# Hypergraph Attention Isomorphism Network by Learning Line Graph Expansion

Sambaran Bandyopadhyay
*IBM Research AI & IISc*
Bangalore, India
samb.bandyo@gmail.com

Kishalay Das
*Indian Institute of Science*
Bangalore, India
kishalaydas@iisc.ac.in

M. Narasimha Murty
*Indian Institute of Science*
Bangalore, India
mnm@iisc.ac.in

2020 IEEE International Conference on Big Data (Big Data) | 978-1-7281-6251-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/BigData50022.2020.9378335

***Abstract*—Graph neural networks (GNNs) are able to achieve state-of-the-art performance for node representation and classification i n a n etwork. B ut, m ost o f t he e xisting G NNs c an be applied to simple graphs, where an edge connects only a pair of nodes. Studies have shown that hypergraphs are effective to model real-world relationships which are of higher order in nature. Recently, graph neural networks are proposed for hypergraphs, but they implicitly use clique or star expansions to convert the hypergraph to a simple graph, or use computationally expensive hypergraph Laplacian.**

**In this work, we propose a novel hypergraph neural network for semi-supervised hypernode classification, w hich operates directly on the hypergraphs with varying hyperedge sizes. Within each layer, it indirectly works on the line graph of the given hypergraph, without actually forming the line graph explicitly. Moreover, it also employs a self-attention mechanism to learn the weights of those edge relationships. Experimentally, HAIN is able to improve the state-of-the-art hypernode classification performance on all the datasets we use. We make the source code available to ease the reproducibility of the results.**

***Index Terms*—Graph Neural Network, Representation Learning, Hypernode Classification**

## I. Introduction

Graph representation learning got remarkable attention in the last few years. Performance on semi-supervised tasks such as node classification in a graph has improved significantly due to invent of different types of graph neural networks (GNNs) [1], [2]. Graph convolution network (GCN) [3] is one of the most popular graph neural network structure that aggregates the transformed attributes over the neighborhood of a node. Graph neural networks for node representation can differ in the way they aggregate attributes from a neighborhood [4]. Spatial neighborhood aggregation through node subsampling [5] and via self-attention [2] are also proposed. GNNs are typically learned in a semi-supervised way by minimizing the cross entropy loss of predicting the labels of a subset of nodes. More recently, Graph Isomorphism Network (GIN) is proposed in [6], which is theoretically shown to be as powerful as the Weisfeiler-Lehman graph isomorphism test. Researchers have come up with higher order GNNs [7] which can aggregate information from a higher order neighborhood of a node.

Most of the existing graph neural network approaches are suited for *simple graphs*, i.e., when the relationship between the nodes is pairwise. In such a graph, each edge connects only two vertices (or nodes). However, real life interactions among the entities are more complex in nature and relationships can be of high-order (beyond pairwise connections). For example, when four authors write a research paper together, it is not necessary that any two of them are already connected directly [8]. They are still coauthors because of some other coauthor in the paper who is strongly connected to both of them. Thus, representing such a co-authorship network by a simple graph may not be suitable. Hypergraphs [9] are introduced to model such complex relationships among the real world entities in a graph structure. In a hypergraph, each edge may connect more than two nodes. So an edge is essentially denoted by a subset of nodes, rather than just a pair. Edges in a hypergraph are called hyperedges and nodes are called hypernodes. Computation on hypergraphs is more expensive and also complicated. But due to their power to capture real world interactions, it is important to design learning algorithms for hypergraph representation.

Network analysis community shows interest in different applications of hypergraph mining [10], [11]. There exist different ways to transform a hypergraph to a simple graph such as via clique expansion and star expansion [12]–[14]. Clique expansion creates cliques by connecting any pair of nodes belonging to the same hyperedge in the transformed simple graph. The star expansion creates a bipartite graph by replacing each hyperedge by a new node, and then connects that node to all the hypernodes which belong to that hyperedge. Typically, the analysis on a hypergraph is done by performing the downstream mining tasks on the transformed simple graph. There are also tensor based approaches [15] available to deal with hypergraphs, but typically they assume that the size of all the hyperedges in the hypergraph to be the same (i.e., uniform hypergraph[1]) [16], [17]. Very recently, some graph neural networks based approaches are proposed for hypergraphs [18], [19]. Many of them implicitly use clique expansion [20] or hypergraph Laplacian [21] to use convolution on the hypergraph.

Line graph of a graph is a classical concept in graph theory [22], [23]. Edges of the original graph are mapped to the nodes of a line graph and edges in the line graph preserve the connectivity of the edges in the original graph. Very recently,

[1]A k-uniform hypergraph is a hypergraph with all its hyperedges of size k.

line graphs have been used in graph representation learning and graph neural networks [24]–[26]. They have shown that propagating information through edge-to-edge relationships helps the overall feature learning. Line graph of a hypergraph is still a simple graph (i.e., each edge connects only two nodes) [23]. Naturally, line graph can offer an effective way of applying graph neural networks and convolution on hypergraphs. Besides, many of the existing works on hypergraph neural networks [21], [27] assume that hyperedges are decomposable in nature. They inherently break an hyperedge into multiple nodes and tend to relate those nodes because of their common membership in the same hyperedge. But as suggested in [19], [27], relationship held within a complete hyperedge may not exist within an incomplete subset of the hyperedge. As line graph of a hypergraph creates a node for each entire hyperedge, it does not assume the notion of decomposability of an hyperedge in a hypergraph.

However, not much work is developed to exploit the concept of line graph in hypergraph analysis and representation. This is mainly due to the following two challenges: (i) Line graph of a hypergraph is computationally quite expensive to form and operate on, (ii) Mere conversion of a hypergraph to its line graph with heuristics to set edge weights (as proposed in [22]) in the line graph cannot capture the real dynamics between hyperedge-to-hyperedge relationships in the hypergraph. In this work, we precisely address them by making the following novel contributions.

- We propose a hypergraph neural network, referred as HAIN (**H**ypergraph **A**ttention **I**somorphism **N**etwork)[2], which directly operates on a hypergraph structure for semi-supervised hypernode classification. HAIN can handle hypergraphs with varying hyperedge sizes. Within each layer of HAIN, it indirectly works on the line graph of the given hypergraph, without actually forming it explicitly. In contrast to other approaches in general graph representation literature which use unweighted or static heuristics to define the weights of the edges of a line graph, our approach learns different weights for different edges in the line graph. A complete HAIN network can be formed by stacking multiple such layers and then training it on the cross entropy loss in an end to end fashion.
- Experimental comparison with state-of-the-art (SOTA) algorithms and model ablation study show the usefulness of HAIN and its different components. HAIN improves the SOTA performance of hypernode classification on multiple real-world hypergraph datasets. The source code of HAIN is made available at https://github.com/kdmsit/HAIN to ease the reproducibility of the results.

## II. Related Work

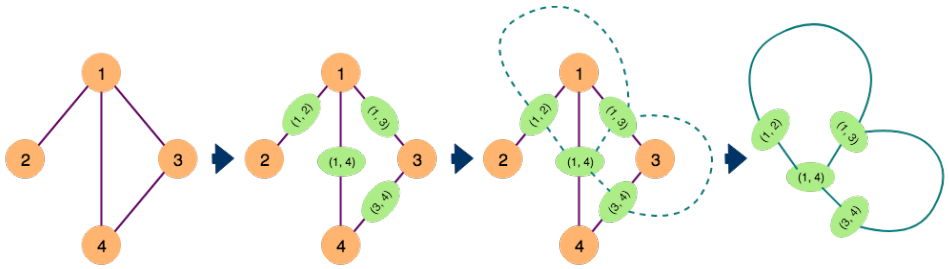
To give a brief but comprehensive idea about the existing literature, we discuss some of the prominent works in the following three domains.

[2]The term 'Isomorphism' is used because we use Graph Isomorphism Network (GIN) as the basic encoder in our proposed hypergraph neural network architecture.

**Network Embedding and Graph Neural Networks**: A detailed survey on network representation learning and graph neural networks can be found in [28], [29]. Conventional approaches based on random walk and matrix factorization have been applied to both on simple graph structure [30] and with node attributes [31], [32] for network embedding. Effort to apply neural networks on graph structure data started long time back in the literature [33]. Different types of semi-supervised graph neural networks exist in the literature for node representation and classification [1], [34]. GCN [35] is a popular semi-supervised message-passing GNN algorithm for node classification. GraphSAGE [5] extends GCN for inductive node classification by employing neighborhood sub-sampling. GAT [2] uses attention mechanism to learn the importance of a node to determine the label of another node in the neighborhood of it in the graph convolution framework. Theoretical analysis on the representation power of GNNs have been studied recently in multiple works [6], [7].

**Learning on Hypergraphs**: As mentioned in Section I, many of the existing analyses on a hypergraph first transform the hypergraph to a simple graph by clique expansion or star expansion [12]–[14], and then do the mining on the simple graph. Conventional analysis on simple graphs are done on the adjacency matrix as it captures the graph structure well. Similarly, in the hypergraph domain, higher order matrices, called tensors [15], are used for multiple computations. A non-negative tensor factorization approach is proposed in [16] for clustering a dataset having complex relations (beyond pairwise) in the form of a hypergraph. The main disadvantage of tensor based approaches is that, mostly they assume the hypergraph to be uniform, i.e., all the hyper edges are of equal size. Link prediction in hypergraphs is also studied in the literature [10]. Recently, a hypergraph based active learning scheme is proposed in [36], which allows one to ask both pointwise queries and pairwise queries.

**Hypergraph Neural Networks**: Application of graph neural networks for hypergraphs is still a new area of research. A hypergraph neural network (HGNN) is proposed in [20] which applies convolution on the hypergraph Laplacian. From Eq. 10 of [20], the framework boils down to the application of graph convolution (as proposed in [35]) on a weighted clique expansion of the hypergraph, where the weights of the edges of a clique are determined by the weight and degree of the corresponding hyperedge. Assuming that the initial hypergraph structure is weak, dynamic hypergraph neural network [18] is proposed by extending the idea of HGNN, where a dynamic hypergraph construction module is added to dynamically update the hypergraph structure on each layer. HyperGCN is proposed in [21], where the authors use the maximum distance of two nodes (in the embedding space) in a hyperedge as a regularizer. They use the hypergraph Laplacian to transform a hypergraph into a simple graph where each hyperedge is represented by a simple edge and the edge weight is proportional to the maximum distance between any pair of nodes in that hyperedge. Then they perform GCN on this simple graph structure. Graph representation and




(a) The Graph G (b) Vertices in L(G) added (c) Edges of L(G) added (d) The line graph L(G)

Fig. 1: Transformation process of a graph into its line graph. (a) Represents a simple graph $G$. (b) Each edge in the original graph has a corresponding node in the line graph. Here the green edges represent the nodes in line graph. (c) For each adjacent pair of edges in $G$ there exists an edge in $L(G)$. The dotted lines here are the edges in the line graph. (d) The line graph $L(G)$ of the graph $G$

neural network based approaches have also been proposed for heterogeneous hypergraph embedding [19], [27], [37]. Our work in this paper expands the literature on hypergraph neural network by proposing a novel GNN algorithm for hypergraphs and improving state-of-the-art performance on multiple real-world datasets.

## III. Problem Statement and Notations Used

We consider a hypergraph $\mathcal{H} = (V, E)$. Here $V$ is the set of hypernodes with $E$ is the set of hyperedges. A hyperedge $e$ connects a subset of nodes. For example, if the hyperedge $e$ connects $v_1$, $v_2$ and $v_3$, it is denoted as $e = \{v_1, v_2, v_3\}$. We also assume the hypergraph to be attributed, i.e., each node $v \in V$ is associated with a $F$ dimensional feature vector $x_v \in \mathbb{R}^F$ and this forms a feature matrix $X \in \mathbb{R}^{|V| \times F}$. We aim to propose a novel graph neural network to operate directly on the hypergraph and which can be used for semi-supervised hypernode classification. So, we assume to have a training set $V^s \subset V$ where for each hypernode $v \in V^s$, we know the label $l_v \in \mathcal{L}$ of it. Here, $\mathcal{L}$ is the set of labels. For example, $\mathcal{L} = \{-1, +1\}$ for a binary classification. Our goal is to learn a function $f : V \mapsto \mathcal{L}$ which can output label of each unlabelled hypernode $v \in V^u = V \setminus V^s$. The desired algorithm to learn such a function $f$ should be able to use both the hypergraph structure, along with the node attributes.

## IV. Proposed Solution: HAIN

In this section, we discuss the details of the proposed algorithm HAIN. It indirectly uses line graph of a hypergraph to capture the relationships between the edges of a hypergraph. But for the sake of presentation, we will assume it to form the line graph explicitly and learn the weights of the edges of the line graph. We will show the trick of avoiding the explicit formation of the line graph at the end of Section IV-B. This saves the expensive computation to form the line graph, but still able to exploit it to build the network. First, we brief the concept of line graph to make the paper self-contained.

### A. Line Graph and Extension to Hypergraphs

First, we define a line graph of a simple graph. Given a simple undirected graph $G = (V, E)$, the line graph $L(G)$ is the graph such that each node of $L(G)$ is an edge in $G$ and two nodes of $L(G)$ are neighbors if and only if their corresponding edges in $G$ share a common endpoint vertex [38]. Formally $L(G) = (V_L, E_L)$ where $V_L = \{(v_i, v_j) : (v_i, v_j) \in E\}$ and $E_L = \{\big((v_i, v_j), (v_j, v_k)\big) : (v_i, v_j) \in E\ ,\ (v_j, v_k) \in E\}$. Figure 1 shows how to convert a graph into the line graph.

Transforming a hypergraph to its line graph [23] is also quite similar to the case of a simple graph. Here, each hyperedge of the hypergraph becomes a node in the line graph and two nodes are connected in the line graph if the corresponding hyperedges share at least one hypernode in the hypergraph. More formally, for a given hypergraph $\mathcal{H} = (V, E)$ (as discussed in Section III), the line graph $L(\mathcal{H}) = (V_L, E_L)$ can be defined as follows: $V_L = \{\mathbf{v}_e \mid e \in E\}$, and $E_L = \{\{\mathbf{v}_{e_p}, \mathbf{v}_{e_q}\} \mid |e_p \cap e_q| \geq 1,\ e_p, e_q \in E\}$. Example of converting a hypergraph to its line graph can be seen in Figure 2, where the input is a hypergraph with 12 hypernodes and 5 hyperedges and the corresponding line graph has 5 nodes. Each node in the line graph has the same color as the corresponding hyperedge in the hypergraph. One can also assign static weight to the edges in the line graph by calculating the fraction of common nodes between the two hyperedges. But such static assignments are difficult to generalize. Instead, we aim to learn the edge importance through a graph neural network.

### B. Network Architecture of HAIN

In this subsection, we discuss the model update rule and the detailed neural architecture of HAIN. For a given hypergraph $\mathcal{H} = (V, E)$, with the number of hypernodes as $|V|$ and number of hyperedges as $|E|$, we assume each hypernode is associated with some $F$ dimensional attribute vector. The attribute matrix (or hypernode feature matrix) of $\mathcal{H}$ is denoted as $X \in \mathbb{R}^{|V| \times F}$. The incidence matrix $H \in \mathbb{R}^{|V| \times |E|}$ of the hypergraph is defined as $H(i, j) = 1$ if the hyperedge $j$ contains the hypernode $i$, and $H(i, j) = 0$ otherwise. A row of the incidence matrix gives how a hypernode is

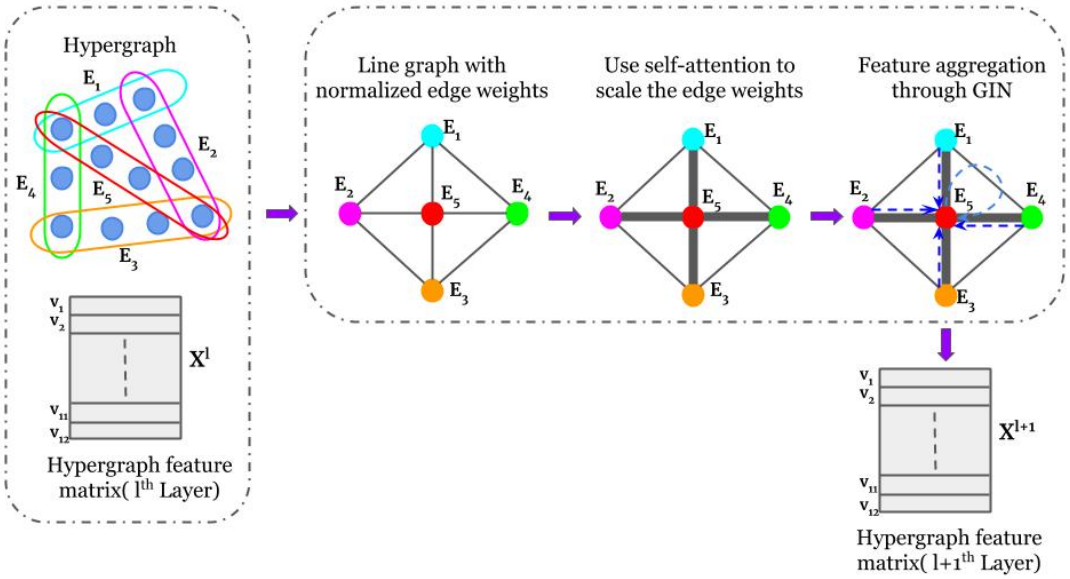



Fig. 2: Overall work flow inside the $l$-th layer of HAIN

connected to each hyperedge of the given hypergraph. Let us denote the hypernode degree matrix of $\mathcal{H}$ as $D_V \in \mathbb{R}^{|V|\times|V|}$, which is a diagonal matrix with the $i$th diagonal element as $D_V(i,i) = \sum_{j=1}^{|E|} H_{i,j}$. Similarly, hyperedge degree matrix of the hypergraph is $D_E \in \mathbb{R}^{|E|\times|E|}$, which is a diagonal matrix with the $j$th diagonal element as $D_E(j,j) = \sum_{i=1}^{|V|} H_{i,j}$. Next, we aim to convert the given hypergraph to its line graph (as discussed in Section IV-A), but within a layer of neural network as discussed below.

The adjacency matrix of the line graph of a given hypergraph contains information on how the nodes of the line graph (or equivalently hyperedges of the hypergraph) are connected to each other. A weighted version of the adjacency matrix of the line graph can be formed as $A_L = H^T H \in \mathbb{R}^{|E|\times|E|}$. Please note, it implicitly assumes that the nodes of the line graph contain self-loops. In this formulation of line graph, $A_L(i,j) = |e_i \cap e_j|$, i.e., the number of common hypernodes contained between the hyperedges $e_i$ and $e_j$ of the hypergraph. But then, the weight of an edge that connects two high degree hyperedges tends to be very high in the line graph. To avoid that, we use row normalization and the column normalization of the incidence matrix $H$ to form the adjacency matrix $A_L$ of the line graph as:

$$A_L = D_E^{-1} H^T D_V^{-1} H \ \in \mathbb{R}^{|E|\times|E|} \tag{1}$$

We assign features to each node of the line graph, again in the form of matrix multiplication as, $X_L = H^T X \in \mathbb{R}^{|E|\times F}$. If edge features of the hypergraph is available, those can also be directly used as the node features of the line graph.

Next, we choose Graph Isomorphism Network (GIN) [6] as the base GNN on the line graph. GIN has become the state-of-the-art graph neural network both due to its theoretical guarantee to represent graphs and experimental superiority. GIN has traditionally been used for graph classification purpose, but we achieve improved results by using it (along with the self-attention strategy to be discussed next) for hypernode classification[3]. The basic node feature update rule of the $l$th layer of a GIN for a general graph can be written as,

$$\begin{aligned} x_v^{l+1} &= \sigma\Big((1+\epsilon^l){W^l}^T x_v^l + \sum_{u\in\mathcal{N}(v)} {W^l}^T x_u^k\Big) \\ &= \sigma\Big(\sum_{u\in\mathcal{N}(v)\cup\{v\}} {W^l}^T x_u^k + \epsilon^l {W^l}^T x_v^l\Big) \end{aligned} \tag{2}$$

Here, $x_v^{l+1} \in \mathbb{R}^K$ is the hidden representation of the node $v$ in $(l+1)$th layer of GIN and $K$ is the feature dimension of the hidden layers of GIN. $\mathcal{N}(v)$ is the neighbors of the node $v$. $\epsilon$ is a parameter of GIN which determine the importance of a node's own representation with respect to the aggregated representation of its neighbors. $W^l$ is a trainable feature transformation matrix. We use a single layer neural network with a non-linear activation function $\sigma$, which is ReLU in all our experiments. Equation 2 can be written in the matrix form as $X^{l+1} = \sigma\Big((AX^l + \epsilon^l X^l)W^l\Big)$, where $A$ is the adjacency matrix of the graph with added self-loops.

Now for the purpose of hypernode representation of the given hypergraph $\mathcal{H} = (V, E)$, we seek to update the feature matrix $X$ via the node of the line graph. Hence, the hypernode

[3]However, one can easily replace GIN with any other GNN in our framework.

features can be updated as follows:

$$X^{l+1} = \sigma(H(A_L H^T X^l + \epsilon^l H^T X^l)W^l) \\ = \sigma(H(A_L X_L^l + \epsilon^l X_L^l)W^l) \in \mathbb{R}^{|V|\times F^{l+1}} \quad (3)$$

Here, $H^T X^l$ is the node feature matrix of the line graph derived from the updated features (by $(l-1)$th layer) of the hypernodes of the hypergraph. For the sake of notational convenience, we denote $X_L^l = H^T X^l$. The leftmost $H$ of Equation 3 maps the updates node features from line graph to the hypergraphs. Equation 3 is essentially to form a line graph with normalized edge weights, and then running GIN there, and transferring the updated node features from line graph to the hypergraph. Here, $A_L$ is the weighted adjacency matrix of the line graph as formed in Equation 1. We assume $F_l$ to be the dimension of the hypernode features at the $l$the layer of HAIN, $l = 0, 1, \cdots, L-1$. We assume, $X_L^0 = X_L = H^T X$ is the initial feature matrix. $W^l \in \mathbb{R}^{F_l \times F_{l+1}}$ and $\epsilon^l \in \mathbb{R}$ are the trainable parameters.

But the main drawback of this strategy is the use of a static line graph. The weight of the edge between any two node in the line graph in Equation 3 is just the normalized number of common nodes between the two corresponding hyperedges. But this static weight assignment does not count for the node features and their updates over the layers of the proposed GNN. Besides, existing works on simple graph embedding has shown that not all the edges are equally important for node representation learning [2]. Hence, for hypernode representation, we aim to learn different weights for different hyperedges. This reduces to associate different importance (weights) to different nodes in the line graph. We use a self-attention strategy [39] to learn them, as follows.

$X_{L,i}^{l+1}$ ($i$th row of $X_L^{l+1}$) is the updated feature of the $i$th node in the line graph, which is formed by aggregating neighborhood features through the graph neural network structure. We introduce an attention vector $\theta_{att}^l \in \mathbb{R}^{F_l}$. Clearly, $\sigma_{att}(X_L^l \theta_{att}^l) \in \mathbb{R}^{|E|}$ contains the importance of each hyperedge in the given hypergraph based on the updated hypernode features at each layer $l$ of the network. $\sigma_{att}$ is a non-linear activation function, for which we use softmax. We assume $\mathcal{D}(\cdot)$ to be a diagonal operator which takes a $|E|$ dimensional vector and converts it to a $|E| \times |E|$ dimensional diagonal matrix with the vector being set to the diagonal. Thus, $A_L \mathcal{D}(\sigma_{att}(X_L^l \theta_{att}^l))$ can be considered as the learned adjacency matrix of the line graph where values are scaled according to the importance of hyperedges of the hypergraph. Hence, the hypernode feature update rule for the $l$th layer ($\forall l = 0, 1, \cdots, L-1$) of HAIN can be written as:

$$X^{l+1} = \sigma\Big(H\big(\underbrace{A_L \mathcal{D}(\sigma_{att}(X_L^l \theta_{att}^l))}_{\text{Scaled adjacency matrix}} X_L^l \; + \; \epsilon^l X_L^l\big)W^l\Big) \quad (4)$$

As explained before, $X_L^l = H^T X^l$, and $X^0 = X$ (input hypergraph attribute matrix). We write the final update rule for the $l$-th layer of HAIN as a function of the input hypergraph and the previous layers, by expanding $A_L$ and $X_L$ below.

$$X^{l+1} = \sigma\Bigg(H\Big(D_E^{-1} H^T D_V^{-1} H \mathcal{D}\big(\sigma_{att}(H^T X^l \theta_{att}^l)\big) H^T X^l \\ + \; \epsilon^l H^T X^l\Big)W^l\Bigg) \; \in \mathbb{R}^{|V|\times F^{l+1}} \quad (5)$$

Equation 5 is insightful from a computational perspective. It contains a chain of matrix multiplications. If we choose to solve it from right-to-left order (which also turns out to be optimal in runtime, as discussed in Section IV-C2), we never need to compute the adjacency matrix of the given hypergraph explicitly. To clarify, let us consider the first term of the summation, which is $H D_E^{-1} H^T D_V^{-1} H \mathcal{D}\big(\sigma_{att}(H^T X^l \theta_{att}^l)\big) H^T X^l W^l$. So, we start multiplying $X^l W^l$, then multiplying $H^T$ with the resulting matrix, and so on. Following this order, we never compute $D_E^{-1} H^T D_V^{-1} H$ to form the expensive line graph $A_L$, which needs $O(|E|^2|V|)$ time. But interestingly, we are able to complete all the operations we intended to do on the explicit line graph.

The overall work flow of a layer of HAIN is summarized in Figure 2. The final hypernode features obtained from HAIN are fed to a softmax layer and use the following cross entropy loss on the training set for hypernode classification.

$$\mathcal{L}oss = -\sum_{v \in V^s} \sum_{k \in \mathcal{L}} y_{v,k} \; \ln \hat{y}_{v,k} \quad (6)$$

Here, $y_{v,k}$ is an indicator function if the actual label of a hypernode $v$ in $V^s$ of the hypergraph is $k$, and $\hat{y}_{v,k}$ is the probability with which the hypernode $v$ has the label $k$ from the output of softmax function. We use back-propagation algorithm with ADAM optimization technique [40] on the cross entropy loss to learn the parameters of the HAIN in an end-to-end fashion.

### C. Key Analysis and Complexity of HAIN

It can be noted that compared to conventional learning approaches on hypergraph, HAIN has multiple advantages. HAIN can handle non-uniform hypergraphs (i.e., hypergraphs with varying hyperedge sizes). In contrast to some existing graph representation techniques which employ the line graph, either as a separate stage [26], or store both the original and the line graph separately [24], [41], we do not need to keep the line graph explicitly in HAIN. This can be seen in Equation 5 where all the terms are part of the input hypergraph or some previous layer of HAIN. Also as explained above, the way we compute $X^{l+1}$ in Equation 5, we never actually form the line graph explicitly (please see Section IV-C2). This helps to reduce the space to store line graph separately and also to improve the scalability of HAIN. However, the analysis of HAIN with respect to the line graph gives cleaner insight about the approach. Moreover, we scale the values of the adjacency matrix of the line graph via self attention. This leads to learning the edge weights (structure) of the line graph rather than using unweighted or static strategy to set the edge weights as done in many existing literature [22], [26].

*1) Number of trainable Parameters:* In $l$th layer of HAIN, there are three sets of trainable parameters: $\epsilon^l \in \mathbb{R}$ and $W \in \mathbb{R}^{F_l \times F_{l+1}}$ and $\theta^l_{att} \in \mathbb{R}^{F_l}$. Thus, for a HAIN with $L$ layers, the total number of trainable parameters is $\sum_{l=0}^{L-1}\big(1 + F_l \times F_{l+1} + F_l\big) = O\big(\sum_{l=0}^{L-1} F_l(F_{l+1}+1)\big)$. As the input and intermediate feature dimensions are small, and the number of trainable parameters do not depend on the number of hypernodes or the number of hyperedges of the given hypergraph, HAIN can be trained efficiently.

*2) Time Complexity:* The order of matrix multiplication is important to optimize the runtime of HAIN in Equation 5. As discussed before, we adopt right-to-left matrix multiplication strategy, as the matrix $X^l \in \mathbb{R}^{|V| \times F_l}$ has the smallest dimension compared to others. For example, multiplying the diagonal matrix $\mathcal{D}(\sigma_{att}(H^T X^l \theta^l_{att}))$ with $H^T X^l$ would take $O(|E||V| + |E|F_l)$ time, and further left multiplying this with $H \in \mathbb{R}^{|V| \times |E|}$ takes $O(|V||E|F_l)$ time. Following this strategy, the total time to compute $X^{l+1}$ takes $O(|V||E|F_l F_{l+1})$. Hence, time complexity of one full forward pass of HAIN with $L$ layers is $O(|V||E|\sum_{l=0}^{L-1} F_l F_{l+1})$. The hypernode feature dimensions $F_l$, $\forall l$ are small, and typically for real-world hypergraphs, $O(|E|) = O(|V|)$. For example, in a citation hypergraph network, each paper (which is a hypernode) induces only one hyperedge which contains all the papers it cites [21]. Thus, $|V| = |E|$ for this. Besides, the final update rule of HAIN can be implemented by any state-of-the-art highly efficient and parallelizable matrix multiplication library. As mentioned at the end of Section IV-B, with this strategy of computing $X^{l+1}$, we never compute or store the adjacency matrix $A_L = D_E^{-1} H^T D_V^{-1} H \in \mathbb{R}^{|E| \times |E|}$ of the line graph. Hence, HAIN is highly scalable to large real-world hypergraph networks.

## V. Experimental Evaluation

We experiment on four publicly available popular network datasets for node classification and compare the results with state-of-the-art hypergraph neural networks. Please note that as the inherent objective of HAIN is semi-supervised in nature, we do not conduct experiment on unsupervised tasks such as hypernode clustering and hyperedge prediction. This is consistent with the recent hypergraph neural network literature [20], [21].

### A. Baseline Algorithms and Model Ablation Study

We have selected the following set of diverse state-of-the-art algorithms [4] to compare with the results of the proposed algorithm HAIN.

- **Confidence Interval based method (CI)** [11]: Authors have proposed a semi-supervised learning approach on hypergraphs and design an algorithm for solving the convex program based on the subgradient method.
- **Multi-layer perceptron (MLP)**: This is a classical MLP which only uses the node attributes for node classification problem. The hypergraph structure is completely ignored here.
- **MLP + explicit hypergraph Laplacian regularisation (MLP + HLR)**: Here the MLP is used along with an added component to capture the hypergraph Laplacian regularizer [21].
- **Hypergraph Neural Networks**: We use recently proposed state-of-the-art hypergraph neural networks like **HGNN** [20], **DHGNN** [18] and **HyperGCN** [21] with its variants.

To avoid any deterioration of performance of the baseline algorithms due to insufficient hyperparameter tuning, we report the hypernode classification results of the baseline algorithms (except for DHGNN) directly from [21] as our experimental setting is exactly the same as in there. As the results of DHGNN [18] on most of the datasets we used were not publicly available, we extensively tune the hyperparameters of DHGNN and report the test accuracy (averaged over 10 independent runs) corresponding to the best validation accuracy in Table II. More details on the experimental set up is presented in Sections V-B and V-C.

HAIN implicitly translates a given hypergraph to its line graph and then scale the edge weights of the line graph through self-attention before aggregating node features from the neighborhood using GIN. To show the benefit of each of these steps in HAIN, we do the following model ablation studies.

- **Star-GIN**: This is to translate a given hypergraph to a simple graph by star expansion and then run GIN on that.
- **Clique-GIN**: This is to translate a given hypergraph to a simple graph by clique expansion and then run GIN on that.
- **Static-HAIN**: This algorithm is obtained by removing the self-attention component which is used to learn the edge weights in the line graph. Thus, $l$th layer of this algorithm is equivalent to Equation 3.

The results of the above three algorithms are also presented, along with the other baselines.

### B. Datasets and Experimental Setup

We use three co-citation network datasets: Cora, Citeseer and Pubmed. We keep all the nodes of the original network in the hypergraph. If a research paper 'a' cites the papers 'b','c' and 'd', then we create a hyperedge $\{a, b, c, d\}$ in the hypergraph. Each node in these datasets is associated with an attribute vector. Attributes represent the occurrence of a word in the research paper through bag-of-words models. For Cora and Citeseer the attributes are binary vector and for Pubmed, they are tf-idf vectors. For the DBLP network, all documents co-authored by an author are in one hyperedge. High level summary of these datasets are presented in Table I.

For the experiments on hypernode classification in Section V-C, we adopt the same experimental setup as discussed in the

[4]We did not use those GNNs as baselines which either cannot work on hypergraphs directly [2], [35] or cannot handle hypergraphs of varying edge sizes [19].

| | **Cora** (co-citation) | **Citeseer** (co-citation) | **Pubmed** (co-citation) | **DBLP** (Co-authorship) |
|---|---|---|---|---|
| No of hypernodes | 2708 | 3312 | 19717 | 43413 |
| No of hyperedges | 1579 | 1079 | 7963 | 22535 |
| Avg. hyperedge size | 3.0 $\pm$ 1.1 | 3.2 $\pm$ 2.0 | 4.3 $\pm$ 5.7 | 4.7 $\pm$ 6.1 |
| No of features | 1433 | 3703 | 500 | 1425 |
| No of Classes | 7 | 6 | 3 | 6 |

TABLE I: Co-citation and co-authorship hypergraph datasets used in this work.

| **Method** | **Cora** (co-citation) | **Citeseer** (co-citation) | **Pubmed** (co-citation) | **DBLP** (Co-authorship) |
|---|---|---|---|---|
| CI | 35.60 $\pm$ 0.8 | 29.63 $\pm$ 0.3 | 47.04 $\pm$ 0.8 | 45.19 $\pm$ 0.9 |
| MLP | 57.86 $\pm$ 1.8 | 58.88 $\pm$ 1.7 | 69.30 $\pm$ 1.6 | 62.23 $\pm$ 2.0 |
| MLP+HLR | 63.02 $\pm$ 1.8 | 62.25 $\pm$ 1.6 | 69.82 $\pm$ 1.5 | 69.58 $\pm$ 2.1 |
| HGNN | 67.59 $\pm$ 1.8 | 62.60 $\pm$ 1.6 | 70.59 $\pm$ 1.5 | 74.35 $\pm$ 2.1 |
| DHGNN | 78.8 $\pm$ 1.25 | 63.45 $\pm$ 1.17 | 71.3 $\pm$ 1.33 | 74.65 $\pm$ 1.85 |
| 1-HyperGCN | 65.55 $\pm$ 2.1 | 61.13 $\pm$ 1.9 | 69.92 $\pm$ 1.5 | 66.13 $\pm$ 2.4 |
| FastHyperGCN | 67.57 $\pm$ 1.8 | 62.58 $\pm$ 1.7 | 70.52 $\pm$ 1.6 | 72.66 $\pm$ 2.1 |
| HyperGCN | 67.63 $\pm$ 1.7 | 62.65 $\pm$ 1.6 | 74.44 $\pm$ 1.6 | 75.91 $\pm$ 2.0 |
| Star-GIN | 75.19 $\pm$ 1.41 | 64.17 $\pm$ 0.73 | 76.91 $\pm$ 0.67 | 76.71 $\pm$ 0.85 |
| Clique-GIN | 76.38 $\pm$ 1.24 | 64.23 $\pm$ 0.95 | 74.59 $\pm$0.83 | 77.23 $\pm$ 0.97 |
| Static-HAIN | 77.17 $\pm$ 1.17 | 66.51 $\pm$ 0.83 | 76.25 $\pm$ 0.77 | 79.13 $\pm$ 0.95 |
| HAIN | **80.15 $\pm$ 1.71** | **68.89 $\pm$ 0.90** | **79.60 $\pm$ 0.67** | **81.69 $\pm$ 0.70** |

TABLE II: Results of hypernode classification (accuracy with standard deviation in %).

recent state-of-the-art [21]. To summarize, for each dataset, we take 5% hypernodes from each class in the training, 10% for validation and the remaining 85% for testing. This ensures that the training set is balanced with respect to the hypernode labels present in the respective datasets. For the experiments on the sensitivity analysis with respect to different training sizes in Sections V-D, we split each dataset into training and test sets, without doing any extra validation. We also keep the other hyperparameters same across those experiments to see the effect of only one hyperparameter.

For HAIN and the three variants (under model ablation study), we decrease the learning rate of the ADAM optimizer after every 100 epochs by a factor of 2. We use only 1 layer of HAIN and variants and keep the hidden dimension as 256. The initial learning rate is set to 0.03 and we train the algorithms for 200 epochs. We perform the experiments in a shared server having Intel(R) Xeon(R) Gold 6142 processor which contains 64 processors with 16 core each.

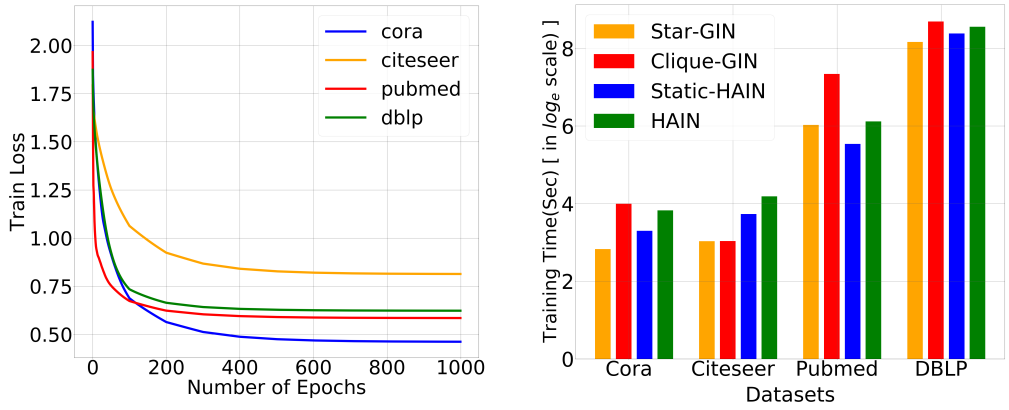
### *C. Performance on Hypernode Classification*

We run HAIN and its variants 10 times on each dataset and report the average hypernode classification accuracy and standard deviation on the test sets in Table II. First, it can be observed that all the GNN based approaches perform better than the first three baselines almost on all the datasets. HAIN turns out to be the best performing algorithm on all the datasets with significant improvement gap compared to the state-of-the-art GNN algorithms. HAIN is able to outperform the closest baseline DHGNN by roughly 2% on Cora and 9% on Citeseer and the closest baseline HyperGCN by roughly 6% on Pubmed and 7% on DBLP. The standard deviation of the performance of HAIN is also comparable or less than the baselines in most of the cases, which shows the stability and robustness of the proposed algorithm HAIN. Please note that the accuracy numbers of the baseline algorithms (as mentioned in [21]) and HAIN with its variants can improve if we add simple edges of the co-citation networks in the datasets, along with the hyperedges formed. But we still use the conventional formation of hypergraph networks to make the experimental comparisons consistent with the existing state-of-the-art hypergraph algorithms.

Usefulness of different components of HAIN can be seen by comparing the performance with the three variants as part of model ablation study. As expected, Star-GIN performs poorly as it creates a bipartite graph which is also heterogeneous in nature. The performance of Clique-GIN and Static-HAIN are comparable, with Static-GIN performing slightly better on Cora, Citeseer and DBLP. But the performance of HAIN is significantly better than Static-HAIN, which shows the importance of learning the edge weights of the line graph through self-attention mechanism, rather than using static normalized edge weights. HAIN and its variants are all GIN based hypergraph neural networks. Whereas, baselines such as HGNN and HyperGCN are based on GCN [35]. Better performance of Star-GIN or Clique-GIN compared to existing baselines also explains the right choice of GIN over GCN as the base GNN for hypernode classification.

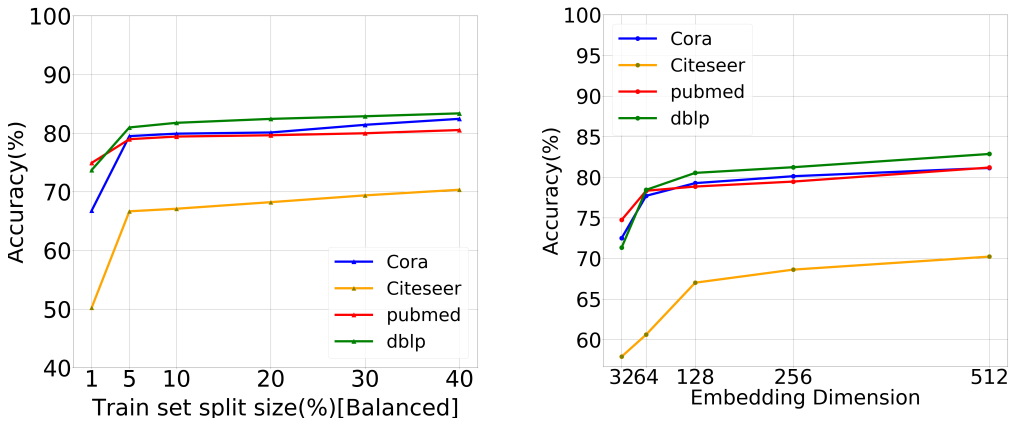
### *D. Further Results on Classification and Sensitivity Analysis*

**Convergence of loss in HAIN**: Convergence of training loss over the epochs is important to understand the stability of the results. Figure 3a shows that training losses of HAIN




(a) (b)

Fig. 3: (a) Shows HAIN training Loss over different epochs of the algorithm. (b) Training time of different approaches on four datasets.




(a) (b)

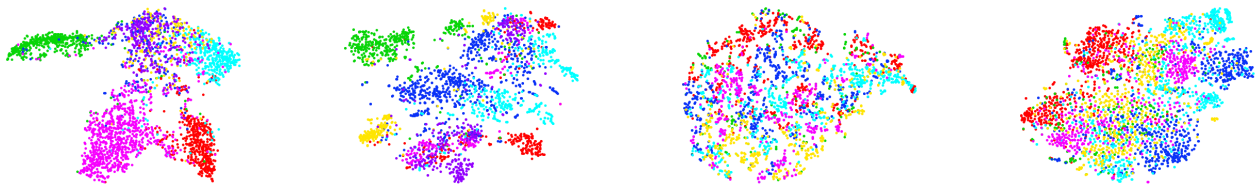
Fig. 4: (a) Shows node classification accuracy over different train-test sizes, (b) Shows node classification accuracy over different embedding dimensions.

over all the datasets are decreasing fast at the beginning and converge at the end. This is due to the use of ADAM and our strategy to decrease the learning rate after every 100 epochs of HAIN.

**Training time of HAIN and variants**: We have shown the training time of HAIN and the three related algorithms in Figure 3b. As expected, Star-GIN takes the least time as star expansion of a hypergraph is fast. But as shown in Table II, Star-GIN performs poorly on all the datasets. Static-HAIN takes less training time than HAIN as it does not use the self-attention mechanism to learn the edge weights in the line graph. Clique-GIN is the most expensive algorithm among them, as clique expansion is computationally heavy for hypergraphs and the resultant simple graph is also large.

**Learning Pattern of HAIN**: In Figure 4a, we change the training size from 1% to 40%, and remaining is the test set. We keep the training set balanced with respect to different class labels. We can see that the hypernode classification performance of HAIN improves significantly over increasing training set at the beginning, while almost saturates after 10%, for all the datasets. It shows that HAIN, being a semi-supervised algorithm, can work well with very less number of labeled hypernodes.

**Sensitivity w.r.t. embedding dimension**: HAIN maps the hypernodes to $F_L$ dimensional space before feeding them to the softmax layer. We vary this hidden layer dimension and plot the hypernode classification accuracy on all the datasets in Figure 4b. If the hidden layer dimension is not sufficiently large, it cannot capture enough information to represent the hypernodes. Figure 4b also shows the same where accuracy improves till dimension 128, beyond which the accuracy saturates. The analysis in this section also shows the consistent



(a) HyperGCN on Cora (b) **HAIN on Cora** (c) HyperGCN on Citeseer (d) **HAIN on Citeseer**

Fig. 5: t-SNE visualization of the hypernode representations (best seen in colors): There are 7 classes in Cora and 6 in Citeseer. These classes are coming more prominently by HAIN compared to that by HyperGCN.

behavior of HAIN across all the datasets, which is important to use it as a baseline algorithm in practice.

### *E. Hypernode Visualization*

HAIN is trained on the cross-entropy loss of hypernode classification, but vector representations of the hypernodes can be obtained from the output of the final layer of HAIN (before the softmax layer). We use t-SNE [42] which maps the vectors to two dimensional space to plot and visually see the quality of the embeddings. We show the hypernode visualization of HAIN and HyperGCN, only on Cora and Citeseer in Figure 5. Different colors represent different node classes. We can observe that cluster of points mostly belong to the same class in the visualization of HAIN compared to that in HyperGCN. This observation is consistent with the performance of HyperGCN and HAIN for hypernode classification in Table II.

## VI. Conclusion

In this work, we proposed a novel hypergraph neural network, referred as HAIN, for semi-supervised hypernode classification. In each layer of HAIN, the input hypergraph is implicitly transformed to a line graph whose edge weights are learned by a self-attention mechanism based on the updated hypernode features and GIN is applied there for neighborhood attribute aggregation. All the parameters of a HAIN network with multiple layers can be learned from an end-to-end fashion. The main advantage of HAIN is that, we never have to explicitly compute and store the expensive line graph, and thus it is scalable to large hypergraphs. Experimentally, HAIN is able to improve the state-of-the-art hypernode classification performance on all the datasets we use. Model ablation study shows the usefulness of each component of HAIN through experiments. As the domain of hypergraph neural network is still new, HAIN can encourage further development of novel algorithms and analysis towards this direction.

## References


[1] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*, 2016, pp. 2014–2023.

[2] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ

[3] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *arXiv preprint arXiv:1611.07308*, 2016.

[4] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1263–1272.

[5] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1025–1035.

[6] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ryGs6iA5Km

[7] H. Maron, H. Ben-Hamu, H. Serviansky, and Y. Lipman, "Provably powerful graph networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 2153–2164.

[8] Y. Han, B. Zhou, J. Pei, and Y. Jia, "Understanding importance of collaborations in co-authorship networks: A supportiveness analysis approach," in *Proceedings of the 2009 SIAM International Conference on Data Mining*. SIAM, 2009, pp. 1112–1123.

[9] A. Bretto, "Hypergraph theory," *An introduction. Mathematical Engineering. Cham: Springer*, 2013.

[10] F. Chen, Y. Gao, D. Cao, and R. Ji, "Multimodal hypergraph learning for microblog sentiment prediction," in *2015 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2015, pp. 1–6.

[11] C. Zhang, S. Hu, Z. G. Tang, and T. Chan, "Re-revisiting learning on hypergraphs: confidence interval and subgradient method," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 4026–4034.

[12] D. Zhou, J. Huang, and B. Schölkopf, "Learning with hypergraphs: Clustering, classification, and embedding," in *Advances in neural information processing systems*, 2007, pp. 1601–1608.

[13] S. Agarwal, K. Branson, and S. Belongie, "Higher order learning with graphs," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 17–24.

[14] L. Pu and B. Faltings, "Hypergraph learning with hyperedge expansion," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2012, pp. 410–425.

[15] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[16] A. Shashua, R. Zass, and T. Hazan, "Multi-way clustering using super-symmetric non-negative tensor factorization," in *European conference on computer vision*. Springer, 2006, pp. 595–608.

[17] S. R. Bulò and M. Pelillo, "A game-theoretic approach to hypergraph clustering," in *Advances in neural information processing systems*, 2009, pp. 1571–1579.

[18] J. Jiang, Y. Wei, Y. Feng, J. Cao, and Y. Gao, "Dynamic hypergraph neural networks." in *IJCAI*, 2019, pp. 2635–2641.

[19] R. Zhang, Y. Zou, and J. Ma, "Hyper-sagnn: a self-attention based graph neural network for hypergraphs," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=ryeHuJBtPH

[20] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3558–3565.

[21] N. Yadati, M. Nimishakavi, P. Yadav, V. Nitin, A. Louis, and P. Talukdar, "Hypergcn: A new method for training graph convolutional networks on hypergraphs," in *Advances in Neural Information Processing Systems*, 2019, pp. 1509–1520.

[22] T. S. Evans and R. Lambiotte, "Line graphs of weighted networks for overlapping communities," *The European Physical Journal B*, vol. 77, no. 2, pp. 265–272, 2010.

[23] J.-C. Bermond, M.-C. Heydemann, and D. Sotteau, "Line graphs of hypergraphs i," *Discrete Mathematics*, vol. 18, no. 3, pp. 235–241, 1977.

[24] F. Monti, O. Shchur, A. Bojchevski, O. Litany, S. Günnemann, and M. M. Bronstein, "Dual-primal graph convolutional networks," *arXiv preprint arXiv:1806.00770*, 2018.

[25] X. Jiang, P. Ji, and S. Li, "Censnet: convolution with edge-node switching in graph neural networks," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 2019, pp. 2656–2662.

[26] S. Bandyopadhyay, A. Biswas, M. Murty, and R. Narayanam, "Beyond node embedding: A direct unsupervised edge representation framework for homogeneous networks," *arXiv preprint arXiv:1912.05140*, 2019.

[27] K. Tu, P. Cui, X. Wang, F. Wang, and W. Zhu, "Structural deep embedding for hyper-networks," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[28] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *arXiv preprint arXiv:1901.00596*, 2019.

[29] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[30] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[31] S. Bandyopadhyay, N. Lokesh, and M. N. Murty, "Outlier aware network embedding for attributed networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 12–19.

[32] S. Bandyopadhyay, A. Biswas, H. Kara, and M. N. Murty, "A multilayered informative random walk for attributed social network embedding," in *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain*, vol. 325. IOS Press, 2020, pp. 1738–1745. [Online]. Available: https://doi.org/10.3233/FAIA200287

[33] A. Micheli, "Neural network for graphs: A contextual constructive approach," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.

[34] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rytstxWAW

[35] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2017.

[36] I. E. Chien, H. Zhou, and P. Li, "$Hs^2$: Active learning over hypergraphs with pointwise and pairwise queries," in *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019, pp. 2466–2475.

[37] I. M. Baytas, C. Xiao, F. Wang, A. K. Jain, and J. Zhou, "Heterogeneous hyper-network embedding," in *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2018, pp. 875–880.

[38] H. Whitney, "Congruent graphs and the connectivity of graphs," *American Journal of Mathematics*, vol. 54, no. 1, pp. 150–168, 1932.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[41] Z. Chen, L. Li, and J. Bruna, "Supervised community detection with line graph neural networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=H1g0Z3A9Fm

[42] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. [Online]. Available: http://www.jmlr.org/papers/v9/vandermaaten08a.html