Edge Representation Learning with Hypergraphs

Anonymous Author(s)
Affiliation
Address
email

Abstract

Graph neural networks have recently achieved remarkable success in representing graph-structured data, with rapid progress in both the node embedding and graph pooling methods. Yet, they mostly focus on capturing information from the nodes considering their connectivity, and not much work has been done in representing the *edges*, which are essential components of a graph. However, for tasks such as graph reconstruction and generation, as well as graph classification tasks for which the edges are important for discrimination, accurately representing edges of a given graph is crucial to the success of the graph representation learning. To this end, we propose a novel edge representation learning framework based on *Dual Hypergraph Transformation* (DHT), which transforms the edges of a graph into the nodes of a *hypergraph*. This dual hypergraph construction allows us to apply message passing techniques for node representations to edges. After obtaining edge representations from the hypergraphs, we then cluster or drop edges to obtain holistic graph-level edge representations. We validate our edge representation learning method with hypergraphs on diverse graph datasets for graph representation and generation performance, on which our method largely outperforms existing graph representation learning methods. Moreover, our edge representation learning and pooling method also 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.

1 Introduction

The recent demand in representing graph-structured data, such as molecular, social, and knowledge graphs, has brought remarkable progress in the *Graph Neural Networks* (GNNs) [44, 35]. Early works on GNNs [22, 16, 37] aim to accurately represent each node to reflect the graph topology, by transforming, propagating, and aggregating information from their neighborhoods based on message passing schemes [13]. More recent works focus on learning holistic graph-level representations, by proposing graph pooling techniques that condense the node-level representations into a smaller graph or a single vector. While such state-of-the-art node embedding or graph pooling methods have achieved impressive performances on graph-related tasks (e.g., node classification and graph classification), they have largely overlooked the *edges*, which are essential components of a graph.

Most existing GNNs, including ones that consider categorical edge features [29, 13], only implicitly capture the edge information in the learned node/graph representations when updating them. While a few of them aim to obtain explicit representations for edges [20, 14, 39], they mostly use them only to augment the node-level representations, and thus suboptimally capture the edge information. This is partly because many benchmark tasks for GNN performance evaluation, such as graph classification, do not require the edge information to be accurately preserved. Thus, on these benchmarks, simple MLPs without any connectivity information can sometimes outperform GNNs [9, 18]. However, for

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

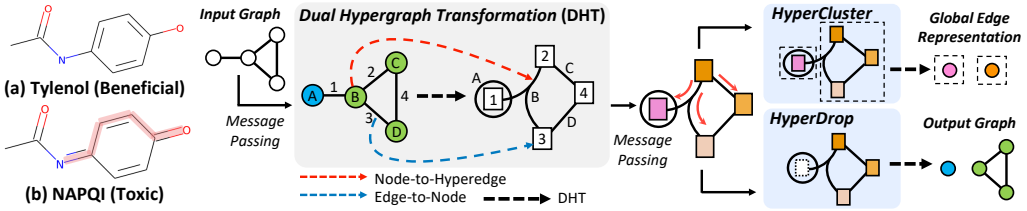

Figure 1: **(Left):** The two molecular graphs have the identical set of nodes, but exhibit completely different properties, due to their difference in the edges. **(Right):** An illustration of the proposed edge representation learning framework, with two novel edge pooling schemes. The grey box in the center describes the proposed **Dual Hypergraph Transformation**, where the numbers (letters) denote the corresponding edges (nodes) in the graph and nodes (hyperedges) in the hypergraph. The two boxes in the right illustrate the proposed edge pooling methods, **HyperCluster** which clusters similar edges, and **HyperDrop** which drops unnecessary edges.

tasks such as graph reconstruction and generation, accurately representing the edges of a graph is crucial to the success, as incorrectly reconstructing/generating edges may result in complete failure of the tasks. For example, the two molecules (a) and (b) in Figure 1 have exactly the same set of nodes and are only different in their edges (bond types), but exhibit extremely different properties.

To overcome such limitations of existing GNN methods in edge representation learning, we propose a simple yet effective scheme to represent the edges. The main challenge of handling edges is the absence or suboptimality of the message passing scheme for edges. We tackle this challenge by representing the edges as nodes in a *hypergraph*, which is a generalization of a graph where that can model higher-order interactions among nodes (a single hyperedge can connect an arbitrary number of nodes). Specifically, we propose *Dual Hypergraph Transformation* (DHT) to transform edges of the original graph to nodes of a *hypergraph* (Figure 1), and nodes to hyperedges. This hypergraph-based approach is effective since it allows us to apply any off-the-shelf message-passing schemes designed for node-level representation learning, for learning the representation of the edges of a graph.

However, representing each edge well alone is insufficient in obtaining an accurate representation of the entire graph. Thus we propose two novel graph pooling methods for the hypergraph to obtain compact graph-level edge representations, namely *HyperCluster* and *HyperDrop*. Specifically, for obtaining global edge representations for an entire graph, HyperCluster coarsens similar edges into a single edge under the global graph pooling scheme (see HyperCluster in Figure 1). On the other hand, HyperDrop drops unnecessary edges from the original graph by calculating pruning scores on the hypergraph (see HyperDrop in Figure 1). HyperCluster is more useful for graph reconstruction and generation as it does not result in removal of any edges, while HyperDrop is more useful for classification as it learns to remove edges that are less useful for graph discrimination.

We first experimentally validate the effectiveness of the DHT with *HyperCluster*, on the reconstruction of synthetic and molecular graphs. Our method obtains extremely high performance on these tasks, largely outperforming baselines, which shows its effectiveness in accurately representing the edges. Then, we validate DHT with *HyperCluster* on molecular graph generation tasks, and show that it largely outperforms base generation methods, as it allows to generate molecules with more correct bonds (edges). Further, we validate *HyperDrop* on 10 benchmark datasets for graph classification, on which *HyperDrop* outperforms all hierarchical pooling baselines, with larger gains on social graphs, for which the edge features are important. Our main contributions are as follows:

- We introduce a novel edge representation learning scheme using *Dual Hypergraph Transformation*, which exploits the dual hypergraph whose nodes are edges of the original graph, on which we can apply off-the-shelf message-passing schemes designed for node-level representation learning.
- We propose novel edge pooling methods for graph-level representation learning, namely *HyperCluster* and *HyperDrop*, to overcome the limitations of existing node-based pooling methods.
- We validate our methods on graph reconstruction, generation, and classification tasks, on which they largely outperform existing graph representation learning methods.

2 Related Work

Graph neural networks Graph neural networks (GNNs) mostly use the message passing scheme [13] to aggregate features from their neighbors. Particularly, Graph Convolutional Network (GCN) [22] generalizes the convolution operation in the spectral domain of graphs, and updates the representation of each node by applying the shared weights on it and its neighbors' representations.

Similarly, GraphSAGE [16] propagates the features of each node's neighbors to itself, based on simple aggregation operations (e.g., mean). Graph Attention Network (GAT) [32] considers the relative importance on neighboring nodes with attention, to update each node's representation as the weighted combination of its neighbors'. Xu et al. [37] show that a simple sum on neighborhood aggregation makes GNNs as powerful as the Weisfeiler-Lehman (WL) test [34], which is effective for distinguishing different graphs. While GNNs have achieved impressive success on graph-related tasks, most of them only focus on learning node-level representations, with less focus on the edges.

Edge-aware graph neural networks Some existing works on GNNs consider edge features while updating the node features [29, 31], however, they only use the edges as auxiliary information and restrict the representation of edges as the discrete features with categorical values. While a few methods [20, 13, 14, 39] explicitly represent edges by introducing edge-level GNN layers, they use the obtained edge features solely for enhancing node features. Also, existing message passing schemes for nodes are not directly applicable to edge-level layers, as they are differently designed from the node-level layers, which makes it challenging to combine them with graph pooling methods [40] for graph-level representation learning. We overcome these limitations by proposing a dual hypergraph transformation scheme, to obtain a hypergraph whose nodes are edges of the original graph.

Hypergraph Hypergraphs can model higher-order interactions among nodes, by grouping multi-node relationships into a single hyperedge [3]. Zhou et al. [43] first represent hypergraphs based on the spectral clustering technique, normalized cut [30]. Then, Feng et al. [11], Yadati et al. [38] propose methods that are similar to GCN [22] for representing hypergraphs, by generalizing a convolution operation on the spectral domain of the hypergraphs. There are a few studies that consider hypergraph duality [3, 28], which transforms the hyperedges into the nodes, to more conveniently deal with them. Following this line of research, Lugo-Martinez and Radivojac [25] use the duality to cast a hyperlink prediction task, as an instance of node classification from the dual form of the original hypergraph. On the other hand, Kajino [21] uses the duality to extract useful rules from the hypergraph structures by transforming molecular graphs, for their generation. However, none of the existing works exploit the relation between the original graph and the dual hypergraph for edge representation learning.

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. The simplest pooling approaches are using `mean`, `sum` or `max` over all node representations [1, 37]. However, they treat all nodes equally, and cannot adaptively adjust the size of graphs for downstream tasks. More advanced methods, such as node clustering methods, coarsen the graph by clustering similar nodes based on their embeddings [40, 4], whereas the node pruning methods reduce the number of nodes from the graph by dropping unimportant nodes based on their scores [12, 23]. Ranjan et al. [27] combine both node pruning and clustering approaches, by dropping meaningless clusters after grouping nodes. Baek et al. [2] propose to use attention-based operations for considering relationships between clusters. Note that all of those pooling schemes not only ignore edge representations, but also alter the node set by dropping, clustering, or merging nodes, which result in inevitable loss of node information.

3 Edge Representation Learning with Hypergraphs

In this section, we first introduce our novel edge representation learning framework with dual hypergraphs, which we refer to as *Edge HyperGraph Neural Network* (EHGNN), and then propose two novel edge pooling schemes for holistic graph-level representation learning: *HyperCluster* and *HyperDrop*. We begin with the descriptions of graph neural networks for node representation learning.

Graph Neural Networks. A graph G with n nodes and m edges, is defined by its node features $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, edge features $\boldsymbol{E} \in \mathbb{R}^{m \times d'}$, and the connectivity among the nodes represented by an adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$. Here, d and d' are the dimension of node and edge features, respectively. Then, given a graph, the goal of a *Graph Neural Network* (GNN) is to learn the node-level representation with message passing between neighboring nodes [13] as follows:

$$\boldsymbol{X}_v^{(l+1)} = \text{UPDATE}\left(\boldsymbol{X}_v^{(l)}, \text{AGGREGATE}\left(\left\{\boldsymbol{X}_u^{(l)} : \forall u \in \mathcal{N}(v; \boldsymbol{A})\right\}\right)\right), \tag{1}$$

where $\boldsymbol{X}^{(l)}$ is the node features 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 $\mathcal{N}(v; \boldsymbol{A})$ is the set of neighboring nodes for node v, obtained from the adjacency matrix $\boldsymbol{A}$. Such message passing schemes can incorporate the graph topology into each node by updating its representation with the representation of its neighbors.

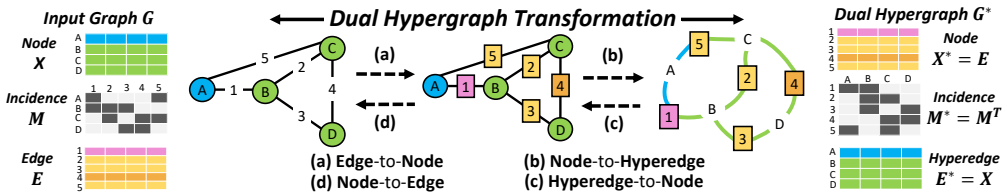

Figure 2: **Dual Hypergraph Transformation.** Illustration of the proposed graph-to-hypergraph transformation.

3.1 Edge-wise representation learning with dual hypergraph transformation

Edge representation learning. To reflect the edge information on message passing, some works on GNNs first obtain the categorical edge features between nodes, and then use them on the AGGREGATE function in equation 1, by adding or multiplying the edge features to the neighboring node's features [13, 29], for example AGGREGATE $\left(\left\{\boldsymbol{X}_u^{(l)} + \boldsymbol{E}_{u,v}^{(l)} : \forall u \in \mathcal{N}(v; \boldsymbol{A})\right\}\right)$ (see Section A.1 of the supplementary file for more details). Similarly, few recent works aim to obtain explicit edge representations, to use them as auxiliary information to augment the node features, by adding or multiplying edge features to them [14, 39]. Thus, existing works only implicitly capture the edge information in the learned node representations. Although this could be sufficient for most benchmark graph classification tasks, many real-world tasks with graphs (e.g., graph reconstruction and generation) require the edges to be accurately represented as they largely affect the task performance.

Even worse, to define a message passing function for edge representation learning, existing works propose to additionally create the adjacency matrix for edges (see Table 1). However, this is highly suboptimal since the transformation of the node adjacency to the edge adjacency requires $\mathcal{O}(m^2)$ time complexity (see Section A.2 of the supplementary file for detailed descriptions), as shown in Table 1. This is the main obstacle for directly applying the existing message passing schemes for nodes to edges. To this end, we propose a simple yet effective method to represent the edges of a graph, using a hypergraph.

Table 1: Comparison of the transformation of edge-aware GNNs [20, 14, 39] and the DHT, where the N->E denotes the transformation cost from nodes to edges for obtaining connectives for edges.

Model	Connectivity Patterns Node (N)	Edge (E)	Cost N->E
Edge-aware GNNs	$\boldsymbol{A} \in \mathbb{R}^{n \times n}$	${}^{e}\boldsymbol{A} \in \mathbb{R}^{m \times m}$	$\mathcal{O}(m^2)$
EHGNN (Ours)	$\boldsymbol{M} \in \mathbb{R}^{n \times m}$	$\boldsymbol{M}^T \in \mathbb{R}^{m \times n}$	$\mathcal{O}(1)$

Hypergraph. A *hypergraph* is a generalization of a graph which can model graph-structured data with higher-order interactions among nodes, wherein a single hyperedge connects an arbitrary number of nodes, unlike in conventional graphs where an edge can only connect two nodes. For example, in Figure 2, the hyperedge B defines the relation among three different nodes. To denote such higher-order relations among arbitrary number of nodes defined by a hyperedge, we use an *incidence matrix* $\boldsymbol{M} \in \{0,1\}^{n \times m}$, which represents the interaction between n nodes and m hyperedges, instead of using an adjacency matrix $\boldsymbol{A} \in \{0,1\}^{n \times n}$ that only considers interactions among n nodes. Each entry in the incidence matrix indicates whether the node is incident to the hyperedge. We can formally define a hyperagraph G^* with n nodes and m hyperedges, as a triplet of three components $G^* = (\boldsymbol{X}^*, \boldsymbol{M}^*, \boldsymbol{E}^*)$, where $\boldsymbol{X}^* \in \mathbb{R}^{n \times d}$ is the node features, $\boldsymbol{E}^* \in \mathbb{R}^{m \times d'}$ is the hyperedge features, and $\boldsymbol{M}^* \in \{0,1\}^{n \times m}$ is the incidence matrix of the hypergraph. We can also represent conventional graphs in the form of a hypergraph, $G = (\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E})$, in which a hyperedge in the incidence matrix $\boldsymbol{M}$ is associated with only two nodes. In the following paragraph, we will describe how to transform the edges of a graph into nodes of a hypergraph, for edge representation learning.

Dual Hypergraph Transformation. If we can change the role of the nodes and edges of the graph with a shared connectivity pattern across the nodes and edges, while accurately preserving their information, then we can use any node-based message passing schemes for learning edges. To this end, inspired by the hypergraph duality [3, 28], we propose to transform an edge of the original graph into a node of a hypergraph, and a node of the original graph into a hyperedge of the same hypergraph. We refer to this graph-to-hypergraph transformation as *Dual Hypergraph Transformation* (DHT) (see Figure 2). To be more precise, during the transformation, we interchange the structural role of nodes and edges from the given graph, obtaining the incidence matrix for the new dual hypergraph simply by *transposing* the incidence matrix of the original graph (see the incidence matrix in Figure 2 and Table 1). Along with the structural transformation through the incidence matrix, the DHT naturally interchanges node and edge features across G and G^* (see the feature matrices in Figure 2). Formally, given a triplet representation of a graph, DHT is defined as the following transformation:

$$\boldsymbol{DHT} \;:\; G = \big(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}\big) \;\mapsto\; G^* = \big(\boldsymbol{E}, \boldsymbol{M}^T, \boldsymbol{X}\big), \tag{2}$$

where we refer to the transformed G^* as the *dual hypergraph* of the input graph G. Since the dual hypergraph $G^* = (\boldsymbol{E}, \boldsymbol{M}^T, \boldsymbol{X})$ retains all the information of the original graph, we can recover the original graph from the dual hypergraph with the same DHT operation as follows:

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

This implies that DHT is a *bijective transformation*. DHT is simple to implement, does not incur the loss of any features or topological information of the input graph, and does not require additional memory for feature representations. Moreover, DHT can be sparsely implemented using the edge list, which is the sparse form of the adjacency matrix, by only reshaping the edge list of the graph into the hyperedge list of the dual hypergraph (see Section A.3 of the supplementary file for details), which is highly efficient in terms of time and memory. Thanks to DHT, we define the message passing between edges of the graph as the message passing between nodes of its dual hypergraph.

Message passing on the dual hypergraph for edge representation learning. After transforming the original graph into its corresponding dual hypergraph using DHT, we can perform the message passing between the edges of the input graph, by performing the message passing between the nodes of its dual hypergraph $G^* = (\boldsymbol{E}, \boldsymbol{M}^T, \boldsymbol{X})$, which is formally denoted as follows:

$$\boldsymbol{E}_e^{(l+1)} = \text{UPDATE}\left(\boldsymbol{E}_e^{(l)}, \text{AGGREGATE}\left(\left\{\boldsymbol{E}_f^{(l)} : \forall f \in \mathcal{N}(e; \boldsymbol{M}^T)\right\}\right)\right), \tag{4}$$

where $\boldsymbol{E}^{(l)}$ is the node features of G^* at 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. Here $\mathcal{N}(e; \boldsymbol{M}^T)$ is the neighboring node set of the node e in G^*, which we obtain using the incidence matrix $\boldsymbol{M}^T$ of G^*. Although acquiring the neighboring node set from the incidence matrix costs $\mathcal{O}(m)$, compared to using the adjacency matrix which costs $\mathcal{O}(n)$, this cost can be ignored by using the sparse implementation of DHT explained above. Note that, since the form of equation 4 is the same as the form of equation 1, we can use any graph neural networks which realize the message passing operation in equation 1, such as GCN [22], GAT [32], GraphSAGE [16], and GIN [37] for equation 4. In other words, to learn the edge representations $\boldsymbol{E}$ of the original graph, we do not require any specially designed layers, but simply need to perform DHT to directly apply existing off-the-shelf message passing schemes.

To simplify, we summarize the equation 1 as follows: $\boldsymbol{X}^{(l+1)} = \text{GNN}\left(\boldsymbol{X}^{(l)}, \boldsymbol{M}, \boldsymbol{E}^{(l)}\right)$, and the equation 4 as follows: $\boldsymbol{E}^{(l+1)} = \text{GNN}\left(\boldsymbol{E}^{(l)}, \boldsymbol{M}^T, \boldsymbol{X}^{(l)}\right) = \text{EHGNN}\left(\boldsymbol{X}^{(l)}, \boldsymbol{M}, \boldsymbol{E}^{(l)}\right)$, where EHGNN indicates our edge representation learning framework using DHT. After updating the edge features $\boldsymbol{E}^{(L)}$ with EHGNN, $\boldsymbol{E}^{(L)}$ is returned to the original graph by applying DHT on the dual hypergraph G^*. Then, the remaining step is how to make use of these edge-wise representations to accurately represent the edges of the entire graph, which we describe in the next subsection.

3.2 Graph-level edge representation learning with edge pooling

Existing graph pooling methods do not explicitly represent edges. To overcome this limitation, we propose two novel edge pooling schemes: *HyperCluster* and *HyperDrop*.

Graph pooling. The goal of graph pooling is to learn a holistic representation of the entire graph. The most straightforward approach for this, is to aggregate all the node features with `mean` or `sum` operations [1, 37], but they treat all nodes equally without consideration of which nodes are important for the given task. To tackle this limitation, recent graph pooling methods propose to either cluster and coarsen nodes [40, 4] or drop unnecessary nodes [12, 23]. While they yield improved performances on graph classification tasks, they suffer from an obvious drawback: inevitable loss of both node and edge information. The node information is lost as the nodes are dropped and coarsened, and the edge information is lost as the edges for the dropped nodes or the internal edges for the coarsened nodes are removed. To overcome this limitation, we propose a graph-level edge representation learning.

HyperCluster. We first introduce *HyperCluster*, which is a novel edge clustering method to coarsen similar edges into a single edge, for obtaining the global edge representation. Generally, a clustering scheme for nodes of the graph [40, 4] is defined as follows:

$$\boldsymbol{X}^{pool} = \boldsymbol{C}^T \boldsymbol{X'}, \ \ \boldsymbol{M}^{pool} = \boldsymbol{C}^T \boldsymbol{M}, \tag{5}$$

where $\boldsymbol{X}^{pool} \in \mathbb{R}^{n_{pool} \times d}$ and $\boldsymbol{M}^{pool} \in \mathbb{R}^{n_{pool} \times m}$ denote the pooled representations, $\boldsymbol{X'} = \text{GNN}\left(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}\right) \in \mathbb{R}^{n \times d}$ is the updated node features, and $\boldsymbol{C} \in \mathbb{R}^{n \times n_{pool}}$ is the cluster assignment

matrix that is generated from the $\boldsymbol{X'}$. Following this approach, the proposed HyperCluster clusters similar edges into a single edge, by clustering nodes of the dual hypergraph obtained from the original graph via DHT. In other words, we first obtain the node representation of the dual hypergraph $\boldsymbol{E'} = \text{EHGNN}(\boldsymbol{X}, \boldsymbol{M}, \boldsymbol{E}) \in \mathbb{R}^{m \times d'}$, and then cluster the nodes of the dual hypergraph as follows:

$$\boldsymbol{E}^{pool} = \boldsymbol{C}^T \boldsymbol{E'}, \;\; (\boldsymbol{M}^{pool})^T = \boldsymbol{C}^T \boldsymbol{M}^T \tag{6}$$

where $\boldsymbol{E}^{pool} \in \mathbb{R}^{m_{pool} \times d'}$ and $\boldsymbol{M}^{pool} \in \mathbb{R}^{n \times m_{pool}}$ denote the pooled edge representation and the incidence matrix of the input graph respectively, and $\boldsymbol{C} \in \mathbb{R}^{m \times m_{pool}}$ is the cluster assignment matrix generated from the input edge features $\boldsymbol{E'}$. Since HyperCluster coarsens the edges rather than dropping them, this edge pooling method is more appropriate for tasks such as graph reconstruction.

HyperDrop. We propose another edge pooling scheme, *HyperDrop*, which drops unnecessary edges to identify task-relevant edges, while performing lossless compression of nodes. Conventional node drop methods [12, 23] for graph pooling remove less relevant nodes based on their scores, as follows:

$$\boldsymbol{X}^{pool} = \boldsymbol{X}_{idx}, \;\; \boldsymbol{M}^{pool} = \boldsymbol{M}_{idx} \;; \;\; idx = \text{top}_k(\text{score}(\boldsymbol{X})), \tag{7}$$

where idx is the row-wise (i.e., node-wise) indexing vector, $\text{score}(\cdot)$ computes the score of each node with learnable parameters, and $\text{top}_k(\cdot)$ selects the top k elements in terms of the score. However, this approach results in the inevitable loss of node information, as it drops nodes. Thus, we propose to coarsen the graph by dropping edges instead of nodes, exploiting edge representations obtained from our EHGNN. *HyperDrop* selects the top ranked edges of the original graph, by selecting the top ranked nodes of the dual hypergraph. The pooling procedure for HyperDrop is formalized as follows:

$$\boldsymbol{E}^{pool} = \boldsymbol{E}_{idx}, \;\; (\boldsymbol{M}^{pool})^T = (\boldsymbol{M}^T)_{idx} \;; \;\; idx = \text{top}_k(\text{score}(\boldsymbol{E})). \tag{8}$$

Then, we can obtain the pooled graph $G^{pool} = (\boldsymbol{X}, \boldsymbol{M}^{pool}, \boldsymbol{E}^{pool})$ by applying DHT to the pooled dual hypergraph. HyperDrop is most suitable for graph classification tasks, as it identifies discriminative edges for the given task. Since HyperDrop preserves the nodes intact, it can also be used for node-level classification tasks, which is impossible with exiting graph pooling methods that modify nodes. A notable advantage of such HyperDrop is that it alleviates the over-smoothing problem in deep GNNs [24] (i.e., the features of all nodes converge to the same values when stacking a large number of GNN layers). As HyperDrop *learns* to remove unnecessary edges, the message passing only happens across relevant nodes, which alleviates over-smoothing.

4 Experiments

We experimentally validate our EHGNN that is coupled with either HyperCluster or HyperDrop on four different tasks: graph reconstruction, generation, classification, and node classification.

4.1 Graph reconstruction

Accurately reconstructing the edges is crucial for graph reconstruction tasks, and thus we validate the efficacy of our method on the graph reconstruction tasks first.

Experimental setup. We first validate our EHGNN with HyperCluster on the *edge reconstruction* tasks. Then, we evaluate it on the graph reconstruction tasks to validate our method's effectiveness in holistic graph-level learning. We start with edge reconstruction of a synthetic two-moon graph, where node features (coordinates) are fixed and edge features are colors. For edge and graph reconstruction of real-world graphs, we use a ZINC dataset [19] that consists of 12K molecular graphs [7], where node features are atom types and edge features are bond types. We use the accuracy, validity, and exact match as evaluation metrics. For more details, please see Section C.1 of the supplementary file.

Implementation details and baselines. We compare the proposed EHGNN framework against edge-aware GNNs, namely EGCN [17], MPNN [13], R-GCN [29], and EGNN [14], which use the edge features as auxiliary information for updating node features. We further combine them with an existing graph pooling method, namely GMPool [2], to obtain a graph-level edge representation for a given graph. In contrast, for our method, we first obtain edge representations with EHGNN, using GCN [22] as the message passing function, and then coarsen the edge-wise representations using HyperCluster, whose cluster assignment matrices are obtained using GMPool [2]. For node reconstruction, we set message passing to GCN and graph pooling to GMPool [2] for all models. We provide further details of the baselines and our model in Section C.1 of the supplementary file.

Figure 3: **Edge reconstruction results** on the ZINC molecule dataset by varying the compression ratio. Solid lines denote the mean of 5 different runs. High scores indicate the better.

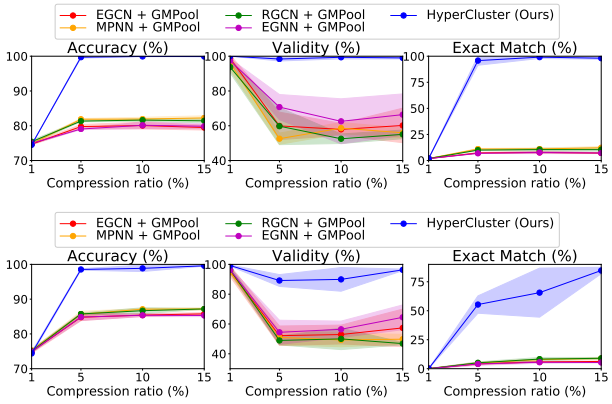

Figure 4: **Edge reconstruction results** of the synthetic two-moon graph. The edge features are represented as colors.

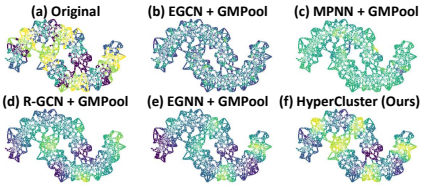

Figure 5: **Graph reconstruction results** on the ZINC molecule dataset by varying the compression ratio. Solid lines denote the mean, and shaded areas denote the standard deviation of 5 runs.

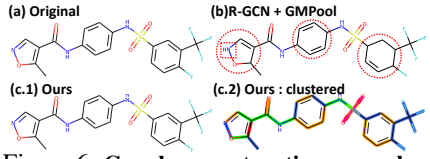

Figure 6: **Graph reconstruction examples.** Red dashed circles and squares indicate the incorrectly predicted edges and nodes, respectively. (c.2) shows the assigned clusters of edges as colors using our method.

Edge reconstruction results. Figure 4 shows the original two-moon graph and edge-reconstructed graphs, where edge features are colors, exhibiting clustered pattern. The baselines fail to reconstruct the edge colors, since they implicitly learn edge representations by using edge features as auxiliary information to update nodes, hence mixing the colors of the neighboring edges. On the other hand, our method distinguishes each edge cluster, which shows that our method can capture meaningful edge information by clustering similar edges. Moreover, as shown in Figure 3, our model obtains significantly higher performance over all baselines on the edge reconstruction task of molecular graphs, in all evaluation metrics. The performance gain of our method over baselines is notably large in exact match, which demonstrates that explicit learning of edge representation is essential for accurate encoding of the edge information.

Graph reconstruction results. We now validate our method on graph reconstruction for reconstructing both the nodes and edges of molecular graphs in Figure 5. Combining our edge representation learning method (EHGNN + HyperCluster) with existing node representation learning method (GCN + GMPool) yields incomparably high reconstruction performance compared to the baselines in exact match, which demonstrates that learning accurate edge representation, as well as node representation, is crucial to the success of the graph representation learning methods on graph reconstruction.

Qualitative analysis. We visualize the original and reconstructed molecular graphs in Figure 6. As shown in Figure 6 (b), the baseline cannot reconstruct the ring structures of the molecule, whereas our method perfectly reconstructs the rings as well as the atom types. The generated edge clusters in Figure 6 (c.2) further show that our method captures the detailed substructures of the molecule, as we can see in the cluster patterns of hexagonal and pentagonal rings.

Graph compression. To validate the effectiveness of our method in dense graph compression, we further apply EHGNN + HyperCluster to the Erdos-Renyi random graph [8] having six discrete edge features, with the number of nodes fixed to 10^3 while the number of edges increases from 10^3 to 10^4. In Figure 7, we report the relative size of compressed graphs from our method against the node pooling method, GMT [2]. As the number of edges increases, we observe that compressing only node features is insufficient for obtaining compact representations, whereas our method is able to obtain highly compact but accurate representation which can be assured from the sufficiently high edge reconstruction accuracy.

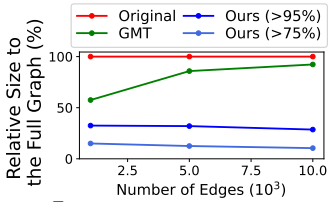

Figure 7: **Graph compression results.** For ours, we report the relative size to the full graph where the edge reconstruction accuracy is higher than 95% and 75%.

4.2 Graph generation

As shown in Figure 1 (left), graph generation depends heavily on the edge representations, as the model may generate incorrect graphs (e.g., toxic chemicals rather than drugs) if the edge information is inaccurate. Thus, we further validate our EHGNN on the graph generation tasks.

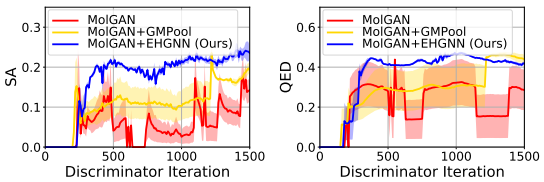

Figure 8: **Graph generation results on MolGAN.** Solid lines denote the mean scores of 3 different runs.

Datasets	Metrics	MARS [36]	MARS + EHGNN (Ours)
ZINC15	Success Rate	59.53 ± 2.11	**64.30** ± 1.54
	QED (≥ 0.67)	95.71 ± 0.09	**96.36** ± 0.49
	GSK3β (≥ 0.6)	86.52 ± 1.67	**90.63** ± 2.57
	JNK3 (≥ 0.6)	71.52 ± 4.15	**73.60** ± 1.29
ChEMBL	Success Rate	56.64 ± 5.79	**58.25** ± 6.07
	QED (≥ 0.67)	91.01 ± 2.79	**91.13** ± 4.84
	GSK3β (≥ 0.6)	87.45 ± 1.73	**90.34** ± 2.65
	JNK3 (≥ 0.6)	**70.57** ± 4.75	70.01 ± 4.83

Table 2: **Graph generation results on MARS.** The results are the mean and standard deviation of 3 runs.

Experimental setup. We directly forward the edge representation from the EHGNN to molecule generation networks, namely MolGAN [5] and MArkov moleculaR Sampling (MARS) [36]. MolGAN uses the Generative Adversarial Network (GAN) [15], to generate the molecular graph by balancing weights between its generator and discriminator. MolGAN uses R-GCN [29] for node-level message passing, whereas, for ours, we first obtain the edge representations using EHGNN, and use them with mean pooling in the graph encoder. For evaluation metrics, we use the Synthetic Accessibility (SA) and Druglikeness (QED) scores. We further apply EHGNN to MARS [36] that generates the molecule by sequentially adding or deleting its fragment, with MCMC sampling. While the original model uses MPNN [13] to implicitly obtain edge representations for the actions, we use EHGNN to explicitly learn edge representations. We train models to maximize the four molecule properties: the inhibition scores against two proteins, namely GSK3β and JNK3 (Biological); QED and SA scores (Non-biological). Then we report the success rate of which the molecule satisfies all the properties. For more details, please see Section C.2 of the supplementary file.

MolGAN result. Figure 8 shows the SA and QED scores of the generated molecules, of the MolGAN architecture with different encoders. Our EHGNN framework obtains significantly improved generation performance, over the original MolGAN which uses the R-GCN encoder and the MolGAN with GMPool, a state-of-the-art global node pooling encoder. This is because EHGNN learns explicit edge representation which enhances the ability of the discriminator to distinguish between real and generated graphs. The improvement in the discriminator also leads to notably more stable results compared to the baseline, which shows large variance in the quality of the generated molecules.

MARS result. To perform correct editing actions to generate graphs with MARS, we need accurate edge representations, as edges determine the structure of the generated molecule. Table 2 shows that using our EHGNN achieves significantly higher generation performance over original MARS, that uses edges as auxiliary information only to enhance node representations. Notably, performance gain on the GSK3β metric for which structural binding is important, suggests that accurate learning of edges is beneficial in generating more effective molecules that interact with the target protein.

4.3 Graph and node classification

Now, we validate the performance of our EHGNN with HyperDrop on classification tasks. Our approach is effective for classification of graphs with or without edge features, since it allows lossless compression of nodes and drops edges to allow message passing only across relevant nodes.

Experimental setup. Following the experimental setting of Baek et al. [2], we use the GCN as the node-level message passing layers for all models, and compare our edge pooling method against existing graph pooling methods. For this experiment, our HyperDrop uses SAGPool [23] on the hypergraph, which is a node drop pooling method based on self-attention. We use 6 datasets from TU datasets [26] including three from the biochemical domains (i.e., DD, PROTEINS, MUTAG) and the remaining half from the social domains (i.e., IMDB-BINARY, IMDB-MULTI, COLLAB). Also, we further use the 4 molecule datasets (i.e., HIV, Tox21, ToxCast, BBBP) from the recently released OGB datasets [17]. We evaluate the accuracy of each model with 10-fold cross validation [42] on the TU datasets, and use ROC-AUC as the evaluation metric for OGB datasets. For both datasets, we follow the standard experimental settings, from the feature extraction to the dataset splitting. We provide additional details of the experiments in Section C.3 of the supplementary file.

Baselines. We compare our EHGNN with HyperDrop, against the set encoding (DeepSet [41]), GNNs with naive pooling baselines (GCN and GIN [22, 37]), and state-of-the-art hierarchical pooling methods (DiffPool [40], SAGPool [23], TopKPool [12], MinCutPool [4], ASAP [27], EdgePool [6], and HaarPool [33]) that drop or coarsen node representations. We also additionally compare or combine the state-of-the-art global node pooling methods (SortPool [42], GMT [2]) with our model, for example HyperDrop + GMT. For more details, see Section C.3 of the supplementary file.

Table 3: **Graph classification results**. The results are the mean and standard deviation over 10 different runs. Best performance and its comparable results ($p > 0.05$) from the t-test are highlighted in bold. Hyphen (-) denotes out-of-resources that take more than 10 days. The results for the baselines are taken from Baek et al. [2].

		TU : Biochemical			TU : Social			OGB : Molecule				Average
		D&D	PROTEINS	MUTAG	IMDB-B	IMDB-M	COLLAB	HIV	Tox21	ToxCast	BBBP	
# graphs		1178	1113	188	1000	1500	5000	41127	7831	8576	2039	
# classes		2	2	2	2	3	3	2	12	617	2	
Avg # nodes		284.32	39.06	17.93	19.77	13.00	74.49	25.51	18.57	18.78	24.06	
Avg # edges		715.66	72.82	19.79	96.53	65.94	2457.78	27.47	19.27	19.26	25.95	
Set	DeepSet	77.39 ± 0.67	68.95 ± 0.92	72.56 ± 1.09	72.42 ± 0.36	50.24 ± 0.32	75.27 ± 0.21	71.20 ± 1.26	72.25 ± 0.23	59.44 ± 0.39	63.64 ± 0.62	68.34
Naive GNN	GCN	72.05 ± 0.55	73.24 ± 0.73	69.50 ± 1.78	73.26 ± 0.46	50.39 ± 0.41	80.59 ± 0.27	76.81 ± 1.01	75.04 ± 0.80	60.63 ± 0.51	65.47 ± 1.73	69.70
	GIN	70.79 ± 1.17	71.46 ± 1.66	81.39 ± 1.53	72.78 ± 0.86	48.13 ± 1.36	78.19 ± 0.63	75.95 ± 1.35	73.27 ± 0.84	60.83 ± 0.46	**67.65** ± 3.00	70.04
Global	SortPool	75.58 ± 0.72	73.17 ± 0.88	71.94 ± 3.55	72.12 ± 1.12	48.18 ± 0.83	77.87 ± 0.47	71.82 ± 1.63	69.54 ± 0.75	58.69 ± 1.71	65.98 ± 1.70	68.49
	GMT	**78.72** ± 0.59	**75.09** ± 0.59	83.44 ± 1.33	73.48 ± 0.76	50.66 ± 0.82	80.74 ± 0.54	**77.56** ± 1.25	**77.30** ± 0.59	**65.44** ± 0.58	**68.31** ± 1.62	73.07
Hierarchical	DiffPool	77.56 ± 0.41	73.03 ± 1.00	79.22 ± 1.02	73.14 ± 0.70	**51.31** ± 0.72	78.68 ± 0.43	75.64 ± 1.86	74.88 ± 0.81	62.28 ± 0.56	**68.25** ± 0.96	71.40
	SAGPool	74.72 ± 0.82	71.56 ± 1.49	73.67 ± 4.28	72.55 ± 1.28	50.23 ± 0.44	78.03 ± 0.31	71.44 ± 1.67	69.81 ± 1.75	58.91 ± 0.80	63.94 ± 2.59	68.49
	TopKPool	73.63 ± 0.55	70.48 ± 1.01	67.61 ± 3.36	71.58 ± 0.95	48.59 ± 0.72	77.58 ± 0.85	72.27 ± 0.91	69.39 ± 2.02	58.42 ± 0.91	65.19 ± 2.30	67.47
	MinCutPool	**78.22** ± 0.54	74.72 ± 0.48	79.17 ± 1.64	72.65 ± 0.75	51.04 ± 0.70	80.87 ± 0.34	75.37 ± 2.05	75.11 ± 0.69	62.48 ± 1.33	65.97 ± 1.13	71.56
	ASAP	76.58 ± 1.04	73.92 ± 0.63	77.83 ± 1.49	72.81 ± 0.50	50.78 ± 0.75	78.64 ± 0.50	72.86 ± 1.40	72.24 ± 1.66	58.09 ± 1.62	63.50 ± 2.47	69.73
	EdgePool	75.85 ± 0.58	**75.12** ± 0.76	74.17 ± 1.82	72.46 ± 0.74	50.79 ± 0.59	-	72.66 ± 1.70	73.77 ± 0.68	60.70 ± 0.92	67.18 ± 1.97	-
	HaarPool	-	-	66.11 ± 1.50	73.29 ± 0.34	49.98 ± 0.57	-	-	-	-	66.11 ± 0.82	-
Ours	HyperDrop	**78.74** ± 0.68	**75.30** ± 0.45	84.00 ± 0.69	73.96 ± 0.41	**51.68** ± 0.41	**81.29** ± 0.25	76.79 ± 0.86	76.95 ± 0.32	64.21 ± 0.70	**69.04** ± 0.86	**73.20**
	HyperDrop + GMT	**78.39** ± 0.33	**75.39** ± 0.26	**85.72** ± 0.61	**74.45** ± 0.61	**51.45** ± 0.28	80.59 ± 0.33	**77.84** ± 0.37	**77.58** ± 0.43	**65.15** ± 0.65	**69.16** ± 1.04	**73.57**

Model	MUTAG	PROTEINS	Tox21
HyperDrop	84.00 ± 0.69	**75.30** ± 0.45	**76.95** ± 0.32
HyperCluster	**84.50** ± 1.50	72.76 ± 1.12	76.68 ± 0.56
w/ RandDrop	83.06 ± 1.15	74.92 ± 0.51	76.39 ± 0.47
w/o HyperDrop	83.06 ± 1.20	75.08 ± 0.37	76.60 ± 0.45
w/o EHGNN	69.50 ± 1.78	73.24 ± 0.73	75.04 ± 0.80

Table 4: **Ablation study** of HyperDrop on the MUTAG, PROTEINS, and Tox21 datasets for classification.

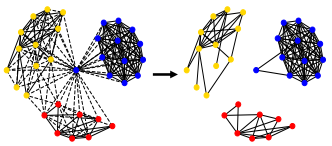

Figure 9: **Edge pooling results** on the COLLAB dataset. Colors denote connected components.

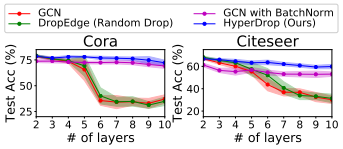

Figure 10: **Node classification results.** Lines denote means over 10 runs and shades denote variances.

Classification results. Table 3 shows that the proposed EHGNN + HyperDrop significantly outperforms all hierarchical pooling baselines. This is because HyperDrop not only retains nodes by removing edges that are less useful for graph discrimination, but also explicitly uses the edge features for graph classification tasks. Since HyperDrop does not remove any nodes on the graph, it can be jointly used with any node pooling methods, and thus, we pair HyperDrop with GMT. This model largely outperforms GMT, obtaining best performance on most of the datasets, which demonstrates that accurate learning of both the nodes and edges is important for classifiying graphs. We further visualize the edge pooling process of HyperDrop in Figure 9, which shows that our method accurately captures the substructures of the entire graph, which leads to dividing the large graph into the several connected components, thus adjusting the graph topology for more effective message passing.

Ablation study. To see how much each component contributes to the performance gain, we conduct an ablation study on EHGNN with HyperDrop. Table 4 shows that, compared with a model that only uses node features obtained by GCN, learning explicit edge representations significantly improves the performance. Our model with HyperCluster, or without HyperDrop, or the model with random edge drop obtains decent performance, but substantially underperforms HyperDrop.

Over-Smoothing with Deep GNNs. Lastly, we demonstrate that our EHGNN with HyperDrop alleviates the over-smoothing problem of deep GNNs on semi-supervised node classification tasks, following the setting of existing works [22, 32, 10]. We provide the experimental details in Section C.4 of the supplementary file. As shown in Figure 10, HyperDrop retains the accuracy as the number of layers increases, whereas the naive GCN or random drop results in largely degraded performance. Further, our method outperforms BatchNorm which alleviates over-smoothing by yielding differently normalized feature distribution at each node. This is because HyperDrop splits the given graph into smaller subgraphs that capture meaningful message passing substructures as shown in Figure 9.

5 Conclusion

We tackled the problem of accurately representing the edges of a graph, which has been relatively overlooked over node representation learning. To this end, we proposed a novel edge representation learning framework using *Dual Hypergraph Transformation* (DHT), which transforms the edges of the original graph into nodes on a hypergraph. This allows us to apply a message passing scheme for node representation learning for edge representation learning. Further, we proposed two edge pooling methods to obtain a holistic edge representation for a given graph, where one clusters similar edges into a single edge for graph reconstruction and the other drops unnecessary edges for graph classification. We validated our edge representation learning framework on graph reconstruction, generation, and classification tasks, showing its effectiveness over relevant baselines. We describe the **limitations and societal impacts** of our work in section E of the supplementary file.

References

[1] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 1993–2001, 2016.

[2] Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. In *International Conference on Learning Representations*, 2021.

[3] Claude Berge. Graphs and hypergraphs. 1973.

[4] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. Spectral clustering with graph neural networks for graph pooling. *arXiv preprint*, arXiv:1907.00481, 2019.

[5] Nicola De Cao and Thomas Kipf. Molgan: An implicit generative model for small molecular graphs. *arXiv preprint*, arXiv:1805.11973, 2018.

[6] Frederik Diehl. Edge contraction pooling for graph neural networks. *arXiv preprint arXiv:1905.10990*, 2019.

[7] Vijay Prakash Dwivedi, Chaitanya K. Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint*, arXiv:2003.00982, 2020.

[8] P. Erdős and A Rényi. On the evolution of random graphs. In *PUBLICATION OF THE MATHEMATICAL INSTITUTE OF THE HUNGARIAN ACADEMY OF SCIENCES*, pages 17–61, 1960.

[9] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[10] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. Graph random neural networks for semi-supervised learning on graphs. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[11] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 3558–3565. AAAI Press, 2019.

[12] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2083–2092. PMLR, 2019.

[13] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1263–1272. PMLR, 2017.

[14] Liyu Gong and Qiang Cheng. Exploiting edge features for graph neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 9211–9219. Computer Vision Foundation / IEEE, 2019.

[15] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *arXiv preprint*, arXiv:1406.2661, 2014.

[16] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1024–1034, 2017.

[17] 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.

[18] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin Benson. Combining label propagation and simple models out-performs graph neural networks. In *International Conference on Learning Representations*, 2021.

[19] John J. Irwin, Teague Sterling, Michael M. Mysinger, Erin S. Bolstad, and Ryan G. Coleman. ZINC: A free tool to discover chemistry for biology. *J. Chem. Inf. Model.*, 52(7):1757–1768, 2012.

[20] Xiaodong Jiang, Pengsheng Ji, and Sheng Li. Censnet: Convolution with edge-node switching in graph neural networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 2656–2662. ijcai.org, 2019.

[21] Hiroshi Kajino. Molecular hypergraph grammar with its application to molecular optimization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3183–3191. PMLR, 2019.

[22] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[23] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3734–3743. PMLR, 2019.

[24] Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3538–3545. AAAI Press, 2018.

[25] Jose Lugo-Martinez and Predrag Radivojac. Classification in biological networks with hypergraphlet kernels. *arXiv preprint*, arXiv:1703.04823, 2017.

[26] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2014–2023. JMLR.org, 2016.

[27] Ekagra Ranjan, Soumya Sanyal, and Partha P. Talukdar. ASAP: adaptive structure aware pooling for learning hierarchical graph representations. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 5470–5477. AAAI Press, 2020.

[28] Edward Scheinerman and Daniel Ullman. *Fractional graph theory: a rational approach to the theory of graphs*. Courier Coporation, 2011.

[29] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, volume 10843 of *Lecture Notes in Computer Science*, pages 593–607. Springer, 2018.

[30] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[31] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha P. Talukdar. Composition-based multi-relational graph convolutional networks. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[32] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[33] Yu Guang Wang, Ming Li, Zheng Ma, Guido Montúfar, Xiaosheng Zhuang, and Yanan Fan. Haarpooling: Graph pooling with compressive haar basis. *arXiv preprint arXiv:1909.11580*, 2019.

[34] B. Yu. Weisfeiler and A. A. Leman. Reduction of a graph to a canonical form and an algebra arising during this reduction. 1968.

[35] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[36] Yutong Xie, Chence Shi, Hao Zhou, Yuwei Yang, Weinan Zhang, Yong Yu, and Lei Li. Mars: Markov molecular sampling for multi-objective drug discovery. *arXiv preprint*, arXiv:2103.10432, 2021.

[37] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[38] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha P. Talukdar. Hypergcn: A new method for training graph convolutional networks on hypergraphs. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 1509–1520, 2019.

[39] Yulei Yang and Dongsheng Li. NENN: incorporate node and edge features in graph neural networks. In *Proceedings of The 12th Asian Conference on Machine Learning, ACML 2020, 18-20 November 2020, Bangkok, Thailand*, volume 129 of *Proceedings of Machine Learning Research*, pages 593–608. PMLR, 2020.

[40] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 4805–4815, 2018.

[41] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3391–3401, 2017.

[42] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 4438–4445. AAAI Press, 2018.

[43] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 1601–1608. MIT Press, 2006.

[44] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint*, arXiv:1812.08434, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We discuss them in section E of the supplementary file.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] We describe the potential societal impacts in section E of the supplementary file.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We provide the code, data, and instructions in the supplementary materials, with detailed experimental setups in section C of the supplementary file.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We specify them in section C of the supplementary file.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All of our main results contain variances or standard deviations of multiple runs.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We provide the details of computing resources in section C of the supplementary file.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A] If the assets that we used are brought from the public repository on the web, then we do our best to specify them with links in the section C of the supplementary file.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A] We do not additionally curate the new assets, but follow the standard settings of existing works on each task.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] Since we do not curate new datasets, we cite the dataset paper. Please see the original dataset paper for details.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] Since we do not curate new datasets, we cite the dataset paper. Please see the original dataset paper for details.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]