HNHN: Hypergraph Networks with Hyperedge Neurons

Yihe Dong [1] **Will Sawin** [2] **Yoshua Bengio** [3]

Abstract

Hypergraphs provide a natural representation for many real world datasets. We propose a novel framework, HNHN, for hypergraph representation learning. HNHN is a hypergraph convolution network with nonlinear activation functions applied to both hypernodes and hyperedges, combined with a normalization scheme that can flexibly adjust the importance of high-cardinality hyperedges and high-degree vertices depending on the dataset. We demonstrate improved performance of HNHN in both classification accuracy and speed on real world datasets when compared to state of the art methods. Our code is available at https://github.com/twistedcubic/HNHN.

1. Introduction

Much real world data can be represented as graphs, as data come in the form of objects and relations between the objects. In many cases, a relation can connect more than two objects. A hypergraph can naturally represent such structures. Our goal is to learn representations of such structured data with a novel hypergraph convolution algorithm. First let's recall the vanilla graph neural network (GNN): fix a graph G with n vertices, and let $A \in \mathbb{R}^{n\times n}$ be its adjacency matrix. Choosing a suitable d, we define node representations $X \in \mathbb{R}^{n\times d}$, as well as weights $W \in \mathbb{R}^{d\times d}$ and bias $b \in \mathbb{R}^d$, and a nonlinear activation function σ. In each layer of the GNN, the node representations are updated as: $X' = \sigma(AXW + b)$.

Some prior research has generalized this to hypergraphs by defining an appropriate hypergraph analogue of the adjacency matrix A (Feng et al., 2018; Bai et al., 2019). For vertices i and j, the entry A_{ij} can be defined as a sum over hyperedges containing both i and j of a weight function, that may depend on the number of vertices in the hyperedge. However, these approaches do not fully utilize the hypergraph structure. In fact, the matrix A defined this way can be seen to equal the adjacency matrix of the graph obtained by replacing each hyperedge with a (weighted) clique. Thus, the hypergraph algorithm will have no better accuracy on a given task than the corresponding graph algorithm, applied to this graph built from cliques.

Hyperedge nonlinearity. To solve this problem, we treat hyperedges as objects worthy of study in their own right. We therefore train a network with one representation X_V for hypernodes and another representation X_E for hyperedges. We can then use the hypernode-hyperedge incidence matrix for the convolution step. A nonlinear function σ then acts on both hypernodes and hyperedges. By making this change, we allow the network to learn nonlinear behavior of individual hyperedges, which plausibly exists in many real-world data sets. For instance, the probability that one author of a paper with unknown research interests works in a particular research area might be a nonlinear function of the number of other authors of the paper that work in that research area.

Normalization. In addition, we take a different, more flexible approach to normalization. Normalization ensures numerical stability during training. Different options exist for normalization, depending on how we weight the different vertices (as a function of their degrees). We allow these choices to depend on hyperparameters that are optimized for a given data set. Indeed, there is little reason to suspect that the most mathematically elegant normalization formula, for instance using the symmetric normalized Laplacian, $D^{-1/2}LD^{-1/2}$, necessarily gives the best accuracy for prediction tasks. Instead, normalization represents a choice of how to weight vertices of large degree compared to vertices of smaller degree, and is dataset-dependent.

2. Model architecture and analysis

First some **notation**: we define a hypergraph H to consist of a set V of hypernodes and set E of hyperedges, where each hyperedge is itself a set of hypernodes. Let $n = |V|$ and $m = |E|$. Indexing the hypernodes as v_i for $i \in \{1, ..., n\}$, and the hyperedges as v_j for $j \in \{1, ..., m\}$, we define the *incidence matrix* $A \in \mathbb{R}^{n\times m}$ by $A_{ij} = 1$ if $v_i \in e_j$ and $A_{ij} = 0$ if $v_i \notin e_j$.

We then update the hypernode representations $X_V \in \mathbb{R}^{n\times d}$ and hyperedge representations $X_E \in \mathbb{R}^{m\times d}$ by a con-

[1]Microsoft [2]Department of Mathematics, Columbia University [3]Mila, Université de Montréal.

volution using the incidence matrix A. Thus, our update rule is given by the formulas $X'_E = \sigma(A^T X_V W_E + b_E)$ and $X'_V = \sigma(A X'_E W_V + b_V)$, where σ is a nonlinear activation function, $W_V, W_E \in \mathbb{R}^{d\times d}$ are weight matrices, and $b_V, b_E \in \mathbb{R}^D$ are bias matrices.

We use $\mathcal{N}_i$ throughout to denote the edge neighborhood of v_i, i.e. the set of j with $v_i \in e_j$. Similarly, $\mathcal{N}_j$ denotes the vertex neighborhood of e_j, i.e. the set of i with $v_i \in e_j$.

2.1. Relationship with clique and star expansions

In this subsection, we discuss how HNHN hypergraph convolution relates to graph convolution. To apply graph convolution to hypergraph problems, we must build a graph G from our hypergraph H. There are two main approaches to this in the literature.

The first, the *clique expansion* (Sun et al., 2008; Zhou et al., 2007; Tu et al., 2018; Muhan Zhang & Chen, 2018) produces a graph whose vertex set is V by replacing each hyperedge $e = \{v_1, \ldots, v_k\}$ with a clique on the vertices $\{v_1, \ldots, v_k\}$. A slight variant of this produces a weighted graph where the weight of each edge in the clique is equal to some fixed function of k. We will consider another variant G_c, where we replace each hyperedge with a clique plus an edge from each vertex $v_1, \ldots, v_k$ to itself.

The second, the *star expansion* or *bipartite graph model* (Zien et al., 1999), produces a graph G_* whose vertex set is $V \cup E$, with an edge between a hypernode v and a hyperedge e if $v \in e$, and with no edges between hypernodes and hypernodes or hyperedges and hyperedges, making the graph bipartite. (For $e = \{v_1, \ldots, v_k\}$ a hyperedge, the subset $\{e, v_1, \ldots, v_k\}$ is a star graph, explaining the name.)

We briefly summarize the relationships between our method and graph convolution on these two graphs:

- If we use the same weights for hypernodes and hyperedges, setting $W_E = W_V$ and $b_E = b_V$, then HNHN is equivalent to graph convolution on the star expansion G_*.
- If we apply the nonlinear activation function σ only to hypernodes, and not to hyperedges, then HNHN is equivalent to graph convolution on the clique expansion G_c.

These claims will be formally justified in the weight simplification section and Lemma 2.1.

Note here that if we remove both the nonlinear activation function and the weight matrix, so we consider only the linear action of the adjacency matrix, then graph convolution on G_c and G_* are equivalent to each other, because the spectra of the adjacency matrices of G_c and G_* are related by a simple operation (explained after Lemma 2.1).

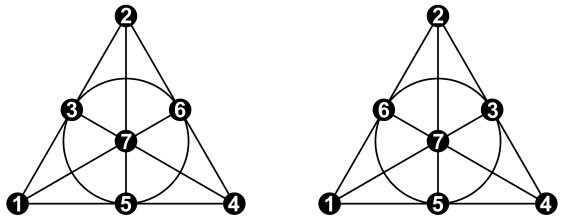

Figure 1. The Fano plane, and a copy with the nodes 3 and 6 permuted. The six straight lines and the circle represent hyperedges. When given the hypernode label numbers, or other equivalent information, as input features, approaches based on the clique expansion will not be able to distinguish these two hypergraphs, while HNHN and other approaches can.

We now explain the advantages of our method over graph convolution on both the clique and star expansion.

Comparison with clique expansion. The clique expansion is a problematic approach to studying any hypergraph problem because it entails a loss of information - i.e. there can be two distinct hypergraphs on the same vertex set with the same clique expansion. An example is provided by the Fano plane, which is the hypergraph F with vertex set $\{1, 2, 3, 4, 5, 6, 7\}$ and the seven hyperedges $\{\{1, 2, 3\}, \{1, 4, 5\}, \{1, 6, 7\}, \{2, 4, 6\}, \{2, 5, 7\}, \{3, 4, 7\}, \{3, 5, 6\}\}$. Because each pair of distinct vertices lies in exactly one hyperedge, the clique expansion of F is the complete graph K_7 on seven vertices.

If we permute the seven hypernodes of F by an arbitrary permutation in S_7, changing the hyperedges appropriately, we will obtain another hypergraph whose clique expansion is again K_7 for the same reason. However, it will usually not be the same hypergraph. In fact, by the orbit-stabilizer theorem we can obtain $\frac{7!}{168} = 30$ different hypergraphs this way, where 168 is the number of permutations that stabilize F.

Hence graph convolution on the clique expansion will not be able to solve problems on F whose answers change when the hypernode labels are permuted. More generally, examples of this kind can be produced of hypergraphs on $q^2 + q + 1$ hypernodes for any prime power q, using finite projective planes.

Comparison with star expansion. The star expansion does not lose information in this way. However, it treats hypernodes and hyperedges the same. Because, in many datasets, hypernodes and hyperedges describe completely different objects, it is reasonable to give them different weight matrices, and we expect the additional expressive power to improve accuracy.

(Agarwal et al., 2006) showed that clique expansion and star expansion are equivalent in a certain sense, and has been

cited to claim that they are equivalent in general. However, (Agarwal et al., 2006) worked in a linear setting, considering just the action of the adjacency matrix. In a neural network setting, when a nonlinear activation function σ is included, their proof does not apply.

Weight simplification. We now explain how removing the distinction between W_V and W_E reduces HNHN to graph convolution on G_*.

For this purpose, let $B \in \mathbb{R}^{(n+m)\times(n+m)}$ be the adjacency matrix of G_*. Then $B = \begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$. If we set $W_E = W_V = W$ and $b_E = b_V = b$ in the update rule, we have for all i:

$$X_E^{i+1} = \sigma(A^T X_V^i W + b) \text{ and } X_V^{i+1} = \sigma(A X_E^{i+1} W + b)$$

and for all i we define X^{2i} to be the concatenation of X_V^i with the $m \times d$ matrix of zeroes, and X^{2i+1} to be the concatenation of the $n \times d$ matrix of zeroes with X_E^{i+1}, then for all i we have $X^{i+1} = \sigma(BX^iW + b)$, because the matrix B sends the E coordinates to the V coordinates and vice versa.

Linear simplification. Let C be the adjacency matrix of G_c. The following lemma shows that convolution on convolution on H, with the nonlinear activation function σ applied only to hypernodes, is equivalent to convolution on G_c. All proofs are in the supplement.

Lemma 2.1. *If we define*

$$X'_E = A^T X_V W_E + b_E \quad \text{and} \quad X'_V = \sigma(A X'_E W_V + b_V)$$

then $X'_V = \sigma(C X_V W_c + b_c)$, *where* $W_c \in \mathbb{R}^{d\times d}$, *and* $b_c \in \mathbb{R}^d$.

A key step in the proof is the following lemma, which relates the adjacency matrix C of G_c to the incidence matrix A.

Lemma 2.2. *We have* $C = AA^T$.

Relationship between clique and star expansions. It is a classical result that the eigenvalues of B are equal to the set of square roots of eigenvalues of $AA^T = C$ together with $m - n$ additional zeroes (see e.g. (Bhattacharya et al., 2008)). Thus, the spectral theory of G_* is no more complex or simpler than the spectral theory of G_c.

Additional connections to prior work. Both HGNN (Feng et al., 2018) and the method of HCHA (Bai et al., 2019) (when the optional attention module is not used) are mathematically equivalent to ordinary graph convolution on the clique expansion. As shown above with the Fano plane, these approaches don't utilize all structural information in the hypergraph.

HyperGCN (Yadati et al., 2019) relies on replacing a hypergraph with a graph, but in a more subtle way than a clique expansion. Rather than replacing a hyperedge $\{v_1, \ldots, v_k\}$ by a clique, i.e., an edge between every pair of vertices, they pick two special vertices from $\{v_1, \ldots, v_k\}$, called "mediators", and keep only the edges connecting to at least one of the two mediators (Chan & Liang, 2020). This also does not always retain the full information from the hypergraph structure. In fact, in the case where every hyperedge contains at most three hypernodes, HyperGCN is equivalent to the clique expansion, because every edge in the clique touches at least one mediator.

Another hypergraph learning method, Hyper-SAGNN (Zhang et al., 2019), uses a very different architecture. Rather than studying the global graph structure, Hyper-SAGNN focuses on a fixed set of vertices to predict whether they are connected by a hyperedge, treating them symmetrically when doing so.

On the other hand, some prior graph convolution works have taken a similar approach to HNHN by placing neurons on each hyperedge. These include (Kearnes et al., 2016), and, building on their work, (Gilmer et al., 2017). Unlike HNHN, these works restrict to the case of graphs, and they do not use our normalization scheme, discussed below.

2.2. Hypergraph normalization

One common message passing scheme is to normalize the messages by the cardinality of the neighborhood after message pooling. In the case of passing messages from hyperedges to hypernodes, this can be expressed as $X'_V = \sigma(D_V^{-1} A X'_E W_V + b_V)$, where $D_V \in \mathbb{R}^{n\times n}$ is the diagonal matrix where the i^{th} diagonal entry $(D_V)_{ii}$ is $|\mathcal{N}_i|$. This ensures, for unbounded activation functions σ like ReLU, that the representation vector does not grow rapidly from one layer to the next when v_i has large degree, and, for bounded activation functions σ, that the input of σ is not too large. We could also multiply on the left and right by $D_V^{-1/2}$, which would be equivalent for activation functions like ReLU that commute with multiplication by a positive real number.

However, the contribution of each hyperedge is given the same weight, regardless of its degree. We consider a generalization where, before summing over hyperedges incident to a given hypernode, we weight the contribution of each hyperedge by a power of its degree, depending on a real parameter α:

$$X'_V = \sigma(D_{V,l,\alpha}^{-1} A D_{E,r,\alpha} X'_E W_V + b_V)$$

where $D_{E,r,\alpha} \in \mathbb{R}^{m\times m}$ and $D_{V,l,\alpha} \in \mathbb{R}^{n\times n}$ are the diagonal matrices with diagonal terms: $(D_{E,r,\alpha})_{jj} = |\mathcal{N}_j|^\alpha$, and $(D_{V,l,\alpha})_{ii} = \sum_{j\in\mathcal{N}_i} |\mathcal{N}_j|^\alpha$. If $\alpha = 0$, then $D_{E,r,\alpha}$ is the identity matrix and $D_{V,l,\alpha} = D_v$, so this generalizes the normalization above.

Equivalently, to calculate the $(i,t)^{th}$ entry of X'_V, the matrix multiplication above gives the following formula:

$$(X'_V)_{it} = \sigma\left(\frac{\sum_{j\in\mathcal{N}_i}|\mathcal{N}_j|^\alpha \sum_{s=1}^{d}(X'_E)_{is}(W_V)_{st}}{\sum_{j\in\mathcal{N}_i}|\mathcal{N}_j|^\alpha} + (b_V)_t\right)$$

Here the $D_{E,r,\alpha}$ matrix produces the factor $|\mathcal{N}_j|^\alpha$ in each term in the numerator. After including $D_{E,r,\alpha}$ we choose the $D_{E,l,\alpha}$ matrix to produce the factor $\sum_{j\in\mathcal{N}_i}|\mathcal{N}_j|^\alpha$ in the denominator, to normalize for the total size of the numerator.

When $\alpha > 0$, the contributions of hyperedges with large degrees are increased compared with the previous normalization, while if $\alpha < 0$, their contributions are decreased. We treat α as a hyperparameter to be optimized for a given data set. By fixing $\alpha = 0$, as is equivalent to the normalization in many prior works (Kipf & Welling, 2017; Bai et al., 2019; Feng et al., 2018), we would be assuming that hyperedges of different sizes have equal importance, on average, for learning the representation of vertices. Instead, we believe that on some datasets, hyperedges with a large number of vertices are an ineffective guide to labels, compared to hyperedges with a small number of vertices, while for other datasets it may be the other way around. For instance, in coauthorship graphs, papers with a large number of authors may frequently reflect large collaborations between scientists of different fields, and thus will not be as predictive as papers with a smaller number of authors. By allowing a varying hyperpameter α, we can choose a weighting for hyperedges that is appropriate for a given dataset.

Analogously, we weight vertices in terms of their degrees according to hyperparameter $\beta \in \mathbb{R}$: $X'_E = \sigma(D_{E,l,\beta}^{-1} A D_{V,r,\beta} X_V W_E + b_E)$ where the diagonal matrices $D_{E,l,\beta}$ and $D_{V,r,\beta}$ are given by: $(D_{E,l,\beta})_{jj} = \sum_{i\in\mathcal{N}_j}|\mathcal{N}_i|^\beta$, and $(D_{V,r,\beta})_{ii} = |\mathcal{N}_i|^\beta$.

Implementation. As outlined in Algorithm 1 in the supplement, the inputs of our algorithm are the incidence relations between the hypernodes and hyperedges, the input feature vectors of the hypernodes, and the target labels for a subset of the hypernodes. From the hypergraph structure, we extract neighborhood information to compute the normalization factors as in §2.2, then each layer of HNHN hypergraph convolution relays signals from the hypernodes to the hyperedges, with a nonlinearity and node-specific normalization, then analogously from hyperedges to hypernodes. Finally the loss is calculated using cross entropy between the predicted labels based on the learned node representations and the given target labels. Dropout is added to any convolution layer that's not the last to mitigate overfitting (Srivastava et al., 2014). ReLU (Glorot et al., 2011) is used as the nonlinearity throughout.

Time complexity. For a hypergraph H with n hypernodes, m hyperedges, average vertex degree δ_V, and hidden dimension d, the time complexity of HNHN is $O(n\delta_V d + nd^2 + md^2)$. For derivation please refer to the supplement. Thus, HNHN is similar to the most computationally efficient existing hypergraph convolution algorithms, such as HyperGCN, and is faster than hypergraph algorithms based on clique expansions.

	Accuracy			
	DBLP	Cora	CiteSeer	PubMed
HyperGCN	71.3±1.2	55.0±.9	54.7±9.8	60.0±10.7
* Fast	70.5±14.3	45.2±12.9	56.1±11.2	54.4±10.0
HGNN	77.6±.4	58.2±.3	61.1±2.2	63.3±2.2
HNHN	**85.1±.2**	**63.9±.8**	**64.8±1.6**	**75.9±1.5**
	Timing			
HyperGCN	563.4±27.8	183.4±2.7	15.6±.2	171.1±2.8
* Fast	**11.5±.1**	**2.9±.1**	**1.1±0.**	**2.5±.1**
HGNN	802.9±59.2	298.4±12.2	30.5±.8	270.1±10.5
HNHN	44.2±1.3	13.6±5.4	1.3±.1	26.6±.4

Table 1. Hypernode classification accuracy and timing results. Accuracies are in %, timings are measured in seconds. * Fast stands for HyperGCN Fast.

3. Experiments

Datasets. We evaluate the quality of HNHN hypergraph representations on several commonly used benchmark datasets: CiteSeer (Bhattacharya & Getoor, 2007) and PubMed (Namata et al., 2012), both co-citation datasets, Cora (Liu & Getoor, 2003; Sen et al., 2008) and DBLP (for Informatics), both co-authorship datasets. Please see the appendix for additional experiments, including studies on normalization.

Hypernode prediction. Given a hypergraph and node labels on a small subset of hypernodes, this task is to predict labels on the remaining hypernodes. To train, cross-entropy loss is calculated between the predicted and target labels. We use the Adam (Kingma & Ba, 2015) optimizer with a learning rate scheduler that multiplicatively reduces the learning rate at regular intervals. As shown in Table 1, HNHN outperforms state of the art techniques across datasets, while achieving competitive timing results.

Reducing feature dimensions. We experiment with node prediction after reducing the input feature dimensions using latent semantic analysis (Deerwester et al., 1990). As expected, this led to faster training, at the cost of slightly reduced accuracy. On Cora, after the input feature dimension is reduced from 1000 to 300, the accuracies are 63.69±0.58, 56.63±0.33, and 42.4±1.58, for HNHN , HGNN, and HyperGCN, respectively, with corresponding training times, 10.84±0.14, 152.34±1.14, and 182.56±2.38 seconds. This is a drop of 0.17, 1.52, and 12.6 points in accuracy, respectively, while the running time is 79.5%, 51.1%, and 98.8% that of the non-reduced case. The accuracy advantage of HNHN over state of the art methods is preserved in this setting, showing it is also suitable when computational resources are limited.

References

Agarwal, S., Branson, K., and Belongie, S. Higher order learning with graphs. In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 17–24, 2006.

Bai, S., Zhang, F., and Torr, P. H. Hypergraph convolution and hypergraph attention. arXiv:1902.09702, 2019.

Bhattacharya, A., Friedland, S., and Peled, U. N. On the first eigenvalue of bipartite graphs. *The Electronic Journal of Combinatorics*, 15, 2008.

Bhattacharya, I. and Getoor, L. Collective entity resolution in relational data. *TKDD*, 2007.

Chan, T. H. H. and Liang, Z. Generalizing the hypergraph Laplacian via a diffusion process with mediators. *Theoretical Computer Science*, 806:416–428, 2020.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990. doi: 10.1002/(SICI)1097-4571(199009)41:6⟨391::AID-ASI1⟩3.0.CO;2-9. CiteSeerX 10.1.1.108.8490.

Feng, Y., You, H., Zhang, Z., Ji, R., and Gao, Y. Hypergraph neural networks. *AAAI*, 2018.

for Informatics, L. C. DBLP. https://dblp.uni-trier.de/.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1263–1272. JMLR.org, 2017.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. *AISTATS*, 2011.

Group, U. S. R. L. Citeseer for document classification. https://linqs.soe.ucsc.edu/data.

Kearnes, S., McCloskey, K., Berndl, M., Pande, V., and Riley, P. Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design*, 30(8):595–608, 2016. doi: 10.1007/s10822-016-9938-8. URL https://doi.org/10.1007/s10822-016-9938-8.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *International Conference for Learning Representations*, 2015.

Kipf, T. and Welling, M. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Liu, Q. and Getoor, L. Link-based classification. In *ICML'03: Proceedings of the Twentieth International Conference on International Conference on Machine*, 2003.

Mccallum, A. Cora information extraction data. https://people.cs.umass.edu/ mccallum/data.html.

Muhan Zhang, Zhicheng Cui, S. J. and Chen, Y. Beyond link prediction: Predicting hyperlinks in adjacency space. In *Proceedings of the Thirty-Second Conference on Association for the Advancement of Artificial Intelligence*, 2018.

Namata, G., London, B., Getoor, L., and Huang, B. Query-driven active surveying for collective classification. In *Proceedings of the Workshop on Mining and Learning with Graphs*, 2012.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. Collective classification in network data. *AI magazine*, 29(3), 2008.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Sun, L., Ji, S., and Ye, J. Hypergraph spectral learning for multi-label classification. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 668–676, 2008.

Tu, K., Cui, P., Wang, X., Wang, F., and Zhu, W. Structural deep embedding for hypernetworks. In *Proceedings of the Thirty-Second Conference on Association for the Advancement of Artificial Intelligence*, 2018.

Xu, Q., Jeon, H., and Annavaram, M. Graph processing on GPUs: where are the bottlenecks? *IEEE International Symposium on Workload Characterization*, 2018.

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

Zhang, R., Zou, Y., and Ma, J. Hyper-SAGNN: a self-attention based graph neural network for hypergraphs. https://arxiv.org/abs/1911.02613, 2019.

Zhou, D., Huang, J., and Schölkopf, B. Learning with hypergraphs: clustering, classification, and embedding. In *Advances in Neural Information Processing Systems*, pp. 1601–1608, 2007.

Zien, J. Y., Schlag, M. D., and Chan, P. K. Multilevel spectral hypergraph partitioning with arbitrary vertex sizes. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 18(9):1389–1399, 1999.

Appendix

We first provide proofs to lemmas 2.1 and 2.2, followed by an algorithm outline for HNHN and further analysis such as time complexity, finally we give additional details on dataset processing and empirically study HNHN properties, such as effects of the normalization parameters.

1. Proofs to lemmas

We prove the lemmas on the linear simplification of HNHN . Recall that C is the adjacency matrix of G_c.

Lemma 2.2. We have $C = AA^T$.

Proof. By definition

$$(AA^T)_{i_1 i_2} = \sum_j A_{i_1 j} A_{i_2 j} = \sum_{\substack{j \\ v_{i_1} \in e_j \\ v_{i_2} \in e_j}} 1$$

and $C_{i_1 i_2}$ is the number of edges from v_{i_1} to v_{I_2} in G_c. Recall that G_c is the graph obtained from H by replacing each hyperedge with a clique and a union of self-edges. The clique obtained from a hyperedge e_j contains an edge from v_{i_1} to v_{i_2} if and only if $v_{i_1} \in e_j, v_{i_2} \in e_j$, and $v_{i_1} \neq v_{i_2}$, and the self-edges obtained from a hyperedge e contain an edge from v_{i_1} to v_{i_2} if and only if $v_{i_1} = v_{i_2} \in e_j$, so in total the number of edges from v_{i_1} to v_{i_2} of G_c is the number of hyperedges containing both v_{i_1} and v_{i_2}.

Because each entry of C equals the corresponding entry of AA^T, it follows that $C = AA^T$. □

With this, we can now prove:

Lemma 2.1. If we define

$$X'_E = A^T X_V W_E + b_E \quad \text{and} \quad X'_V = A X'_E W_V + b_V$$

then

$$X'_V = C X_V W_c + b_c \tag{1}$$

where $W_c \in \mathbb{R}^{d \times d}$, and $b_c \in \mathbb{R}^d$.

Proof. We have

$$\begin{aligned} X'_V &= A X'_E W_V + b_V = A(A^T X_V W_E + b_E) W_V + b_V \\ &= AA^T X_V W_E W_V + A b_E W_V + b_V. \end{aligned}$$

Setting $W_c = W_E W_V$ and $b_c = A b_E W_V + b_V$, and using $C = AA^T$, we obtain (1). □

2. Algorithm outline

Algorithm 1 gives an outline of the HNHN hypergraph convolution applied to hypernode classification. While Algorithm 1 is stated in terms of the incidence matrix A for clarity, HNHN convolution doesn't require any explicit instantiation of A, which has size $O(mn)$. Even though the number of nonzero entries in A is usually much smaller and thus can be represented as a sparse matrix, never instantiating A is more advantageous for training, as GPUs are much more adaptable for dense over sparse operations (Xu et al., 2018). Our implementation uses efficient GPU operations that only uses indices of hyperedges connected to a hypernode and of hypernodes connected a hyperedge, without ever instantiating the incidence matrix. This contributes to the fast HNHN runtimes as reported in Table 1.

3. Time Complexity

For a hypergraph H with n hypernodes, m hyperedges, average vertex degree δ_V, and hidden dimension d, the time complexity of HNHN is $O(n\delta_V d + nd^2 + md^2)$. This complexity is because multiplying X_V by A^T or X_E by A takes time proportional to the number of nonzero entries of A, which is $n\delta_V$, times d. Multiplying by W_V takes $O(nd^2)$ while multiplying by W_E takes time $O(md^2)$. Finally, applying σ to each entry takes time $O(nd+md)$ which is $\leq O(nd^2+md^2)$.

Algorithm 1 HNHN hypergraph convolution algorithm for node prediction

Input: Hypergraph incidence matrix $A \in \mathbb{Z}^{n \times m}$, hypernode representations X_V
Input: Set of target labels $\{y\}_{k \in L}$
Compute: $D_{E,l,\alpha}$, $D_{V,r,\alpha}$, $D_{V,l,\alpha}$, and $D_{E,r,\alpha}$ as in § 2.2
for $i = 1$ **to** *n_epochs* **do**
  Initialize $X_E \leftarrow \tilde{0}$. Project X_V to hidden dimension
  for $j = 1$ **to** *n_layers* **do**
    Normalize hypernodes: $\widetilde{X_V} = D_{E,l,\beta}^{-1} A D_{V,r,\beta} X_V$
    Update hyperedges: $X_E = \sigma(\widetilde{X_V} W_{E,j} + b_{E,j})$
    Normalize hyperedges: $\widetilde{X_E} = D_{V,l,\alpha}^{-1} A D_{E,r,\alpha} X_E$
    Update hypernodes: $X_V = \sigma(\widetilde{X_E} W_{V,j} + b_{V,j})$
  end for
  Compute cross-entropy **loss** between $\{y\}_{k \in L}$ and predictions using $\{X_V\}_{k \in L}$
  Backpropagate on loss and **optimize** parameters e.g. $\{W_E\}_j$, $\{W_V\}_j$ $\{b_E\}_j$, $\{b_V\}_j$
end for
Return: Learned representations X_V, X_E for prediction tasks

Thus, HNHN is similar to the most computationally efficient existing hypergraph convolution algorithms, such as HyperGCN. HyperGCN replaces each hyperedge e_j with $2|\mathcal{N}_j| - 3 = O(|\mathcal{N}_j|)$ edges, thus replacing H with a graph that has $O(m\delta_E) = O(n\delta_V)$ edges, where δ_E is the average hyperedge degree of H. Applying graph convolution on this graph then takes time proportional to the number of edges, which is again $O(n\delta_V)$.

HNHN is faster than hypergraph algorithms based on clique expansion, which require replacing a hyperedge e_j with $\frac{|\mathcal{N}_j|(|\mathcal{N}_j|-1)}{2} = O(|\mathcal{N}_j|^2)$ edges, for a total of $O(m\delta_E^2) = O(n\delta_V\delta_E)$ edges, producing a graph on which graph convolution takes time $O(n\delta_V\delta_E d)$.

Table 1 describes the timing results for training node classification. Consistent with the above analysis, algorithms that expand each hyperedge into a linear number of edges perform faster than clique expansion-based algorithms.

4. Dataset processing

We give further details on the datasets used and the data processing pipeline. Co-authorship data consist of a collection of papers with their authors. To create a hypergraph, each paper becomes a hypernode and each author a hyperedge. The hypernodes $v_1, ..., v_k$ are connected to a hyperedge e if the papers corresponding to $v_1, ..., v_k$ are written by the author corresponding to e. Co-citation data consists of a collection of papers and their citation links. To create a hypergraph, each hypernode represents a paper, the hypernodes $v_1, ..., v_k$ are connected to the hyperedge e if the papers corresponding to $v_1, ..., v_k$ are cited by e.

To create the initial hypernode representation vectors X_V, TFIDF representations are created based on contents in the paper. The vocabulary used for the TFIDF vectors consist of the most common words in each dataset. CiteSeer and PubMed data were obtained from (Group), DBLP was processed according to (Yadati et al., 2019), and Cora was parsed and processed from the source (Mccallum). Nodes not connected to any hyperedge, as well as hyperedges containing only one hypernode, were removed.

5. Additional experiments

Implementation details. Hyperparameters α and β are determined by 5-fold cross-validation on the training set. We tune hyperparameters such as the number of layers and dropout rate. In practice, at most two convolution layers are needed to achieve the reported accuracies. Experiments are done on a 24-core 2.6 GHz-CPU machine with a Nvidia P40 GPU. As shown in Table 1, HNHN outperforms state of the art techniques across datasets, while achieving competitive timing results.

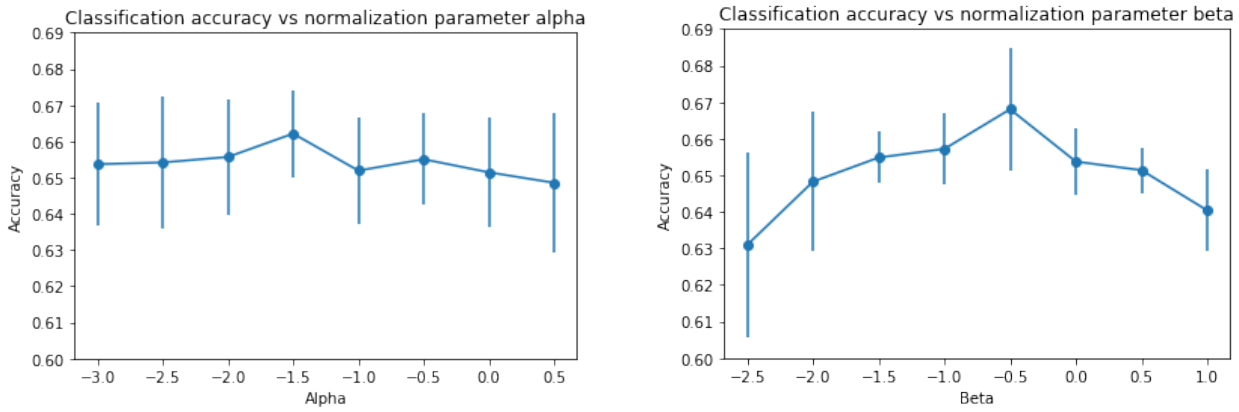

Figure 2. Node classification accuracy on CiteSeer as a function of the normalization parameter α while fixing $\beta = 0$ (left) and β while fixing $\alpha = 0$ (right). This shows normalization has tangible effects on accuracy, and that the commonly used normalization corresponding to $\alpha = \beta = 0$ is not always optimal.

5.1. Effects of normalization parameters

As discussed in §2.2, the normalization parameter α controls how much weight difference exists between hyperedges of different cardinalities. When $\alpha > 0$, larger-sized hyperedges are given greater weight; when $\alpha < 0$, smaller hyperedges are weighted more. Analogously for β on weights of hypernodes. Figure 2 demonstrates the effects of α and β on CiteSeer node prediction accuracy, showing that the commonly used normalization corresponding to the special case $\alpha = \beta = 0$ is not necessarily optimal for node prediction. We note that the effects of β on the accuracy shown in Figure 2 are greater than the effects of α. This is likely because the vertex degrees (which we raise to the power β) vary more than the edge degrees (which we raise to the power α). Indeed, the vertex degrees on this dataset have a standard deviation of 3.08, compared to an average degree of 2.07, for a ratio of 1.49, while the edge degrees have a standard deviation of 1.67 with an average degree of 2.8, for a ratio of only 0.60.

5.2. Edge representations learning

Compared to approaches that rely on hyperedge expansions, HNHN has the advantage in that it produces one dedicated vector representation per hyperedge. This differs from prior works that expand out a hyperedge into multiple nodes or select a representative subset of hypernodes for each hyperedge (Feng et al., 2018; Yadati et al., 2019). Hence HNHN learned representations can be easily adapted for downstream edge-related tasks such as edge prediction. Indeed, on the co-citation dataset CiteSeer where both hypernodes and hyperedges represent papers, when given 15% of hyperedge labels (and no hypernode labels), the hyperedge classification accuracy is 62.79 ± 1.43. This is comparable with the 64.76 ± 1.63 hypernode prediction accuracy (when only 15% of the hypernodes are labeled), reflecting the hyperedge-hypernode symmetry in HNHN.