the original graph during the transformation. Practically, graph transformations have more complexity in real-world graph data, so it is necessary to analyze complexity of transformations theoretically and empirically. A complete complexity analysis of transformation and message passing operations of existing edge-aware GNNs has been presented by [3, 2, 21, 1]. Moreover, [1] further experimentally verifies the transformation complexity on Erdos-Renyi graphs.

Graph pooling Graph pooling methods aim to learn accurate graph-level representation by compressing a graph into a smaller graph or a vector with pooling operations. This is crucial in practical implementations. The usual pooling approaches are using mean, sum or max over all node representations. We refer [22, 23, 17, 24, 25] for more details in graph poolings. In the appendix of [1], the authors provide a full descriptions for different methods of graph poolings. The new methods proposed in [1] are inspired by [26, 23–25]. Specifically, [26] combines both node pruning and clustering approaches, by dropping meaningless clusters after grouping nodes, while [23] proposes to use attention-based operations for considering relationships between clusters. However, there is no graph poolings for edge-level representations. We aim to follow the idea in [1] to study the edge-level graph pooling.

3 Main Techniques

In this section, we will first recap the new techniques invented by [1]: Dual Hypergraph Transformation for edge representation learning, and new graph pooling methods. This edge representation learning and pooling method largely outperforms state-of-the-art graph pooling methods on graph classification, not only because of its accurate edge representation learning, but also due to its lossless compression of the nodes and removal of irrelevant edges for effective message-passing. We will first introduce these techniques in details and implement them in experiments and further try to improve these advanced new methods.

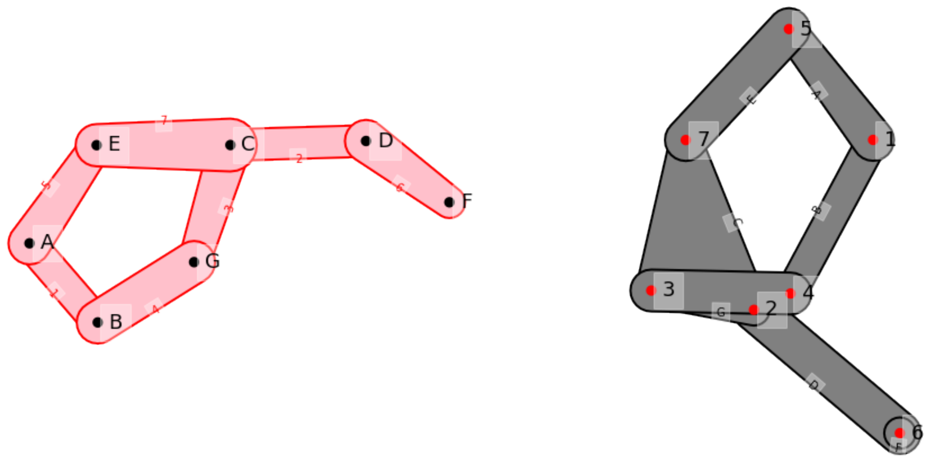
Given a graph G with m edges and n nodes. Define its node features by $X \in \mathbb{R}^{n \times d_1}$ and edge features by $E \in \mathbb{R}^{m \times d_2}$. Let $A \in \mathbb{R}^{n \times n}$ be its adjacency matrix. We recall that a graph neural network is defined as follows [16]:

$$X_v^{(l+1)} = \text{UPDATE}\left(X_v^{(l)}, \text{Aggregate}\left(X_u^{(l)} : \forall u \in \mathcal{N}(v; A)\right)\right), \tag{1}$$

where $X_v^{(l)}$ is the feature for node v at l-th layer, Aggregate is the function that aggregates messages from a set of neighboring nodes of the node v, UPDATE is the function that updates the representation of the node v from the aggregated messages, and $\mathcal{N}(v; A)$ is the set of neighboring nodes for the node v, obtained from the adjacency matrix A.

3.1 Edge representation learning: Dual Hypergraph Transformation

For a given graph $G = (V, E)$, the authors in [27] constructed *line graph* $L(G) = (V_L, E_L)$ according to the non-backtracking matrix $B \in \mathbb{R}^{2|E| \times 2|E|}$, where $V_L = \{(i \to j) | (i, j) \in E\}$, $|V_L| = 2|E|$ and $B_{(i,j),(i' \to j')} = 1$ if $j = i'$ and $j' \neq i$, or 0 otherwise, to encode the directed edge adjacency structure of G. Even though the edge representations was constructed explicitly, it is only used as an augmentation of node information, while the complexity for message passing is increased to $O(|E|^2)$,



(a) Original graph (b) Dual Hypergraph

Figure 1: An illustration of the proposed edge representation learning framework with dual hypergraph transformation (DHT) in [1]. We transform edges (red numbers) in original graph to nodes (red dots) in the dual hypergraph.

which is at least $O(|V|^2)$ for sparse graph and $O(|V|^4)$ for dense graph. However, the complexity for message passing was $O(|V|)$ for sparse graph and $O(|V|^2)$ for dense in original graph.

In [1], the authors presented a novel way construct edge representation by applying *Dual Hypergraph Transformation*. Specifically, we can use a triple $G = (V, M, E)$, where matrix $M \in \{0, 1\}^{|V| \times |E|}$ is the incidence matrix defined by $M_{ie} = \mathbf{1}_{\{i \in e\}}$ for $i \in V$ and $e \in E$, to incorporate the connectivity information of G. The *dual hypergraph transformation* is obtained by interchanging the structural role of nodes and edges, and transposing the incidence matrix, i.e., $G^* = (E, M^T, V)$. The DHT interchanges node and edge features across G and G^* (see the feature matrices in Figure 1). Mathematically, given a triplet representation of a graph, DHT is defined as the following transformation:

$$\mathbf{DHT} : G = (X, M, E) \longmapsto G^* = (E, M^T, X), \tag{2}$$

where X represents the node features, and E represents the edge features. Inversely, we can easily get

$$\mathbf{DHT} : G^* = (E, M^T, X) \longmapsto G = (X, M, E). \tag{3}$$

Hence, we can recover the original graph from the dual hypergraph with the same DHT operation.

After applying DHT, one can perform the message passing between edges of the input graph, by performing the message-passing between nodes of its dual hypergraph $G^* = (E, M^T, X)$. Comparing with (1), we can obtain edge representation learning via

$$E_e^{(l+1)} = \text{UPDATE}\left(E_e^{(l)}, \text{Aggregate}\left(E_u^{(l)} : \forall u \in \mathcal{N}(e; M^T)\right)\right), \tag{4}$$

where $E_e^{(l)}$ is the node features of G^* at the l-th layer, the AGGREGATE function summarizes the neighboring messages of the node e of the dual hypergraph G^*, and the UPDATE function updates the representation of the node e from the aggregated messages. Similarly, $\mathcal{N}(e; M^T)$ is the neighboring node set of the node e in G^*.

Under DHT, the message passing complexity in each layer is at least $O(|E| \times d_{\max})$ for sparse graph and $O(|E|^2)$ for dense graph, where $d_{\max}$ denotes the maximum degree of the original graph. It still increases the complexity for message passing, thus the *graph pooling* procedure is necessary.

3.2 Graph pooling

The goal of *graph pooling* is to learn an embedding for the entire graph G, by compressing node-level information to graph level. The most common approaches are simply using mean, sum or max over all node representations [9], which is sufficient for fixed size small graph tasks. However, they do not exploit the graph structure.

Another popular approach for graph pooling is to perform dimension reduction on node representations by *clustering* similar nodes based on their embeddings, or *coarsening*, i.e., dropping unimportant nodes. In [1], the authors applied pooling strategies, *HyperCluster* and *HyperDrop*, to cluster similar edges into a single edge, by clustering nodes of the dual hypergraph obtained via DHT. The main drawback of this approach is inevitable loss of node information. HyperCluster is an extension of GMPool studied in [23]. To apply GMPool on a hypergraph, [1] modified the graph multi-head attention block (GMH), which is used to construct key and value matrices using GNNs for the original graph structure in the GMPool paper [23], for the hypergraph structure by replacing the adjacency matrix to the incidence matrix. As for HyperDrop, [1] applied methods in [24, 25] with their score functions to select the top-ranked k nodes in G^* to obtain the pooled edge features.

4 Experiments

In this section, we perform both node and edge classification task on the well-known benchmarking dataset **Cora** (`https://relational.fit.cvut.cz/dataset/CORA`), which contains 2708 scientific publications on machine learning, and the network consists of 5429 directed edges representing citations. Nodes, representing papers, in this citation network are divided into 7 categories. Each node is described by a $0/1$-valued key word vector of length 1433, indicating the absence/presence of the corresponding word from the dictionary, where there are 1433 unique words in this dictionary. Each edge is described by $0/1$-valued scalar, indicating the direction of citation. All the experiments are conducted mainly by using two open-source Python libraries, Pytorch [28] and Pytorch Geometric [29], working with graph datasets. The code is available at `https://github.com/haixiaowang/ECE285_GNN`.

4.1 Node level tasks

The node level challenge is to predict the category of a paper. In practice, given the citation network and the node feature vector(key word vector), we need to assign the paper to one of the seven classes. In this subsection, nodes are split into three disjoint sets *Train*, *Valid* and *Test* with ratio $0.85, 0.05, 0.1$ accordingly, for training and testing process. In the following, we will respectively adopt three useful and advanced GNN architectures, GCN, GAT and GraphSage, to implement this node-level tasks. By virtual these preliminary experiments, we will further review the definitions of these three GNN architectures respectively, which we will employ later in the edge-level tasks.

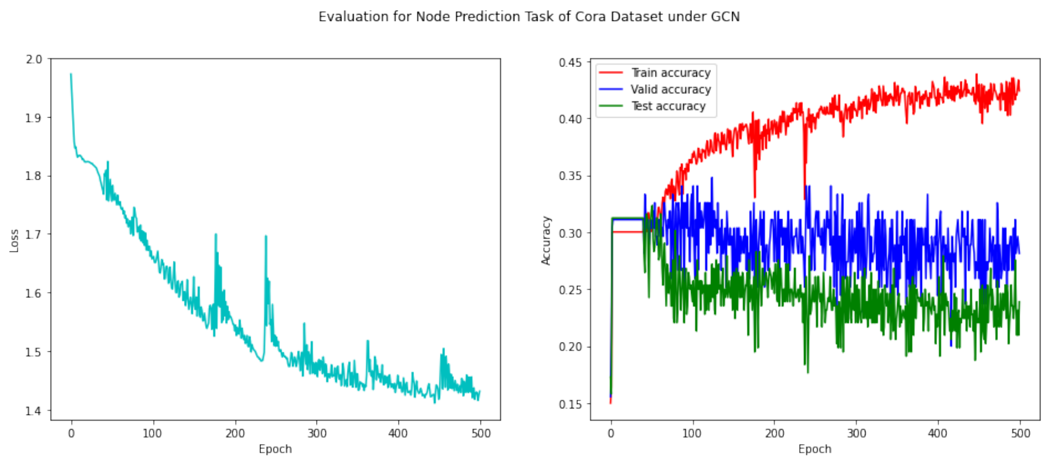


Figure 2: Test accuracy and training loss dynamics on Cora dataset using Node GCN

4.1.1 Vanilla GCN

We first performed node classification task using a 2-layer vanilla GCN (graph convolutional neural network) with hidden dimension being 32. We used Adam optimizer with learning rate 0.01, weight decay $5 * 10^{-3}$ and momentum 0.9. The result can be seen in Figure 2. For more details on GCN, we refer to [11]. In our experiments, the best model achieves the following accuracy, Train: 43.37%, Valid: 30.37% Test: 27.94%, which is far from optimal yet.

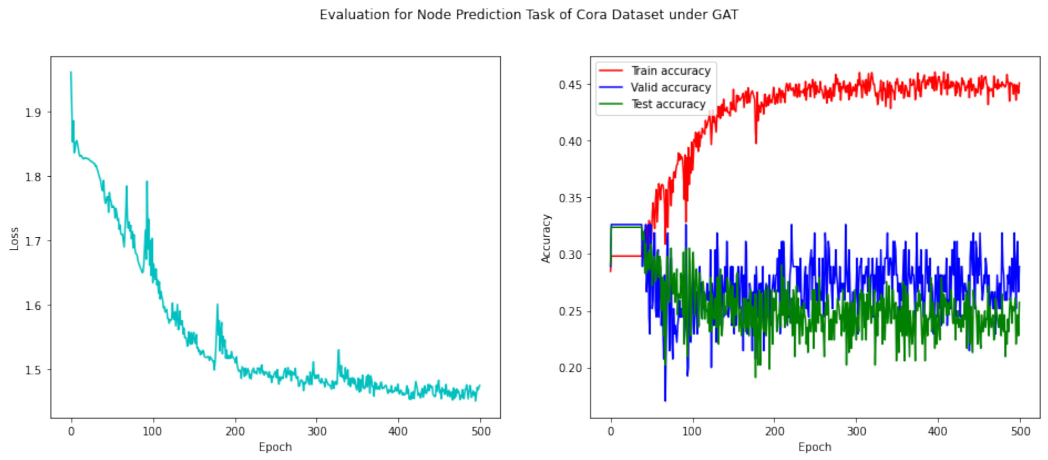


Figure 3: Test accuracy and training loss dynamics on Cora dataset using Node GAT

4.1.2 GAT

To improve the performance of classification task, we applied GAT (Graph Attention Networks) [30], where the graph attention layers, a variant of the aggregation function, were used as the building block. The mechanism of GAT is that, input and output of attentional layers are sets of node features $\{h_v | h_v \in \mathbb{R}^F\}_{v\in V}$ and $\{h'_v | h'_v \in \mathbb{R}^{F'}\}_{v\in V}$ respectively, where F and F' denote corresponding feature vector dimensions. The normalized attention coefficients α_{uv} were used to compute a linear combination of the features according to the equation $h'_v = \sum_{u\in N(v)} \alpha_{uv} \mathbf{W_r} h_v$, where coefficients $\alpha_{uv} = \text{softmax}_j(e_{uv}) = \exp(e_{uv})[\sum_{w\in N(v)} \exp(e_{vw})]^{-1}$ with node feature importance captured by $e_{uv} = a(\mathbf{W}_l h_u, \mathbf{W}_r h_v)$. Here, $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \to \mathbb{R}$ denotes a shared attentional mechanism, where a shared linear transformation parametrized by the weight matrices $\mathbf{W}_l, \mathbf{W}_r \in \mathbb{R}^{F' \times F}$ is applied when u and v are neighbors. To stabilize the learning process of self-attention, we will try multihead attention, that is, we compute M output features by attention layers independently then concatenate these feature representations together by $h'_v = ||_{m=1}^{M} \Big(\sum_{u\in N(v)} \alpha_{uv}^{(m)} \mathbf{W_r}^{(m)} h_v \Big)$, where $||$, $\alpha_{uv}^{(m)}$ denote concatenation and the normalized attention coefficients computed by the m-th attention $(a^{(m)})$ respectively, and $\mathbf{W}_r^{(m)}$ is the corresponding input linear transformation's weight matrix.

In our experiments, we used a 2-layer multihead attention GAT network, with hidden dimension being 64 and $M = 2$. The result can be seen in Figure 3. The best model achieves the following accuracy, Train: 45.81%, Valid: 32.59% Test: 32.35%, which is a bit better than vanilla GCN.

4.1.3 GarphSage

Another solution to improve the performance of the node classification task, is to try GraphSage [31]. While the update rule between layers in vanilla GCN, using element-wise mean-pooling, follows $h_v^{(l+1)} \leftarrow \sigma(W_l \cdot (h_v^{(l)} + \text{MEAN}\{h_u^{(l)}, \forall u \in N(v)\}))$, the update rule in GraphSage uses

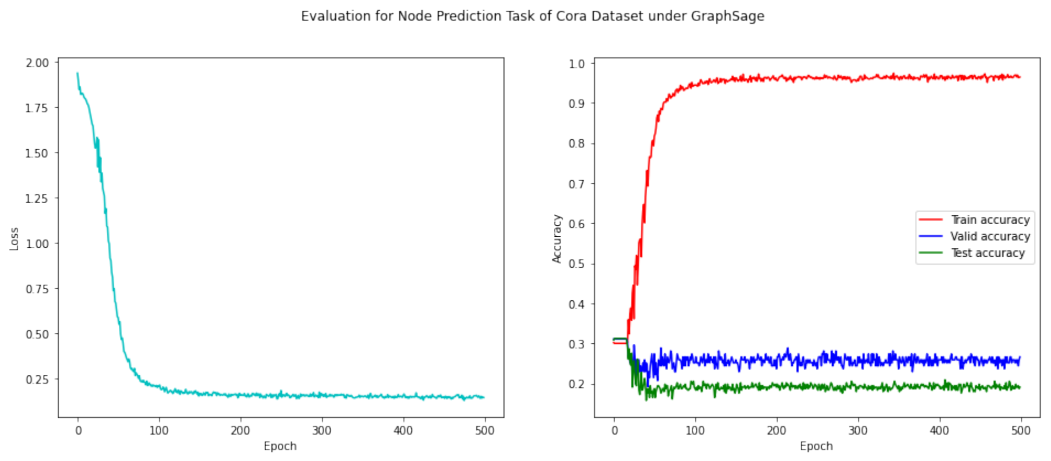


Figure 4: Test accuracy and training loss dynamics on Cora dataset using Node GarphSage

element-wise max-pooling, and follows

$$h_v^{(l+1)} \leftarrow \sigma(W_l \cdot (h_v^{(l)} + \text{MAX}\{h_u^{(l)}, \forall u \in N(v)\}))\,. \tag{5}$$

In our experiments, we used a 2-layer GraphSage network with hidden dimension being 64. The result can be seen in Figure 4. The best model achieves the following accuracy, Train: 96.39%, Valid: 29.63% Test: 31.35%. As we may see, this model doesn't generalize well even though the training accuracy is close to 100%.

4.2 Edge level task

The edge level challenge is to predict the label of an edge, either 0 or 1, given the citation network and the node feature vectors. In this subsection, we train GNN and let h_v denote feature vector of node v. The feature of edge $e = (u, v)$ is then represented by the dot product of node feature vectors, that is, $h_e = h_u^T h_v$. We used cross-entropy loss for this binary classification task. In this subsection, nodes and edges are split into three disjoint sets *Train*, *Valid* and *Test* with ratio $0.85, 0.05, 0.1$ accordingly as well.

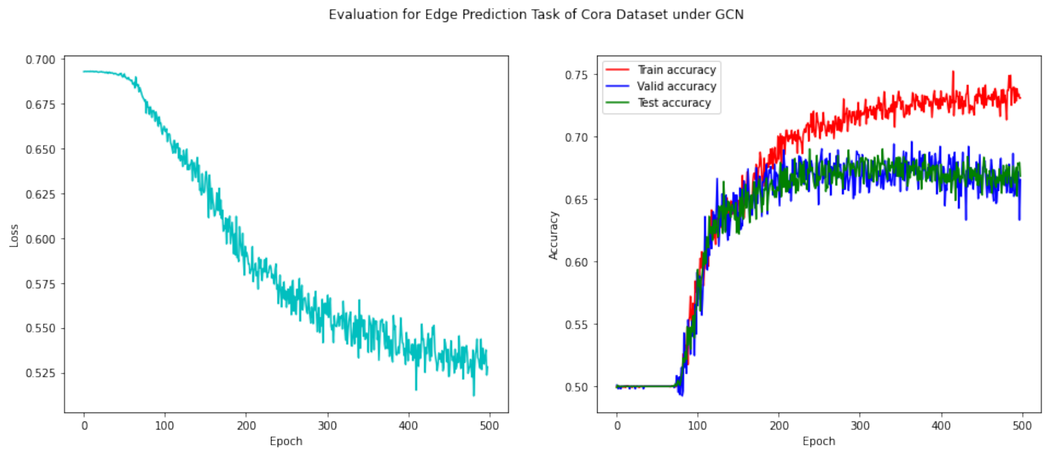


Figure 5: Test accuracy and training loss dynamics on Cora dataset using edge GCN

We first used a 2-layer vanilla GCN with hidden dimension being 64. We used SGD optimizer with learning rate 0.1, weight decay $5 * 10^{-4}$ and momentum 0.9. The result can be seen in Figure 5. In our experiment, the best model achieves the following accuracy, Train: 73.39%, Val: 67.59%, Test: 66.82%.

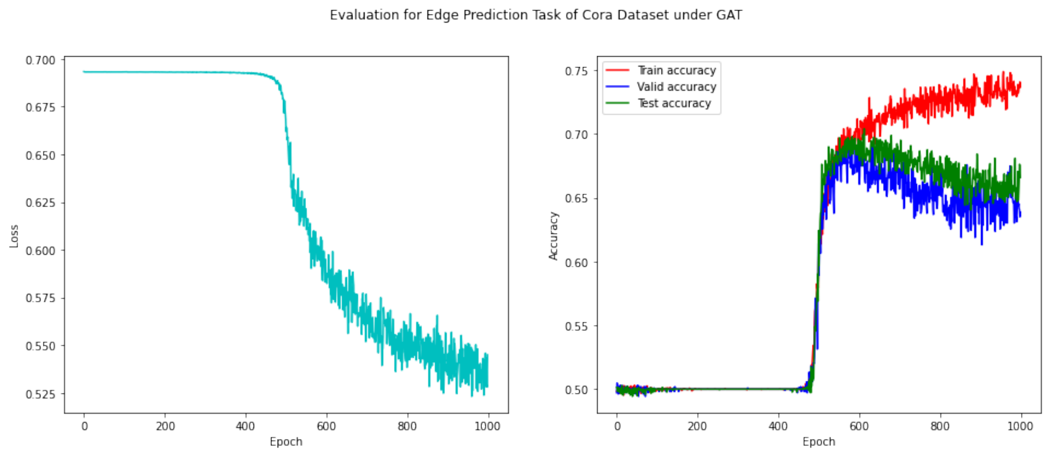


Figure 6: Test accuracy and training loss dynamics on Cora dataset using edge GAT

We then applied a 2-layer multihead attention GAT, with hidden dimension being 64 and $M = 2$. The best model achieves the following accuracy, Train: 73.89%, Val: 67.97%, Test: 66.59%. The training history has been presented in Figure 6. Interestingly, as we can see, the loss at first is not decreasing and begins to decrease until 500 epochs.

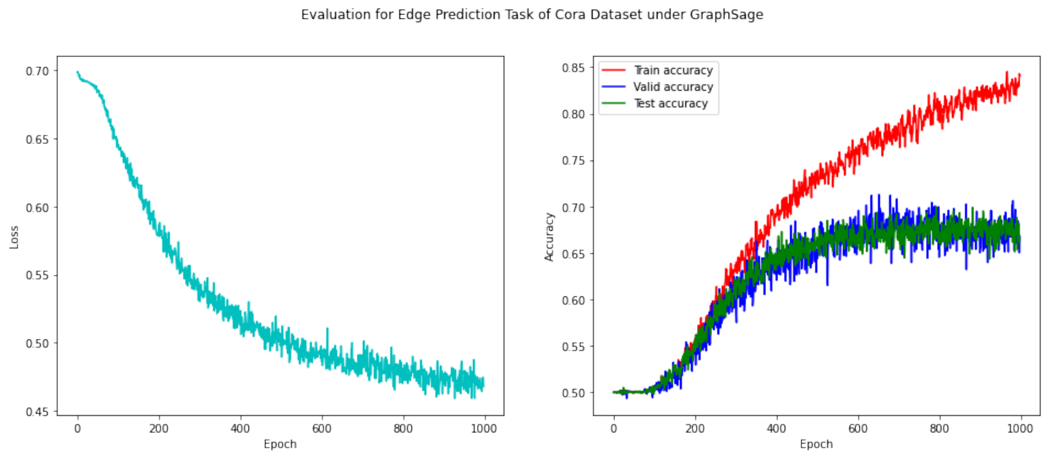


Figure 7: Test accuracy and training loss dynamics on Cora dataset using edge GraphSage

Finally, we tried a 2-layer GraphSage network with hidden dimension being 128. The result can be seen in Figure 7. The best model achieves the following accuracy, Train: 84.05%, Valid: 68.92% Test: 67.58%, which is better than previous two models.

4.3 DHT methods for edge-level tasks

We now still focus on the edge-prediction task, but considering the methods with DHT (dual hypergraph transformation) illustrated in section 3.1. The original graph is transformed to the dual hypergraph, then nodes in dual hypergraph represent the edges in original graph (see Figure 1), where the node-to-edge mapping is bijective. We then train a GNN on the dual hypergraph for node classification task, thus finishing the edge task on original graph. In this subsection, nodes and edges are split into three disjoint sets *Train*, *Valid* and *Test* with ratio $0.7, 0.1, 0.2$ respectively. We would apply GCN, GAT and GraphSage on the dual hypergraph and compare the results in section 4.2.

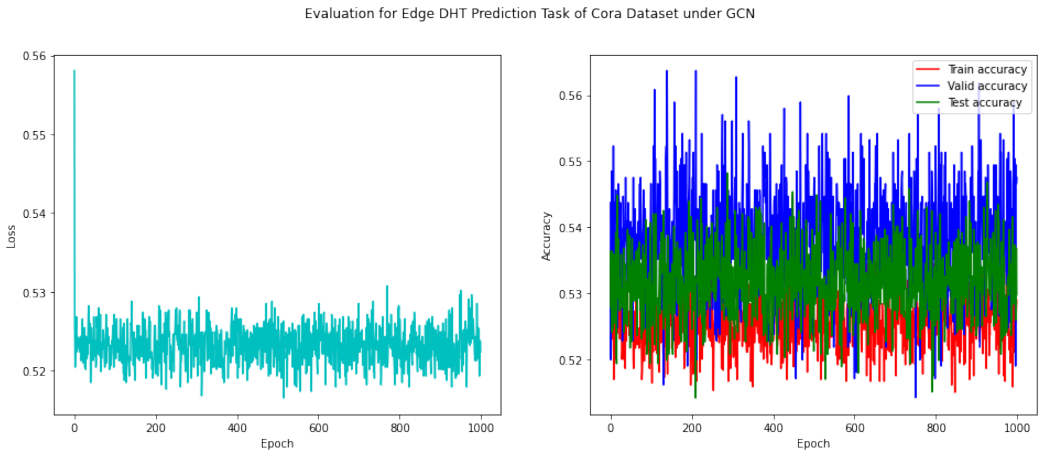


Figure 8: Test accuracy and training loss dynamics on Cora dataset using DHT and GCN.

We first tried a 2-layer vanilla GCN with hidden dimension being 128, and the result can be seen in Figure 8. In our experiment, the best model achieves the following accuracy, Train: 52.87%, Val: 56.72%, Test: 54.07%. Comparing with Figure 5, our new result is not better the classical edge classification with GCN. We believe it is because of hyper parameter tuning. As we can see in Figure 8, the loss decreases at first several steps and then fluctuates without convergence. Hence, we believe, it requires more careful selection of hyper parameters and the optimization tools. This task is more sensitive to training parameters.

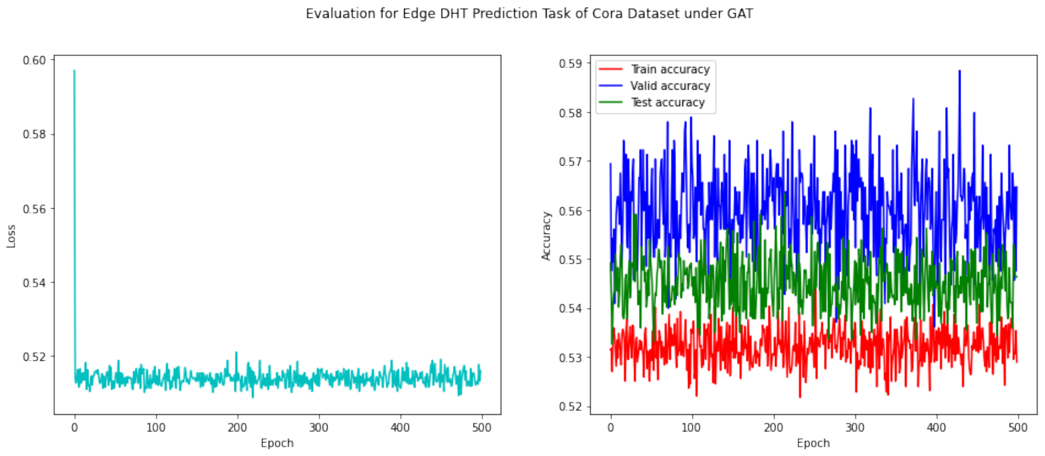


Figure 9: Test accuracy and training loss dynamics on Cora dataset using DHT and GAT.

We then applied a 2-layer multi-head attention GAT, with hidden dimension being 128 and $M = 2$. The best model achieves the following accuracy, Train: 53.71%, Val: 58.82%, Test: 54.64%. Similarly, this result is not as good as Figure 6. Figure 9 shows that the convergence of this training process is very fast. We have tried different hyper parameters, such as learning rate, weight decay rate, momentum and dropout rate, and could not find out a case where this new method is better than the result in section 4.2. This eventually indicates that using GAT on dual hyper graph for node classification is not a reasonable way for edge level tasks of the original graph.

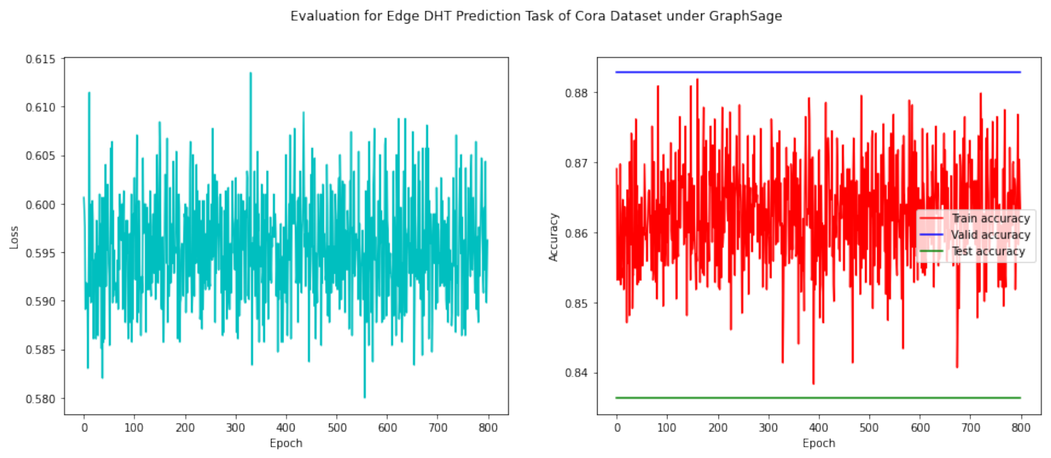


Figure 10: Test accuracy and training loss dynamics on Cora dataset using DHT GraphSage.

Finally, we tried a 2-layer GraphSage network with hidden dimension being 128. The result can be seen in Figure 10. The learning rate for Adam optimization we use is 0.001. The convergence of this model is very fast at first several steps and then stays stable and does not change anymore in the test and validation dataset. The best model achieves the following accuracy, Train: 86.33%, Valid: 88.28% Test: 83.63%, which is better than previous two models (Figure 8 and Figure 9). It is even better than all the results in section 4.2. In other words, our final experiments shows the the best performance of the edge-level tasks, which indicates that using DHT with GraphSage on dual hyper graph is a better approach of solving edge-level problems.

5 Discussion

In this project, we performed both node and edge classification task on **Cora** dataset, and implemented DHT method [1] for edge task as well. As we have seen in section 4.3, the DHT method could not outperform the traditional method using GCN and GAT, but it is the best method for edge-level tasks when using DHT with GraphSage on dual hyper graphs. Possible reasons are listed as follows.

1. Optimization tools and hyper parameter tuning. The loss function is highly non-convex with respect to training parameters, and there are plenty of local minimums. Thus the final results is sensitive to the initialization. Also, as emphasized in [32], the performance of the optimization tools, Adam and SGD, is sensitive to its hyper parameters. Therefore, future hyper parameter tuning is needed for further study.
2. The main drawback of the DHT method is that its complexity increases drastically when it is applied to dense graphs. Given graph $G = (V, E)$, let $|V|$ denote the number of nodes in G. The number of edges $|E|$ is at most $O(|V|)$ when G is sparse, thus the number of nodes in dual hypergraph is still $O(|V|)$. However, when it comes to dense graphs where number of edges $|E|$ is $O(|V|^2)$, the number of nodes in dual hypergraph increases to $O(|V|^2)$.
3. Graph pooling. As we reviewed in 3.2, in [1], the authors adopted some advanced graph pooling techniques when training on dual hypergraph, which may be efficient for very large graphs. We may need to employ and develop some graph pooling methods in [1] when training on dual hyper graph, which possibly reduces the complexity of the hypergraph mentioned above. For future work, it is reasonable to employ some graph pooling techniques in the hypergraph.
4. GraphSage is a framework for inductive representation learning on large graphs [31]. Our implementations indicate that GraphSage could be a better choice when training on hypergraph. As we know, GraphSage is usually used to generate low-dimensional vector representations for nodes, and is especially useful for graphs that have rich node attribute information. After applying DHT and transferring the edge-level tasks into a node-level

task on the dual hypergraph, it is more likely that such new graph has rich node attribute information in the structure, which indicated the best performance showed in Figure 10.

References

[1] Jaehyeong Jo, Jinheon Baek, Seul Lee, Dongki Kim, Minki Kang, and Sung Ju Hwang. Edge representation learning with hypergraphs. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=tSfud5OOqR`.

[2] Liyu Gong and Qiang Cheng. Exploiting edge features for graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9211–9219, 2019.

[3] Xiaodong Jiang, Pengsheng Ji, and Sheng Li. Censnet: Convolution with edge-node switching in graph neural networks. In *IJCAI*, pages 2656–2662, 2019.

[4] 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. *arXiv preprint arXiv:2005.00687*, 2020.

[5] William L Hamilton. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3):1–159, 2020.

[6] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[7] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1025–1035, 2017.

[8] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

[9] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[10] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3558–3565, 2019.

[11] Fenxiao Chen, Yun-Cheng Wang, Bin Wang, and C-C Jay Kuo. Graph representation learning: A survey. *APSIPA Transactions on Signal and Information Processing*, 9, 2020.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[13] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. *arXiv preprint arXiv:1911.03082*, 2019.

[14] Tetsu Kasanishi, Xueting Wang, and Toshihiko Yamasaki. Edge-level explanations for graph neural networks by extending explainability methods for convolutional neural networks. In *2021 IEEE International Symposium on Multimedia (ISM)*, pages 249–252. IEEE, 2021.

[15] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[16] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[17] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

[18] Hiroshi Kajino. Molecular hypergraph grammar with its application to molecular optimization. In *International Conference on Machine Learning*, pages 3183–3191. PMLR, 2019.

[19] Federico Monti, Oleksandr Shchur, Aleksandar Bojchevski, Or Litany, Stephan Günnemann, and Michael M Bronstein. Dual-primal graph convolutional networks. *arXiv preprint arXiv:1806.00770*, 2018.

[20] Edward R Scheinerman and Daniel H Ullman. *Fractional graph theory: a rational approach to the theory of graphs*. Courier Corporation, 2011.

[21] Yulei Yang and Dongsheng Li. Nenn: Incorporate node and edge features in graph neural networks. In *Asian conference on machine learning*, pages 593–608. PMLR, 2020.

[22] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. In *International Conference on Machine Learning*, pages 874–883. PMLR, 2020.

[23] Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. *arXiv preprint arXiv:2102.11533*, 2021.

[24] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.

[25] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International conference on machine learning*, pages 3734–3743. PMLR, 2019.

[26] Ekagra Ranjan, Soumya Sanyal, and Partha Talukdar. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5470–5477, 2020.

[27] Zhengdao Chen, Xiang Li, and Joan Bruna. Supervised community detection with line graph neural networks, 2020.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[29] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *ArXiv*, abs/1903.02428, 2019.

[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

[31] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017. URL `http://arxiv.org/abs/1706.02216`.

[32] Mohammad Rasool Izadi, Yihao Fang, Robert L. Stevenson, and Lizhen Lin. Optimization of graph neural networks with natural gradient descent. *2020 IEEE International Conference on Big Data (Big Data)*, pages 171–179, 2020.