Pattern Recognition 110 (2021) 107637

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/patcog

Hypergraph convolution and hypergraph attention

Song Bai*, Feihu Zhang, Philip H.S. Torr

Department of Engineering Science, University of Oxford, Oxford, OX1 3PJ, UK

ARTICLE INFO

Article history:
Received 5 February 2020
Revised 13 June 2020
Accepted 6 September 2020
Available online 14 September 2020

Keywords:
Graph learning
Hypergraph learning
Graph neural networks
Semi-supervised learning

ABSTRACT

Recently, graph neural networks have attracted great attention and achieved prominent performance in various research fields. Most of those algorithms have assumed pairwise relationships of objects of interest. However, in many real applications, the relationships between objects are in higher-order, beyond a pairwise formulation. To efficiently learn deep embeddings on the high-order graph-structured data, we introduce two end-to-end trainable operators to the family of graph neural networks, *i.e.*, hypergraph convolution and hypergraph attention. Whilst hypergraph convolution defines the basic formulation of performing convolution on a hypergraph, hypergraph attention further enhances the capacity of representation learning by leveraging an attention module. With the two operators, a graph neural network is readily extended to a more flexible model and applied to diverse applications where non-pairwise relationships are observed. Extensive experimental results with semi-supervised node classification demonstrate the effectiveness of hypergraph convolution and hypergraph attention.

1. Introduction

In the last decade, Convolution Neural Networks (CNNs) [1] have led to a wide spectrum of breakthrough in various research domains, such as visual recognition [2], speech recognition [3], machine translation [4] *etc*. Due to its innate nature, CNNs hold an extremely strict assumption, that is, input data shall have a regular and grid-like structure. Such a limitation hinders the promotion and application of CNNs to many tasks where data of irregular structures widely exists.

To handle the ubiquitous irregular data structures, there is a growing interest in Graph Neural Networks (GNNs) [5], a methodology for learning deep models with graph data. GNNs have a wide application in social science [6], knowledge graph [7], recommendation system [8], geometrical computation [9], *etc*. And most existing methods assume that the relationships between objects of interest are in pairwise formulations. Specifically in a graph model, it means that each edge only connects two vertices (see Fig. 1(a)).

However, in many real applications, the object relationships are much more complex than pairwise. For instance in recommendation systems, an item may be commented by multiple users. By taking the items as vertices and the rating of users as edges of a graph, each edge may connect more than two vertices. In this case, the affinity relations are no longer dyadic (pairwise), but rather triadic, tetradic or of a higher-order. This brings back the concept of hypergraph [10–13], a special graph model which leverages hyperedges to connect multiple vertices simultaneously (see Fig. 1(b)). Unfortunately, most existing variants of graph neural networks [14,15] are not applicable to the high-order structure encoded by hyperedges.

Although the machine learning and pattern recognition community has witnessed the prominence of graph neural networks in learning patterns on simple graphs, the investigation of deep learning on hypergraphs is still in a very nascent stage. Considering its importance, we propose hypergraph convolution and hypergraph attention in this work, as two strong supplemental operators to graph neural networks. The advantages and contributions of our work are as follows

1) Hypergraph convolution defines a basic convolutional operator in a hypergraph. It enables an efficient information propagation between vertices by fully exploiting the high-order relationship and local clustering structure therein. We mathematically prove that graph convolution is a special case of hypergraph convolution when the non-pairwise relationship degenerates to a pairwise one.
2) Apart from hypergraph convolution where the underlying structure used for propagation is pre-defined, hypergraph attention further exerts an attention mechanism to learn a dynamic connection of hyperedges. Then, the information propagation and gathering is done in task-relevant parts of the graph, thereby generating more discriminative node embeddings.
3) Both hypergraph convolution and hypergraph attention are end-to-end trainable, and can be inserted into most variants of graph neural networks as long as non-pairwise relation-

* Corresponding author.
E-mail addresses: songbai.site@gmail.com (S. Bai), feihu.zhang@eng.ox.ac.uk (F. Zhang), philip.torr@eng.ox.ac.uk (P.H.S. Torr).

https://doi.org/10.1016/j.patcog.2020.107637
0031-3203/

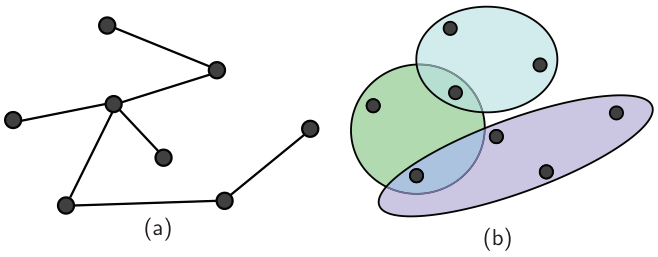

Fig. 1. The difference between a simple graph (a) and a hypergraph (b). In a simple graph, each edge, denoted by a line, only connects two vertices. In a hypergraph, each edge, denoted by a colored ellipse, connects more than two vertices.

ships are observed. Extensive experimental results on benchmark datasets demonstrate the efficacy of the proposed methods for semi-supervised node classification.

The rest of paper is organized as follows. In Section 2, we briefly review some representative methods in graph learning and graph neural networks. In Section 3, we systematically introduce the motivation, formulation and implementation of the proposed method. Experimental evaluations and comparisons are presented in Section 4, followed by the conclusions and future work given in Section 5.

2. Related work

Graphs are a classic kind of data structure [16–18], where its vertices represent objects and its edges linking two adjacent vertices describe the relationship between the corresponding objects.

Graph Neural Network (GNN) is a methodology for learning deep models or embeddings on graph-structured data, which was first proposed by Scarselli et al. [5]. One key aspect in GNN is to define the convolutional operator in the graph domain. Bruna et al. [19] firstly define convolution in the Fourier domain using the graph Laplacian matrix, and generate non-spatially localized filters with potentially intense computations. Henaff et al. [20] enable the spectral filters spatially localized using a parameterization with smooth coefficients. Defferrard et al. [21] focus on the efficiency issue and use a Chebyshev expansion of the graph Laplacian to avoid an explicit use of the graph Fourier basis. Kipf and Welling [22] further simplify the filtering by only using the first-order neighbors and propose Graph Convolutional Network (GCN), which has demonstrated impressive performance in both efficiency and effectiveness with semi-supervised classification tasks.

Meanwhile, some spatial algorithms directly perform convolution on the graph. For instance, Duvenaud et al. [23] learn different parameters for nodes with different degrees, then average the intermediate embeddings over the neighborhood structures. Niepert et al. [24] propose the PATCHY-SAN architecture, which selects a fixed-length sequence of nodes as the receptive field and generate local normalized neighborhood representations for each of the nodes in the sequence. Atwood and Towsley [25] demonstrate that diffusion-based representations can serve as an effective basis for node classification. Zhuang and Ma [26] further explore a joint usage of diffusion and adjacency basis in a dual graph convolutional network. Gilmer et al. [27] defines a unified framework via a message passing function, where each vertex sends messages based on its states and updates the states based on the message of its immediate neighbors. Hamilton et al. [6] propose GraphSAGE, which customizes three aggregating functions, *i.e.*, element-wise mean, long short-term memory and pooling, to learn embeddings in an inductive setting.

Some other works focus on gate mechanism [28], skip connection [29], jumping connection [30], attention mechanism [31], sampling strategy [32,33], hierarchical representation [34], generative models [35,36], adversarial attack [37], *etc*. As a thorough review is simply unfeasible due to the space limitation, we refer interested readers to surveys for more representative methods. For example, Zhang et al. [14] and [15] present two systematical and comprehensive surveys over a series of variants of graph neural networks. Bronstein et al. [9] provide a review of geometric deep learning. Battaglia et al. [38] generalize and extend various approaches and show how graph neural networks can support relational reasoning and combinatorial generalization. Lee et al. [39] particularly focus on the attention models for graphs, and introduce three intuitive taxonomies. Monti et al. [40] propose a unified framework called MoNet, which summarizes Geodesic CNN [41], Anisotropic CNN [42], GCN [22] and Diffusion CNN [25] as its special cases.

As analyzed above, most existing variants of GNN assume pairwise relationships between objects, while our work operates on a high-order hypergraph [43,44] where the between-object relationships are beyond pairwise. Hypergraph learning methods differ in the structure of the hypergraph, *e.g.*, clique expansion and star expansion [45], and the definition of hypergraph Laplacians [46–48]. Following [21,49] propose a hypergraph neural network using a Chebyshev expansion of the graph Laplacian. By analyzing the incident structure of a hypergraph, our work directly defines two differentiable operators, *i.e.*, hypergraph convolution and hypergraph attention, which is intuitive and flexible in learning more discriminative deep embeddings.

3. Proposed approach

In this section, we first give the definition of hypergraph in Section 3.1, then elaborate the proposed hypergraph convolution and hypergraph attention in Section 3.2 and Section 3.3, respectively. At last, Section 3.4 provides a deeper analysis of the properties of our methods.

3.1. Hypergraph revisited

Most existing works [22,31] operate on a simple graph $\mathcal{G} = (V, E)$, where $V = \{v_1, v_2, \ldots, v_N\}$ denotes the vertex set and $E \subseteq V \times V$ denotes the edge set. A graph adjacency matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ is used to reflect the pairwise relationship between every two vertices. The underlying assumption of such a simple graph is that each edge only links two vertices. However, as analyzed above, the relationships between objects are more complex than pairwise in many real applications.

To describe such a complex relationship, a useful graph model is hypergraph, where a hyperedge can connect more than two vertices. Let $\mathcal{G} = (V, E)$ be a hypergraph with N vertices and M hyperedges. Each hyperedge $\epsilon \in E$ is assigned a positive weight $W_{\epsilon\epsilon}$, with all the weights stored in a diagonal matrix $\mathbf{W} \in \mathbb{R}^{M\times M}$. Apart from a simple graph where an adjacency matrix is defined, the hypergraph $\mathcal{G}$ can be represented by an incidence matrix $\mathbf{H} \in \mathbb{R}^{N\times M}$ in general. When the hyperedge $\epsilon \in E$ is incident with a vertex $v_i \in V$, in other words, v_i is connected by ϵ, $H_{i\epsilon} = 1$, otherwise 0. Then, the vertex degree is defined as

$$D_{ii} = \sum_{\epsilon=1}^{M} W_{\epsilon\epsilon} H_{i\epsilon} \tag{1}$$

and the hyperedge degree is defined as

$$B_{\epsilon\epsilon} = \sum_{i=1}^{N} H_{i\epsilon}. \tag{2}$$

Note that $\mathbf{D} \in \mathbb{R}^{N\times N}$ and $\mathbf{B} \in \mathbb{R}^{M\times M}$ are both diagonal matrices.

In the following, we define the operator of convolution on the hypergraph $\mathcal{G}$.

3.2. Hypergraph convolution

The primary obstacle to defining a convolution operator in a hypergraph is to measure the transition probability between two vertices, with which the embeddings (or features) of each vertex can be propagated in a graph neural network. To achieve this, we hold two assumptions: 1) more propagations should be done between those vertices connected by a common hyperedge, and 2) the hyperedges with larger weights deserve more confidence in such a propagation. Then, one step of hypergraph convolution is defined as

$$x_i^{(l+1)} = \sigma\left(\sum_{j=1}^{N}\sum_{\epsilon=1}^{M} H_{i\epsilon}H_{j\epsilon}W_{\epsilon\epsilon}x_j^{(l)}\mathbf{P}\right), \tag{3}$$

where $x_i^{(l)}$ is the embedding of the ith vertex in the (l)th layer. $\sigma(\cdot)$ is a non-linear activation function like LeakyReLU [50] and eLU [51]. $\mathbf{P} \in \mathbb{R}^{F^{(l)}\times F^{(l+1)}}$ is the weight matrix between the (l)th and $(l+1)$th layer. Eq. (3) can be written in a matrix form as

$$\mathbf{X}^{(l+1)} = \sigma\left(\mathbf{H}\mathbf{W}\mathbf{H}^{\mathrm{T}}\mathbf{X}^{(l)}\mathbf{P}\right), \tag{4}$$

where $\mathbf{X}^{(l)} \in \mathbb{R}^{N\times F^{(l)}}$ and $\mathbf{X}^{(l+1)} \in \mathbb{R}^{N\times F^{(l+1)}}$ are the input of the (l)th and $(l+1)$th layer, respectively.

However, $\mathbf{H}\mathbf{W}\mathbf{H}^{\mathrm{T}}$ does not hold a constrained spectral radius, which means that the scale of $\mathbf{X}^{(l)}$ will be possibly changed. In optimizing a neural network, stacking multiple hypergraph convolutional layers like Eq. (4) can then lead to numerical instabilities and increase the risk of exploding/vanishing gradients. Therefore, a proper normalization is necessary. Thus, we impose a symmetric normalization and arrive at our final formulation

$$\mathbf{X}^{(l+1)} = \sigma\left(\mathbf{D}^{-1/2}\mathbf{H}\mathbf{W}\mathbf{B}^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{D}^{-1/2}\mathbf{X}^{(l)}\mathbf{P}\right). \tag{5}$$

Here, we recall that $\mathbf{D}$ and $\mathbf{B}$ are the degree matrices of the vertex and hyperedge in a hypergraph, respectively. It is easy to prove that the maximum eigenvalue of $\mathbf{D}^{-1/2}\mathbf{H}\mathbf{W}\mathbf{B}^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{D}^{-1/2}$ is no larger than 1, which stems from a fact [10] that $\mathbf{I} - \mathbf{D}^{-1/2}\mathbf{H}\mathbf{W}\mathbf{B}^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{D}^{-1/2}$ is a positive semi-definite matrix. $\mathbf{I}$ is an identity matrix of an appropriate size.

Alternatively, a row-normalization is also viable as

$$\mathbf{X}^{(l+1)} = \sigma\left(\mathbf{D}^{-1}\mathbf{H}\mathbf{W}\mathbf{B}^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{X}^{(l)}\mathbf{P}\right), \tag{6}$$

which enjoys similar mathematical properties as Eq. (5), except that the propagation is directional and asymmetric in this case.

As $\mathbf{X}^{(l+1)}$ is differentiable with respect to $\mathbf{X}^{(l)}$ and $\mathbf{P}$, we can use hypergraph convolution in model training and optimize it via gradient descent.

3.3. Hypergraph attention

Hypergraph convolution has a sort of innate attentional mechanism [31,39]. As we can find from Eqs. (5) and (6), the transition probability between vertices is non-binary, which means that for a given vertex, the afferent and efferent information flow is explicitly assigned a diverse magnitude of importance. However, such an attentional mechanism is not learnable and trainable after the graph structure (the incidence matrix $\mathbf{H}$) is given. The goal of hypergraph attention is to learn a dynamic incidence matrix, thereby a dynamic transition matrix that can better reveal the intrinsic relationship between vertices.

One natural solution is to exert an attention learning module on $\mathbf{H}$. In this circumstance, instead of treating each vertex as being connected by a certain hyperedge or not, the attention module presents a probabilistic model, which assigns non-binary and real values to measure the degree of connectivity. We expect that the probabilistic model can learn more category-discriminative embeddings and the relationship between vertices can be more accurately described.

Nevertheless, hypergraph attention is only feasible when the vertex set and the hyperedge set are from (or can be projected to) the same homogeneous domain, since only in this case, the similarities between vertices and hyperedges are directly comparable. In practice, it depends on how the hypergraph $\mathcal{G}$ is constructed. For example, Huang et al. [52] apply hypergraph learning to image retrieval where each vertex collects its k-nearest neighbors to form a hyperedge, as also the way of constructing hypergraphs in our experiments. When the vertex set and the edge set are comparable, we define the procedure of hypergraph attention inspired by Velickovic et al. [31]. For a given vertex x_i and its associated hyperedge x_j, the attentional score is

$$H_{ij} = \frac{\exp\left(\sigma\left(\mathrm{sim}(x_i\mathbf{P}, x_j\mathbf{P})\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\sigma\left(\mathrm{sim}(x_i\mathbf{P}, x_k\mathbf{P})\right)\right)}, \tag{7}$$

where $\sigma(\cdot)$ is a non-linear activation function. $\mathcal{N}_i$ is the neighborhood set of x_i, which can be pre-accessed on some benchmarks, such as the Cora and Citeseer datasets [53]. $\mathrm{sim}(\cdot)$ is a similarity function that computes the pairwise similarity between two vertices, defined as

$$\mathrm{sim}(x_i, x_j) = \mathbf{a}^{\mathrm{T}}[x_i \| x_j]. \tag{8}$$

Here $[,\|,]$ denotes concatenation and $\mathbf{a}$ is a weight vector used to output a scalar similarity value.

With the incidence matrix $\mathbf{H}$ enriched by an attention module, one can also follow Eqs. (5) and (6) to learn the intermediate embedding of vertices layer-by-layer. Note that hypergraph attention also propagates gradients to $\mathbf{H}$ in addition to $\mathbf{X}^{(l)}$ and $\mathbf{P}$.

In some applications, the vertex set and the hyperedge set are from two heterogeneous domains. For instance, Zhou et al. [48] assume that attributes are hyperedges to connect objects like newspaper or text. Then, it is problematic to directly learn an attention module over the incidence matrix $\mathbf{H}$. We leave this issue for future work.

3.4. Summary and remarks

The pipeline of the proposed hypergraph convolution and hypergraph attention is illustrated in Fig. 2. Both two operators can be inserted into most variants of graph neural networks when non-pairwise relationships are observed, and used for model training.

As the only difference between hypergraph convolution and hypergraph attention is an optional attention module on the incidence matrix $\mathbf{H}$, below we take hypergraph convolution as a representative to further analyze the properties of our methods. Note that the analyses also hold for hypergraph attention.

Relationship with Graph Convolution. We prove that graph convolution [22] is a special case of hypergraph convolution mathematically.

Let $\mathbf{A} \in \mathbb{R}^{N\times N}$ be the adjacency matrix used in graph convolution. When each edge only links two vertices in a hypergraph, the vertex degree matrix $\mathbf{B} = 2\mathbf{I}$. Assuming equal weights for all the hyperedges (*i.e.*, $\mathbf{W} = \mathbf{I}$), we have an interesting observation of hypergraph convolution. Based on Eq. (5), the definition of hypergraph convolution then becomes

$$\begin{aligned}\mathbf{X}^{(l+1)} &= \sigma\left(\frac{1}{2}\mathbf{D}^{-1/2}\mathbf{H}\mathbf{H}^{\mathrm{T}}\mathbf{D}^{-1/2}\mathbf{X}^{(l)}\mathbf{P}\right),\\ &= \sigma\left(\frac{1}{2}\mathbf{D}^{-1/2}(\mathbf{A}+\mathbf{D})\mathbf{D}^{-1/2}\mathbf{X}^{(l)}\mathbf{P}\right)\\ &= \sigma\left(\frac{1}{2}(\mathbf{I}+\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2})\mathbf{X}^{(l)}\mathbf{P}\right)\\ &= \sigma\left(\hat{\mathbf{A}}\mathbf{X}^{(l)}\mathbf{P}\right),\end{aligned} \tag{9}$$

where $\hat{\mathbf{A}} = 1/2\tilde{\mathbf{A}}$ and $\tilde{\mathbf{A}} = \mathbf{I} + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$. As we can see, Eq. (9) is exactly equivalent to the definition of graph convolution (see

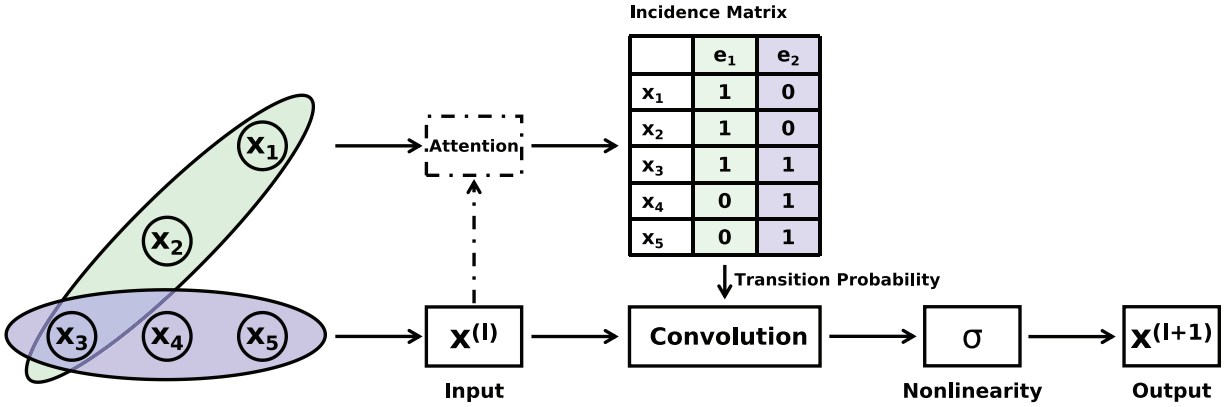

Fig. 2. Schematic illustration of hypergraph convolution with 5 vertices and 2 hyperedges. With an optional attention mechanism, hypergraph convolution upgrades to hypergraph attention.

Eq. (9) in [22]). Note that $\widetilde{\mathbf{A}}$ has eigenvalues in the range [0,2]. To avoid scale changes, [22] have suggested a re-normalization trick, that is

$$\hat{\mathbf{A}} = \widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1/2}. \tag{10}$$

In the specific case of hypergraph convolution, we are using a simplified solution, that is dividing $\widetilde{\mathbf{A}}$ by 2.

With GCN as a bridge and springboard to the family of graph neural networks, it then becomes feasible to build connections with other frameworks, *e.g.*, MoNet [40], and develop the higher-order counterparts of those variants to deal with non-pairwise relationships.

Implementation in Practice and Complexity Analysis. The implementation of hypergraph convolution appears sophisticated as 6 matrices are multiplied for symmetric convolution (see Eq. (5)) and 5 matrices are multiplied for asymmetric convolution (see Eq. (6)). However, it should be mentioned that $\mathbf{D}$, $\mathbf{W}$ and $\mathbf{B}$ are all diagonal matrices, which makes it possible to efficiently implement it in commonly used deep learning platforms.

For asymmetric convolution, we have from Eq. (6) that

$$\mathbf{D}^{-1}\mathbf{H}\mathbf{W}\mathbf{B}^{-1}\mathbf{H}^{\mathrm{T}} = (\mathbf{D}^{-1}\mathbf{H}\mathbf{W})(\mathbf{H}\mathbf{B}^{-1})^{\mathrm{T}}, \tag{11}$$

where $\mathbf{D}^{-1}\mathbf{H}\mathbf{W}$ performs L_1 normalization of $\mathbf{H}\mathbf{W}$ over rows and $\mathbf{H}\mathbf{B}^{-1}$ performs L_1 normalization of $\mathbf{H}$ over columns. In space-saving applications where matrix-form variables are allowed, normalization can be simply done using standard built-in functions in public neural network packages.

In case of space-consuming applications, one can readily implement a sparse version of hypergraph convolution as well. Since $\mathbf{H}$ is usually a sparse matrix, Eq. (11) does not necessarily decrease the sparsity too much. Hence, we can conduct normalization only on non-zero indices of $\mathbf{H}$, resulting in a sparse transition matrix.

Mathematically, if we implement Eq. (11) directly via matrix multiplications, the time complexity would be upper bounded by $O(2N^2M + 2M^2N)$. In comparison, if implemented via normalization, the time complexity is significantly decreased. In more detail, $\mathbf{D}^{-1}\mathbf{H}\mathbf{W}$ requires $2NM$ operations, $\mathbf{H}\mathbf{B}^{-1}$ requires NM operations, and the matrix multiplication between them requires N^2M operations. As a result, the total operations required are bounded by $O(3NM + N^2M)$. It can be envisioned that the speed can be even faster if the matrix sparsity is considered. Because of the well-optimized normalization functions in commonly used computing platforms (*e.g.,* `torch.nn.functional.normalize` in Pytorch or `sklearn.preprocessing.normalize` in scikit-learn), it only takes 1.8ms to fulfill one forward pass on the Cora dataset using a server with an Intel(R) Core(TM) i7-5960X CPU (3.00GHz) and an NVIDIA GeForce RTX 2080 Ti GPU.

Symmetric hypergraph convolution defined in Eq. (5) can be implemented similarly, with a minor difference in normalization using the vertex degree matrix $\mathbf{D}$.

Skip Connection. Hypergraph convolution can be integrated with skip connection [2] as

$$\mathbf{X}_k^{(l+1)} = \mathrm{HConv}\big(\mathbf{X}^{(l)}, \mathbf{H}_k, \mathbf{P}_k\big) + \mathbf{X}^{(l)}, \tag{12}$$

where $\mathrm{HConv}(\cdot)$ represents the hypergraph convolution operator defined in Eq. (5) (or Eq. (6)). Some similar structures (*e.g.*, highway structure [54] adopted in Highway-GCN [55]) can be also applied.

It has been demonstrated [22] that deep graph models cannot improve the performance even with skip connections since the receptive field grows exponentially with respect to the model depth. In the experiments, we will verify the compatibility of the proposed operators with skip connections in model training.

Multi-head. To stabilize the learning process and improve the representative power of networks, multi-head (*a.k.a.* multi-branch) architecture is suggested in relevant works, *e.g.*, [31,56,57]. hypergraph convolution can be also extended in that way, as

$$\begin{aligned}\mathbf{X}_k^{(l+1)} &= \mathrm{HConv}\big(\mathbf{X}^{(l)}, \mathbf{H}_k, \mathbf{P}_k\big),\\ \mathbf{X}^{(l+1)} &= \mathrm{Aggregate}\big(\mathbf{X}_k^{(l+1)}\big)_{k=1}^{K},\end{aligned} \tag{13}$$

where $\mathrm{Aggregate}(\cdot)$ is a certain aggregation like concatenation or average pooling. $\mathbf{H}_k$ and $\mathbf{P}_k$ are the incidence matrix and weight matrix corresponding to the kth head, respectively. Note that only in hypergraph attention, $\mathbf{H}_k$ is different over different heads.

4. Experiments

In this section, we evaluate the proposed hypergraph convolution and hypergraph attention in the task of semi-supervised node classification.

Following [22,31], we first employ three citation network datasets, including the Cora, Citeseer and Pubmed datasets [53], to make a fair comparison with previous methods.

- The Cora dataset contains 2,708 scientific publications divided into 7 categories. There are 5,429 edges in total, with each edge being a citation link from one article to another. Each publication is described by a binary bag-of-word representation, where 0 (or 1) indicates the absence (or presence) of the corresponding word from the dictionary. The dictionary consists of 1,433 unique words.
- Like the Cora dataset, the Citeseer dataset contains 3,327 scientific publications, divided into 6 categories and linked by 4,732 edges. Each publication is described by a binary bag-of-word representation of 3,703 dimensions.

Table 1
Overview of data statistics.

Dataset	#Nodes	#Edges	#Features	#Classes
Cora	2708	5429	1433	7
Citeseer	3327	4732	3703	6
Pubmed	19,717	44,338	500	3
20-newsgroup	16,242	-	100	4

- The Pubmed dataset is comprised of 19,717 scientific publications divided into 3 classes. The citation network has 44,338 links. Each publication is described by a vectorial representation using Term Frequency-Inverse Document Frequency (TF-IDF), drawn from a dictionary with 500 terms.

Then as per [48], a modified version of the 20-newsgroup dataset[1] with binary occurrence values for 100 words is used for text categorization. It consists of 16,242 articles divided into 4 groups, with the sizes being 4605, 3519, 2657 and 5461 respectively. Each word naturally connects multiple postings, which makes this dataset more suitable for the use of a hypergraph. In the meantime, it means the graph used is generated using the same resources (words) as the node features. Table 1 presents an overview of the dataset statistics.

As for the training-testing data split, we adopt the setting used in [58]. In each dataset, 20 nodes per category are used for model training. Another 500 nodes are used for validation purposes and 1000 nodes are used for performance evaluation.

4.1. Experiments on citation networks

4.1.1. Hypergraph construction

Most existing methods interpret the citation network as the adjacency matrix of a simple graph by a certain kind of normalization, *e.g.*, [22]. By doing so, these methods working on a simple graph aim to define the scheme of message passing if one article is cited by another article.

In this work, we construct a higher-order graph to enable hypergraph convolution and hypergraph attention. The whole procedure is divided into three steps: 1) all the articles constitute the vertex set of the hypergraph; 2) each article is taken as a centroid and forms a hyperedge to connect those articles which have citation links to it (either citing it or being cited); 3) the hyperedges are equally weighted for simplicity, but one can set non-equal weights to encode a prior knowledge if existing in other applications. Compared with existing methods, our work focuses on the message passing if two articles are both cited by a third article.

4.1.2. Implementation details

We implement the proposed hypergraph convolution and hypergraph attention using Pytorch[2]. As for the parameter setting and network structure, we closely follow [31] without a carefully parameter tuning and model design.

In more detail, a two-layer graph model is constructed. The first layer consists of 8 branches of the same topology, and each branch generates an 8-dimensional hidden representation. The second layer, used for classification, is a single-branch topology and generates C-dimensional feature (C is the number of classes). Each layer is followed by a nonlinearity activation and here we use Exponential Linear Unit (ELU) [51]. L_2 regularization is applied to the parameters of network with $\lambda = 0.0003$ on the Cora and Citeseer datasets and $\lambda = 0.001$ on the Pubmed dataset, respectively.

[1] https://cs.nyu.edu/~roweis/data.html.

[2] Included in the PyTorch Geometric Library: https://github.com/rusty1s/pytorch_geometric.

Table 2
The comparison with baseline methods in terms of classification accuracy (%). "Hyper-Conv." denotes hypergraph convolution and "Hyper-Atten." denotes hypergraph attention.

Method	Cora dataset	Citeseer dataset
GCN*	81.80	70.29
Hyper-Conv. (**ours**)	82.19	**70.35**
GCN*+Hyper-Conv. (**ours**)	**82.63**	70.00
GAT*	82.43	70.02
Hyper-Atten. (**ours**)	82.61	**70.88**
GAT*+Hyper-Atten. (**ours**)	**82.74**	70.12

Specifically in hypergraph attention, dropout with a rate of 0.6 is applied to both inputs of each layer and the attention transition matrix. As for the computation of the attention incidence matrix $\mathbf{H}$ in Eq. (7), we employ a linear transform as the similarity function $\text{sim}(\cdot)$, followed by LeakyReLU nonlinearity [50] with the negative input slope set to 0.2. On the Pubmed dataset, we do not use 8 output attention heads for classification to ensure the consistency of network structures.

We train the model by minimizing the cross-entropy loss on the training nodes using the Adam [59] optimizer with a learning rate of 0.005 on the Cora and Citeseer datasets and 0.01 on the Pubmed dataset, respectively. An early stop strategy is adopted on the validation loss with a patience of 100 epochs. For all the experiments, we report the mean classification accuracy and standard deviation of 100 trials on the testing dataset.

4.1.3. Analysis

We first analyze the properties of hypergraph convolution and hypergraph attention with a series of ablation studies. The comparison is primarily done with Graph Convolution Network (GCN) [22] and Graph Attention Network (GAT) [31], which are two latest representatives of graph neural networks that have close relationships with our methods.

For a fair comparison, we reproduce the performance of GCN and GAT with *exactly* the same experimental setting aforementioned. Thus, we denote them by GCN* and GAT* in the following. Moreover, we employ the same normalization strategy as GCN, *i.e.*, symmetric normalization in Eq. (5) for hypergraph convolution, and the same strategy as GAT, *i.e.*, asymmetric normalization in Eq. (6) for hypergraph attention. They are denoted by Hyper-Conv. and Hyper-Atten. for short, respectively.

We modify the model of GAT to implement GCN by removing the attention module and directly feeding the graph adjacency matrix with the normalization trick proposed in GCN. Two noteworthy comments are made here. First, although the architecture of GCN* differs from the original one, the principle of performing graph convolution is the same. Second, directly feeding the graph adjacency matrix is not equivalent to the constant attention described in GAT as normalization is used in our case. In GAT, the constant attention weight is set to 1 without normalization.

Comparisons with Baselines. The comparison with baseline methods is given in Table 2.

We first observe that hypergraph convolution and hypergraph attention, as **non-pairwise models**, consistently outperform its corresponding **pairwise models**, *i.e.*, graph convolution network (GCN*) and graph attention network (GAT*). For example on the Cora dataset, GCN* achieves an accuracy of 81.80 and hypergraph convolution reports an accuracy of 82.19. We conduct onesided two-sample t-test and obtain p-value equal to 0.016, indicating the improvement is statistically significant at the 5% significance level. On the Citeseer dataset, hypergraph attention achieves a classification accuracy of 70.88, an improvement of 0.86 over GAT*. This demonstrates the benefit of considering higher-order models in

Table 3
The compatibility of skip connection in terms of accuracy (%).

Method	λ=3e-4		λ=1e-3	
	Cora	Citeseer	Cora	Citeseer
GCN*	79.96	69.24	80.52	70.15
Hyper-Conv. (**ours**)	82.22	69.46	82.66	70.83
GAT*	80.84	68.96	81.33	69.69
Hyper-Atten. (**ours**)	**81.85**	**70.37**	**82.34**	**71.19**

Table 4
The influence of the length of hidden representation on Cora.

Method	The length of hidden representation					
	2	4	8	16	24	36
GCN*	65.9	79.6	**82.0**	81.9	82.0	81.9
Hyper-Conv.	**69.7**	**80.4**	**82.0**	**82.1**	**82.1**	**82.1**

graph neural networks to exploit non-pairwise relationships and local clustering structure parameterized by hyperedges.

Compared with hypergraph convolution, hypergraph attention adopts a data-driven learning module to dynamically estimate the strength of each link associated with vertices and hyperedges. Thus, the attention mechanism helps hypergraph convolution embed the non-pairwise relationships between objects more accurately. As presented in Table 2, the performance improvements brought by hypergraph attention are 0.42 and 0.53 over hypergraph convolution on the Cora and Citeseer datasets, respectively.

Although non-pairwise models proposed in this work have achieved improvements over pairwise models, one cannot hastily deduce that non-pairwise models are more capable in learning robust deep embeddings under all circumstances. A rational claim is that they are suitable for different applications as real data may convey different structures. Some graph-structured data can be only modeled in a simple graph, some can be only modeled in a higher-order graph and others are suitable for both. Nevertheless, as analyzed in Section 3.4, our method presents a more flexible operator in graph neural networks, where graph convolution and graph attention are special cases of non-pairwise models with guaranteed mathematical properties and performance.

One may be also interested in another question, *i.e.*, does it bring performance improvements if using hypergraph convolution (or attention) in conjunction with graph convolution (or attention)? We further investigate this by averaging the transition probability learned by non-pairwise models and pairwise models with equal weights, and report the results in Table 2. As it shows, a positive synergy is only observed on the Cora dataset, where the best results of convolution operator and attention operator are improved to 82.63 and 82.74, respectively. By contrast, our methods encounter a slight performance decrease on the Citeseer dataset. From another perspective, it also supports our above claim that different data may fit different structural graph models.

Analysis of Skip Connection. We study the influence of skip connection [2] by adding an identity mapping in the first hypergraph convolution layer. We report in Table 3 two settings of the weight decay, *i.e.*, λ=3e-4 (default setting) and λ=1e-3.

As it shows, both GCN* and GAT* report lower recognition rates when integrated with skip connection compared with those reported in Table 2. In comparison, the proposed non-pairwise models, especially hypergraph convolution, seem to benefit from skip connection. For instance, the best-performing trial of hypergraph convolution yields 82.66 on the Cora dataset, better than 82.19 achieved without skip connection, and yields 70.83 on the Citeseer dataset, better than 70.35 achieved without skip connection.

Such experimental results encourage us to further train a much deeper model implemented with hypergraph convolution or hypergraph attention (say up to 10 layers). However, we also witness a performance deterioration either with or without skip connection. This reveals that a better training paradigm and architecture are still urgently required for graph neural networks.

Analysis of Hidden Representation. Table 4 presents the performance comparison between GCN* and hypergraph convolution with an increasing length of the hidden representation. The number of heads is set to 1.

It is easy to find that the performance keeps increasing with an increase of the length of the hidden representation, then peaks when the length is 16. Moreover, hypergraph convolution consistently beats GCN* with a variety of feature dimensions. As the only difference between GCN* and hypergraph convolution is the used graph structure, the performance gain purely comes from a more robust way of establishing the relationships between objects. It firmly demonstrates the ability of our methods in graph knowledge embedding.

4.1.4. Comparison with state-of-the-art

We compare our method with the state-of-the-art algorithms, which have followed the experimental setting in [58] and reported classification accuracies on the Cora, Citeseer and Pubmed datasets. Note that the results are directly quoted from the original papers, instead of being re-implemented in this work. Besides GCN and GAT, the selected algorithms also include Manifold Regularization [60], Semi-supervised Embedding [61], Label Propagation [62], DeepWalk [63], Iterative Classification Algorithm [64], Planetoid [58], Chebyshev [21], MoNet [40], and Variance Reduction [33].

As presented in Table 5, our method achieves the second best performance on the Cora and Citeseer datasets, which is slightly inferior to GAT [31] by 0.3 and 1.3, respectively. The performance gap is attributed to multiple factors, such as the difference in deep learning platforms and better parameter tuning. As shown in Section 4.1.3, thorough experimental comparisons under the same setting have demonstrated the benefit of learning deep embeddings using the proposed non-pairwise models. Nevertheless, we emphasize again that pairwise and non-pairwise models have different application scenarios, and existing variants of graph neural networks can be easily extended to their non-pairwise counterparts with the proposed two operators.

On the Pubmed dataset, hypergraph attention reports a classification accuracy of 78.4, better than 78.1 achieved by GAT*. As described in Section 4.1.2, the original implementation of GAT adopts 8 output attention heads while only 1 is used in GAT* to ensure the consistency of the model architecture. Even though, hypergraph attention also achieves a comparable performance with the state-of-the-art methods.

4.2. Experiments on text categorization

In the experiments presented in Section 4.1, we assume that a pre-defined graph structure, independent from the input features, is given. For example, the citation links presented in the citation network are irrelevant to the feature embeddings of articles. Here, we show the scenario where graph structure and input features are both generated using the same resources, *i.e.*, attributes, which is more suitable for the use of a hypergraph.

As described above, the 20-newsgroup dataset adopted in our work does not have a pre-defined graph structure. To enable a direct comparison between GCN and hypergraph convolution feasible, we first need to construct the graph. For the graph construction of GCN, we connect two vertices if there is at least one shared

Table 5
Comparison with the state-of-the-art methods in terms of classification accuracy (%). The best and second best results are marked in bold and italic, respectively.

Method	Cora	Citeseer	Pubmed
Multilayer Perceptron	55.1	46.5	71.4
Manifold Regularization [60]	59.5	60.1	70.7
Semi-supervised Embedding [61]	59.0	59.6	71.7
Label Propagation [62]	68.0	45.3	63.0
DeepWalk [63]	67.2	43.2	65.3
Iterative Classification Algorithm [64]	75.1	69.1	73.9
Planetoid [58]	75.7	64.7	77.2
Chebyshev [21]	81.2	69.8	74.4
Graph Convolutional Network [22]	81.5	70.3	**79.0**
Feng et al. [49]	81.6	-	-
MoNet [40]	81.7	-	***78.8***
Variance Reduction [33]	82.0	70.9	**79.0**
Graph Attention Network [31]	**83.0**	**72.5**	**79.0**
Ours	***82.7 ± 0.3***	***71.2 ± 0.4***	78.4 ± 0.3

word. In this case, we do not necessarily distinguish how many words they share. However, it is quite likely that more than two articles share the same word, thus naturally forming a hypergraph structure. Hence, we use words directly as hyperedges to connect articles for evaluating hypergraph convolution.

As for the implementation, we use the same model architecture, the same training strategy, the same loss function and the same set of hyper-parameters as those in Section 4.1. That means, except for the way of graph construction, all the rest settings remain the same. Without bells and whistles, our hypergraph convolution reports an accuracy of 61.7 while GCN reports an accuracy of 57.0. The performance gain 4.7 solely comes from a better way of modeling the occurrence of words, which suggests the capability of our method in handling non-pairwise structures.

5. Conclusion

In this work, we have contributed two end-to-end trainable operators to the family of graph neural networks, *i.e.*, hypergraph convolution and hypergraph attention. While most variants of graph neural networks assume pairwise relationships of objects of interest, the proposed operators handle non-pairwise relationships modeled in a high-order hypergraph. We theoretically demonstrate that some recent representative works, *e.g.*, graph convolution network [22] and graph attention network [31], are special cases of our methods. Hence, our proposed hypergraph convolution and hypergraph attention are more flexible in dealing with arbitrary orders of relationships and diverse applications, where both pairwise and non-pairwise formulations are likely to be involved. Thorough experimental results with semi-supervised node classification demonstrate the efficacy of the proposed methods.

In real applications, the key to applying our method is to abstract the non-pairwise relationships (for defining hyperedges) from the data. For example, i) in retrieval systems, a keyword, being as a hyperedge, could connect more than two webpages; ii) in collaborative filtering, a product tag, being as a hyperedge, could connect more than two users; iii) in social networks, two individuals could be connected by their direct friendship, which is a pairwise relationship in a vanilla graph. However, if a post is commented on by multiple individuals, we could use the post as a hyperedge to connect those individuals. Besides, our method does not distinguish undirected hypergraphs from directed hypergraphs. That means it is capable of handling both cases since the incident structure can be determined similarly. The last remark is about the scalability of our method. We have presented how to efficiently implement our method in Section 3.4 to accommodate popular deep learning platforms. When dealing with billion or trillion scale data, one may resort to distributed systems like MapReduce for the sake of commercial use.

There are still some challenging directions that can be further investigated. Some of them are inherited from the limitation of graph neural networks, such as training substantially deeper models with more than a hundred of layers [2], handling dynamic structures [14,15], batch-wise model training, *etc*. Meanwhile, some issues are directly related to the proposed methods in high-order learning. For example, although hyperedges are equally weighted in our experiments, it is promising to exploit a proper weight mechanism when extra knowledge of data distributions is accessible, and even, adopt a learnable module in a neural network then optimize the weight with gradient descent. The current implementation of hypergraph attention cannot be executed when the vertex set and the hyperedge set are from two heterogeneous domains. One possible solution is to learn a joint embedding to project the vertex features and edge features [65,66] in a shared latent space, which requires further exploration.

Moreover, it is also interesting to plug hypergraph convolution and hypergraph attention into other variants of graph neural network, *e.g.*, MoNet [40], GraphSAGE [6] and GCPN [67], and apply them to other domain-specific applications, *e.g.*, 3D shape analysis [42,68,69], visual search [70], visual question answering [71], chemistry [27,72], knowledge graphs [73], matrix factorization [74] and NP-hard problems [75].

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by EPSRC grant Seebibyte EP/M013774/1 and EPSRC/MURI grant EP/N019474/1.

References

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, ImageNet classification with deep convolutional neural networks, in: NeurIPS, 2012, pp. 1097–1105.
[2] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: CVPR, 2016, pp. 770–778.
[3] G. Hinton, L. Deng, D. Yu, G.E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, et al., Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups, IEEE Signal Process. Mag. 29 (6) (2012) 82–97.
[4] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, ICLR, 2015.
[5] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Netw. 20 (1) (2009) 61–80.

[6] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: NeurIPS, 2017, pp. 1024–1034.
[7] X. Wang, Y. Ye, A. Gupta, Zero-shot recognition via semantic embeddings and knowledge graphs, in: CVPR, 2018, pp. 6857–6866.
[8] R. Ying, R. He, K. Chen, P. Eksombatchai, W.L. Hamilton, J. Leskovec, Graph convolutional neural networks for web-scale recommender systems, KDD, 2018.
[9] M.M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, P. Vandergheynst, Geometric deep learning: going beyond euclidean data, IEEE Signal Process. Mag. 34 (4) (2017) 18–42.
[10] S. Agarwal, K. Branson, S. Belongie, Higher order learning with graphs, in: ICML, 2006, pp. 17–24.
[11] C. Zhang, S. Hu, Z.G. Tang, T. Chan, Re-revisiting learning on hypergraphs: confidence interval and subgradient method, in: ICML, 2017, pp. 4026–4034.
[12] J. Yu, D. Tao, M. Wang, Adaptive hypergraph learning and its application in image classification, TIP 21 (7) (2012) 3262–3272.
[13] D.C.G. Pedronette, L.P. Valem, J. Almeida, R.d.S. Torres, Multimedia retrieval through unsupervised hypergraph-based manifold ranking, TIP 28 (12) (2019) 5824–5838.
[14] Z. Zhang, P. Cui, W. Zhu, Deep learning on graphs: a survey, TKDE (2020).
[15] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, M. Sun, Graph neural networks: a review of methods and applications, arXiv:1812.08434(2018).
[16] X. Yang, L. Prasad, L.J. Latecki, Affinity learning with diffusion on tensor product graph, TPAMI 35 (1) (2012) 28–38.
[17] D.C.G. Pedronette, F.M.F. Gonçalves, I.R. Guilherme, Unsupervised manifold learning through reciprocal kNN graph and connected components for image retrieval tasks, Pattern Recognit. 75 (2018) 161–174.
[18] D.C.G. Pedronette, R.D.S. Torres, Image re-ranking and rank aggregation based on similarity of ranked lists, Pattern Recognit. 46 (8) (2013) 2350–2360.
[19] J. Bruna, W. Zaremba, A. Szlam, Y. LeCun, Spectral networks and locally connected networks on graphs, ICLR, 2014.
[20] M. Henaff, J. Bruna, Y. LeCun, Deep convolutional networks on graph-structured data, arXiv:1506.05163(2015).
[21] M. Defferrard, X. Bresson, P. Vandergheynst, Convolutional neural networks on graphs with fast localized spectral filtering, in: NeurIPS, 2016, pp. 3844–3852.
[22] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, ICLR, 2017.
[23] D.K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, R.P. Adams, Convolutional networks on graphs for learning molecular fingerprints, in: NeurIPS, 2015, pp. 2224–2232.
[24] M. Niepert, M. Ahmed, K. Kutzkov, Learning convolutional neural networks for graphs, in: ICML, 2016, pp. 2014–2023.
[25] J. Atwood, D. Towsley, Diffusion-convolutional neural networks, in: NeurIPS, 2016, pp. 1993–2001.
[26] C. Zhuang, Q. Ma, Dual graph convolutional networks for graph-based semi–supervised classification, in: World Wide Web, 2018, pp. 499–508.
[27] J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, G.E. Dahl, Neural message passing for quantum chemistry, ICML, 2017.
[28] Y. Li, D. Tarlow, M. Brockschmidt, R. Zemel, Gated graph sequence neural networks, ICLR, 2016.
[29] T. Pham, T. Tran, D.Q. Phung, S. Venkatesh, Column networks for collective classification., in: AAAI, 2017, pp. 2485–2491.
[30] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, S. Jegelka, Representation learning on graphs with jumping knowledge networks, ICML, 2018.
[31] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio, Graph attention networks, ICLR, 2018.
[32] J. Chen, T. Ma, C. Xiao, FastGCN: fast learning with graph convolutional networks via importance sampling, ICLR, 2018.
[33] J. Chen, J. Zhu, L. Song, Stochastic training of graph convolutional networks with variance reduction., in: ICML, 2018, pp. 941–949.
[34] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, J. Leskovec, Hierarchical graph representation learning with differentiable pooling, in: NeurIPS, 2018, pp. 4805–4815.
[35] J. You, R. Ying, X. Ren, W. Hamilton, J. Leskovec, GraphRNN: generating realistic graphs with deep auto-regressive models, in: ICML, 2018, pp. 5694–5703.
[36] A. Bojchevski, O. Shchur, D. Zügner, S. Günnemann, NetGAN: generating graphs via random walks, ICML, 2018.
[37] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, L. Song, Adversarial attack on graph structured data, ICML, 2018.
[38] P.W. Battaglia, J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al., Relational inductive biases, deep learning, and graph networks, arXiv:1806.01261(2018).
[39] J.B. Lee, R.A. Rossi, S. Kim, N.K. Ahmed, E. Koh, Attention models in graphs: a survey, TKDD 13 (6) (2019) 1–25.
[40] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, M.M. Bronstein, Geometric deep learning on graphs and manifolds using mixture model CNNs, CVPR, 2017.
[41] J. Masci, D. Boscaini, M. Bronstein, P. Vandergheynst, Geodesic convolutional neural networks on Riemannian manifolds, in: ICCVW, 2015, pp. 37–45.
[42] D. Boscaini, J. Masci, E. Rodolà, M. Bronstein, Learning shape correspondence with anisotropic convolutional neural networks, in: NeurIPS, 2016, pp. 3189–3197.
[43] C. Berge, Graphs and hypergraphs (1973).
[44] P. Li, O. Milenkovic, Inhomogeneous hypergraph clustering with applications, in: NeurIPS, 2017, pp. 2308–2318.
[45] J.Y. Zien, M.D. Schlag, P.K. Chan, Multilevel spectral hypergraph partitioning with arbitrary vertex sizes, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 18 (9) (1999) 1389–1399.
[46] M. Bolla, Spectra, euclidean representations and clusterings of hypergraphs, Discrete Math. 117 (1–3) (1993) 19–39.
[47] J.A. Rodriguez, On the Laplacian spectrum and walk-regular hypergraphs, Linear Multilinear Algebra 51 (3) (2003) 285–297.
[48] D. Zhou, J. Huang, B. Schölkopf, Learning with hypergraphs: clustering, classification, and embedding, in: NeurIPS, 2007, pp. 1601–1608.
[49] Y. Feng, H. You, Z. Zhang, R. Ji, Y. Gao, Hypergraph neural networks, AAAI, 2019.
[50] A.L. Maas, A.Y. Hannun, A.Y. Ng, Rectifier nonlinearities improve neural network acoustic models, ICML, 2013.
[51] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (ELUs), ICLR, 2016.
[52] Y. Huang, Q. Liu, S. Zhang, D.N. Metaxas, Image retrieval via probabilistic hypergraph ranking, in: CVPR, 2010, pp. 3376–3383.
[53] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, T. Eliassi-Rad, Collective classification in network data, AI Mag. 29 (3) (2008) 93.
[54] R.K. Srivastava, K. Greff, J. Schmidhuber, Highway networks, arXiv:1505.00387(2015).
[55] A. Rahimi, T. Cohn, T. Baldwin, Semi-supervised user geolocation via graph convolutional networks, ACL, 2018.
[56] S. Xie, R. Girshick, P. Dollár, Z. Tu, K. He, Aggregated residual transformations for deep neural networks, in: CVPR, 2017, pp. 5987–5995.
[57] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: NeurIPS, 2017, pp. 5998–6008.
[58] Z. Yang, W.W. Cohen, R. Salakhutdinov, Revisiting semi-supervised learning with graph embeddings, ICML, 2016.
[59] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, ICLR, 2015.
[60] M. Belkin, P. Niyogi, V. Sindhwani, Manifold regularization: a geometric framework for learning from labeled and unlabeled examples, J. Mach. Learn. Res. 7 (2006) 2399–2434.
[61] J. Weston, F. Ratle, H. Mobahi, R. Collobert, Deep Learning via Semi-supervised Embedding, in: Neural Networks: Tricks of the Trade, 2012, pp. 639–655.
[62] X. Zhu, Z. Ghahramani, J.D. Lafferty, Semi-supervised learning using gaussian fields and harmonic functions, in: ICML, 2003, pp. 912–919.
[63] B. Perozzi, R. Al-Rfou, S. Skiena, DeepWalk: online learning of social representations, in: KDD, 2014, pp. 701–710.
[64] Q. Lu, L. Getoor, Link-based classification, in: ICML, 2003, pp. 496–503.
[65] M. Simonovsky, N. Komodakis, Dynamic edge-conditioned filters in convolutional neural networks on graphs, CVPR, 2017.
[66] M. Schlichtkrull, T.N. Kipf, P. Bloem, R. van den Berg, I. Titov, M. Welling, Modeling relational data with graph convolutional networks, in: European Semantic Web Conference, 2018, pp. 593–607.
[67] J. You, B. Liu, R. Ying, V. Pande, J. Leskovec, Graph convolutional policy network for goal-directed molecular graph generation, NeurIPS, 2018.
[68] B. Xiao, E.R. Hancock, R.C. Wilson, Graph characteristics from the heat kernel trace, Pattern Recognit. 42 (11) (2009) 2589–2606.
[69] B. Xiao, S. Yi-Zhe, P. Hall, Learning invariant structure for object identification by using graph methods, CVIU 115 (7) (2011) 1023–1031.
[70] L. Zhou, X. Bai, X. Liu, J. Zhou, E.R. Hancock, Learning binary code for fast nearest subspace search, Pattern Recognit. 98 (2020) 107040.
[71] M. Narasimhan, S. Lazebnik, A. Schwing, Out of the box: reasoning with graph convolution nets for factual visual question answering, in: NeurIPS, 2018, pp. 2659–2670.
[72] Q. Liu, M. Allamanis, M. Brockschmidt, A. Gaunt, Constrained graph variational autoencoders for molecule design, in: NeurIPS, 2018, pp. 7806–7815.
[73] B. Fatemi, P. Taslakian, D. Vazquez, D. Poole, Knowledge hypergraphs: prediction beyond binary relations, arXiv:1906.00137(2019).
[74] T. Jin, J. Yu, J. You, K. Zeng, C. Li, Z. Yu, Low-rank matrix factorization with multiple hypergraph regularizer, Pattern Recognit. 48 (3) (2015) 1011–1022.
[75] Z. Li, Q. Chen, V. Koltun, Combinatorial optimization with graph convolutional networks and guided tree search, in: NeurIPS, 2018, pp. 537–546.

Song Bai is currently a research fellow in the Department of Engineering Science at the University of Oxford, and a member of Torr Vision Group. He received the B.E. and Ph.D. degree in Electronics and Information Engineering from Huazhong University of Science and Technology (HUST), Wuhan, China. He was with University of Texas at San Antonio (UTSA) and Johns Hopkins University (JHU) as a research scholar. His research interests include computer vision and machine learning, especi ally graph neural networks, adversarial learning, and medical image analysis. He serves as an Associate Edit or of Pattern Recognition (PR).

Feihu Zhang is a second year DPhil/PhD student in Torr Vision Group at the University of Oxford, supervised by Prof. Philip H.S. Torr and Prof. Victor Prisacariu. His research interests include computer vision, deep learning and autonomous driving.

Philip H.S. Torr is a professor at the University of Oxford. He has won awards from top vision conferences, including ICCV, CVPR, ECCV, NIPS and BMVC. He is a senior member of the IEEE and a Royal Society Wolfson Research Merit Award holder.