Joint Symmetric Tensor Decomposition for Hypergraph Embeddings

Ankit Sharma and Jaideep Srivastava

University of Minnesota
200 Union St. SE
Minneapolis, MN USA - 55455
{ankit,srivasta}@cs.umn.edu

Abstract. Data structured in the form of overlapping or non-overlapping sets are found in a variety of domains, sometimes explicitly but often subtly. For example, teams, which are of prime importance in social science studies, are "sets of individuals"; "item sets" in pattern mining are examples of groups, and for various types of analyses in language studies a sentence can be considered as a "set or bag of words". Although building models and inference algorithms for structured data has been an essential task in the fields of machine learning and statistics, research on "set-like" data remains less explored. Dyadic edges in a graph model relationships between pairs of elements. However, for modeling relationships that involve all members of a set, a hyperedge is a more accurate representation. In this work, we focus on the problem of embedding hyperedges of a hypergraph (a network of overlapping sets) to a low dimensional vector space. While introducing the novel concept of dual tensors, we propose a symmetric tensor-based algebraic model that captures the hypergraph structure in a principled manner (without losing set-level information). More importantly, the proposed tensor methods are for general non-uniform hypergraphs, which has not been studied previously in tensor literature. Our central focus is to: (a) highlight the connection between hypergraphs (topology) and tensors (algebra) and (b) provide a comparison between graphs and hypergraphs. Our hypergraph tensor decomposition method outperforms graph (or graph proxies for hypergraph) based spectral baselines on real-world as well as synthetic datasets. We, therefore, argue that the proposed methods are more generic methods suitable for hypergraphs (and thus also for graphs) that preserve accuracy and efficiency.

1 Introduction

In group-structured data, we have multiple entities related by some form of group relation. Such relationships occur far more frequently in the real world than has usually been studied [13]. Noticeably in social groups, like collaboration data of scientific or software teams [10]; team interaction logs from massive online multi-player games [2, 21], such as World of Warcraft; and further, interaction logs from group communication tools such as Skype and Google Docs, not to mention, e-mail data [18]. There are other fields in which modeling relationships between a collection of entities is crucial as well, and large group-structured datasets are available. Examples include Natural Language Processing [5], Biology [15] and e-commerce [12].

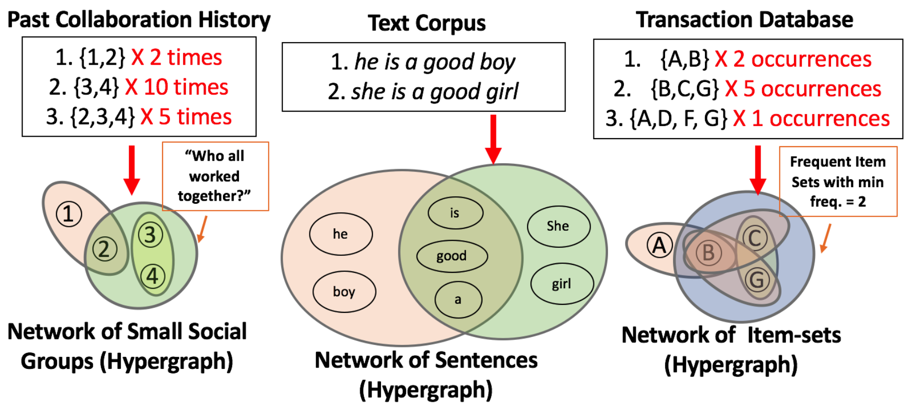
Past Collaboration History
1. {1,2} X 2 times
2. {3,4} X 10 times
3. {2,3,4} X 5 times
"Who all worked together?"
1
2
3
4
Network of Small Social Groups (Hypergraph)
Text Corpus
1. he is a good boy
2. she is a good girl
he
boy
is
good
a
She
girl
Network of Sentences (Hypergraph)
Transaction Database
1. {A,B} X 2 occurrences
2. {B,C,G} X 5 occurrences
3. {A,D, F, G} X 1 occurrences
Frequent Item Sets with min freq. = 2
A
B
C
G
Network of Item-sets (Hypergraph)

Fig. 1: "Set-like" hypergraph structures from various domains

Hypergraph [6], which is a generalization of graphs, is recognized for naturally modeling higher-order relationships between overlapping sets of objects [13]. Figure 1 shows example hypergraphs modeling networks of collaborations, sentences, and item-sets. Within machine learning, algorithms guided by hypergraph structure [24] have found applications in a variety of domains [19, 23].

Often it is useful to embed nodes of a graph to low dimensional vector space (by a process referred to as graph embedding) as various predictive tasks concerning nodes (like node classification) can utilize these embeddings. In this paper, we focus on learning *hypergraph embeddings*, which involves not just learning node embeddings but also hyperedge embeddings for a given hypergraph. However, unlike graphs (see [7]), learning node embeddings for hypergraph have been less explored. In this paper, we propose a method that: (1) learns hypergraph embeddings directly, (2) leverages the hypergraph topology, and (3) does not lose the hyperedge-level joint information. These learned embeddings can then be employed by a supervised or semi-supervised algorithm to perform various predictive tasks on nodes as well as hyperedges. Examples of hyperedge tasks can be, for example, performance prediction of a team hyperedge (set of individuals) engaged in a collaborative task or classification of individuals in a cohort for some psychological evaluation by embedding their response hyperedge (set of "positive" responses).

Most of the past methods learn *node* embeddings for hypergraphs by extending traditional graph embedding methods for hypergraph setting [24]. However, as debated by Agarwal et al. [1], such representations can be learned by constructing graphs, which are proxies for the hypergraph structure. However, these proxy graphs for a given hypergraph are not without merit, as observed in some recently theoretical studies [14]. In this work, we show how a k-way tensor can be used to represent a k-uniform hypergraph [17]; and to capture hypergraphs in a principled manner, any model should work at the level of tensors and not matrices, which only model dyadic affinities. This fact was first leveraged in the computer vision community by Shashua et al. [22], where they perform image segmentation using higher-order affinity tensor decomposition, and pointed to its connection with uniform hypergraphs.

In this work, we leverage these observations to design node embeddings for *general (non-uniform)* hypergraphs based on the joint decomposition of various cardinality hypergraph tensors. Further, we also introduce the concept of *dual tensors* for obtaining the *hyperedge* embeddings directly. Lastly, since a central focus of this work is to compare graphs versus hypergraphs, we compare our method with the graph (or graph proxies for hypergraph) based spectral baselines. Proposed hypergraph tensor decomposition outperforms baselines on several real-world as well as synthetic datasets. The main contributions of this work are as follows:

- We propose a general *hypergraph tensor decomposition* method designed for general hypergraphs (containing different cardinality hyperedges), unlike simple uniform hypergraph tensor decomposition, which is restricted to fixed cardinality hyperedges (i.e., uniform hypergraph). We are unaware of any such works that address this general problem.
- We propose the novel concept of a *dual tensor*, corresponding to the hypergraph dual that allows us to get a hyperedge embedding directly.
- We provide an empirical comparison between graphs and general hypergraphs in a principled manner using the statistical algorithms proposed.

The following is the outline for the rest of the paper. In Section 2, we describe the problem definition and statement followed by Section 3, where we describe in detail the methodology. Section 4 describes the datasets, experimental tasks & settings for the models, followed by the conclusion and appendix.

2 Preliminaries

Consider a scenario where we have a collection of *elements*. These elements can represent individual actors in case of social groups or words in sentences or items in item-sets within a transaction database. In other words a social group or a sentence or an item-set are *sets* which contain these *elements*. Let $V = \{v_1, v_2, ..., v_n\}$ represents n elements and we have m different sets defined over these elements, denoted by $G = \{g_1, g_2, ..., g_m\}$, where $g_i \subseteq V$ represents the i^{th} set. The cardinality $|g_i|$ represents the number of elements in the set. Also, each set $g_i \in G$ has an occurrence number $R(g_i)$, which denotes the number of times it has occurred. Such overlapping or non-overlapping sets can be modeled as a *hypergraph* [6], where the nodes and hyperedges, represent the elements and sets, respectively. This hypergraph is represented as $N_g = (V, G)$ with G as the collection of hyperedges over the nodes V. The incidence matrix $\mathbf{H} \in \{0, 1\}^{|G| \times |V|}$ for N_g represents the presence of nodes in different hyperedges with $\mathbf{H}(g_i, v) = 1$ if $v \in g_i$ else 0. We also define degree $d(v)$ of a vertex v as the number of hyperedges incident on this vertex i.e. $d(v) = \sum_{g_i \in G} \mathbf{H}(g_i, v)$.

Problem Statement: Given this setting, our goal is to learn the mapping $\phi : G \rightarrow \mathbb{R}^d$ from hyperedges to feature representations (i.e., embeddings) that can be used to build predictive models involving sets. Here d is a parameter specifying the number of dimensions of the embedding vector. Equivalently, ϕ can be thought of as a look-up matrix of size $|G| \times d$, where $|G|$ is the total number of sets or hyperedges.

3 Methodology

In this section, we develop tensor (higher-order matrix) based linear algebraic methods that learn node as well as hyperedge embedding by taking into account the joint probability over a hyperedge. The idea behind using tensors is that they retain the set-level information contained in a hypergraph, unlike the proxy graphs (corresponding to hypergraphs) based techniques (used as baselines in our experiments), which approximate hyperedge or set-level information with dyadic edge-level information. As noted before, this idea of tensor-based higher-order affinity retention was first highlighted in a statistical setting by Shashua et al. [22]. After this, we argue it has primarily remained unnoticed, more specifically, in the context of hypergraphs. The following proposition from combinatorial probability puts this argument more formally.

Proposition 1 *Given a set of random variables $X_1, ...X_c$ $(c \geq 2)$, and $H(.)$ as the information entropy, we have (*[8, p. 34]*):*

$$H(X_1, ..., X_c) \leq \left(\frac{1}{c-1}\right) \sum_{(i,j) \subseteq 2^{[c]}} H(X_i, X_j) \tag{1}$$

Therefore, the joint probability distribution over c cardinality hyperedges is more informative (lower entropy) than the total information attained from probability distributions over each of the $\binom{c}{2}$ dyadic edges.

Although tensors can retain higher order information, most research to date has focused on uniform hypergraphs using symmetric tensors. We propose an approach which is principally suited for *general* hypergraph structured data using higher-order tensors. For a given hypergraph we can extract a sub-hypergraph that only consists of the hyperedges with cardinality k. This sub-hypergraph is a k-uniform hypergraph or k-graph [11]. Corresponding to this k-uniform hypergraph, we can define a k^{th} order n-dimensional symmetric tensor [17] $\mathcal{A}^k_{\mathbf{hyp}} = (a_{p_1,p_2,..,p_k}) \in \mathbb{R}^{[k,n]}$ whose elements are as follows:

$$a_{p_1,p_2,..,p_k} = R(g_i) \tag{2}$$

where $\{v_{p_1}, v_{p_2}, ..., v_{p_k}\} \in g_i$ and $|g_i| = k, \forall i \in \{1, ..., m\}$. Note that symmetry here implies that the value of element $a_{p_1,p_2,..,p_k}$ is invariant under any permutation of its indices $(p_1, p_2, .., p_k)$. Rest of the elements in the tensor are zeros. We also define the lexicographically ordered index set for hyperedges:

$$\begin{aligned} \mathcal{P}^k = \big\{\mathbf{p} | \mathbf{p} = (p_1, p_2, .., p_k) \text{ where } \{v_{p_1}, v_{p_2}, ..., v_{p_k}\} \in g_i, \\ \forall g_i \in G \text{ s.t. } |g_i| = k \text{ and } p_1 < p_2 < ... < p_k\big\}, \end{aligned} \tag{3}$$

and we have different sets $\{\mathcal{P}^k\} \forall k \in \{c_{min}, .., c_{max}\}$ (c_{min} and c_{max} are the maximum and the minimum hyperedge cardinality in the given hypergraph). Notice that $\mathcal{P}^k$ contains unique (non-repetitive) indexes as there is only a single $\mathbf{p}$ corresponding to each of the different hyperedges $g_i \in G$. Consequently, we have $|\mathcal{P}^k| = |\{g_i : |g_i| = k\}|$.

Similarly, by interchanging the roles of nodes and hyperedges, we can also define a *dual tensor*, which corresponds to the *hypergraph dual*. We consider all the hyperedges in the hypergraph dual that are of cardinality k. These correspond to all the vertices in the original hypergraph, which have a degree k, i.e., they are part of exactly k hyperedges in the original hypergraph. Corresponding to this k-uniform hypergraph dual, we can define a k^{th} order m-dimensional symmetric dual tensor $\mathcal{A}^k_{\mathbf{dual}} = (a_{q_1,q_2,..,q_k}) \in \mathbb{R}^{[k,m]}$ whose elements are initialized as follows:

$$a_{q_1,q_2,..,q_k} = 1 \tag{4}$$

where $\{g_{q_1}, g_{q_2}, ..., g_{q_k}\} \ni v_j$ and $d(v_j) = k, \forall j \in \{1, ..., n\}$. Note that this tensor is also symmetric, and rest of the elements in the tensor are zeros. Again, we define the lexicographically ordered index set for *dual* hyperedges (vertices in the *original* hypergraph):

$$\begin{aligned} \mathcal{Q}^k = \big\{\mathbf{q}|\mathbf{q} = (q_1, q_2, .., q_k) \text{ where } v_j \in \{g_{q_1}, g_{q_2}, ..., g_{q_k}\}, \\ \forall v_j \in V \text{ s.t. } |d(v_j)| = k \text{ and } q_1 < q_2 < ... < q_k\big\}, \end{aligned} \tag{5}$$

and we have different sets $\{\mathcal{Q}^k\} \forall k \in \{d_{min}, .., d_{max}\}$ (d_{min} and d_{max} are the maximum and the minimum vertex degree in the original hypergraph). Again, notice that $\mathcal{Q}^k$ contains unique (non-repetitive) indexes as there is only a single $\mathbf{q}$ corresponding to each of the different dual hyperedge (vertex in the original hypergraph) i.e. $v_i \in V$. Consequently, we have $|\mathcal{Q}^k| = |\{v_i : d(v_i) = k\}|$.

To realize our aim of learning node or hyperedge embeddings we perform Symmetric Tensor Decomposition [9], in a manner similar to [16], but jointly across different cardinality hypergraph tensors (for node embeddings) or dual tensors (for hyperedge embeddings). Specifically, for the hyperedge embeddings we consider the following optimization formulation:

$$f(\lambda, \mathbf{Z}) = \sum_{k=\alpha_1}^{\alpha_2} \left\| \mathcal{A}^k_{\mathbf{dual}} - \mathcal{M}^k \right\|^2 = \sum_{k=\alpha_1}^{\alpha_2} k! \left(\sum_{\mathbf{q} \in \mathcal{Q}^k} \left(a^k_{\mathbf{q}} - \mathcal{M}^k_{\mathbf{q}} \right)^2 \right) \tag{6}$$

where,

$$\mathcal{M}^k = \sum_{r=1}^{d} \lambda_r \underbrace{(\mathbf{z}_r \otimes \mathbf{z}_r \otimes ... \otimes \mathbf{z}_r)}_{k \text{ times}} \equiv \sum_{r=1}^{d} \lambda_r \mathbf{z}_r^{\otimes k} \tag{7}$$

with $\otimes$ is the Kronecker product (generalized outer product, ref. [3]), $\mathbf{z}_r \in \mathbb{R}^m$, $\lambda_r \in \mathbb{R}$, $\mathbf{Z} \in \mathbb{R}^{m \times d}$ with $\mathbf{Z}(:, r) = \mathbf{z}_r$ and $a^k_{\mathbf{q}}$ is the $a_{q_1,q_2,..,q_k}$ entry of $\mathcal{A}^k_{\mathbf{dual}}$. Given the dual tensors $\mathcal{A}^k_{\mathbf{dual}}, \forall k \in \{\alpha_1, .., \alpha_2\}$ (α_1 and α_2 are the minimum and maximum cardinalities considered for the dual hyperedges), Equation 6 aims to learn symmetric decomposition $\mathcal{M}^k$ with $\mathbf{Z}$ as the matrix containing d-dimensional embeddings for the m hyperedges. Notice that the embeddings in $\mathbf{Z}$ are shared across the different order (k) decompositions $\mathcal{M}^k$. Furthermore, although there are $k!$ entries for each k cardinality hyperedge in the dual tensor $\mathcal{A}^k_{\mathbf{dual}}$, however, the summation in Equation 6 is performed over only the unique set of indexes in $\mathcal{Q}^k$. This helps us escape the actual construction of the complete dual tensors ($\mathcal{A}^k_{\mathbf{dual}}$) which otherwise may result in exponential space explosion. This idea of using only unique index sets ($\mathcal{Q}^k$) has been leveraged by both Kolda et. al. [4, 16] as well as by [22] in form of semi-norms.

We highlight that $(\mathbf{z}_r^{\otimes k})_{\mathbf{q}} = \mathbf{z}_{q_1 r}\mathbf{z}_{q_2 r}\cdots\mathbf{z}_{q_k r}$ with $\mathbf{z}_{q_p r} = \mathbf{Z}(q_p, r) \in \mathbb{R}$. Also let:

$$\delta_{\mathbf{q}} = \left(a^k_{\mathbf{q}} - \mathcal{M}^k_{\mathbf{q}}\right) = \left(a^k_{\mathbf{q}} - \sum_{r=1}^{d} \lambda_r (\mathbf{z}_r^{\otimes k})_{\mathbf{q}}\right) \tag{8}$$

Then the gradients for Equation 6 are given by:

$$\frac{\partial f(\lambda, \mathbf{Z})}{\partial \lambda_r} = -2 \sum_{k=\alpha_1}^{\alpha_2} k! \sum_{\mathbf{q}\in\mathcal{Q}^k} \delta_{\mathbf{q}} (\mathbf{z}_r^{\otimes k})_{\mathbf{q}} \tag{9}$$

$$\frac{\partial f(\lambda, \mathbf{Z})}{\partial \mathbf{z}_{jr}} = -2 \sum_{k=\alpha_1}^{\alpha_2} k!\, \lambda_r \sum_{\mathbf{q}\in\mathcal{Q}^k} \delta_{\mathbf{q}} \left(\mathbf{z}_{q_1 r}\cdots\mathbf{z}_{q_{s-1} r}\mathbf{z}_{q_{s+1} r}\cdots\mathbf{z}_{q_k r}\right), \tag{10}$$

where, $j = q_s \in \{q_1, \cdot\cdot, q_k\}$. Furthermore, following [16] we add the following penalty to address the scaling ambiguity by enforcing the following penalty and gradient:

$$p_\gamma(\mathbf{Z}) = \gamma \sum_{r=1}^{d} (\mathbf{z_r}^\intercal \mathbf{z_r} - 1)^2 \quad , \quad \frac{\partial p_\gamma}{\partial \mathbf{z_r}} = 4\gamma(\mathbf{z_r}^\intercal \mathbf{z_r} - 1)\mathbf{z_r} \tag{11}$$

Another important issue that requires attention is that the optimization function $f(\lambda, \mathbf{Z})$ only considers dual hyperedges of cardinalities within $\{\alpha_1, .., \alpha_2\}$. As we will discuss in section 4, due to scalability reasons most likely $\{\alpha_1, .., \alpha_2\} \subset \{c_{min}, ..., c_{max}\}$. Given this setting, it is, therefore, possible that some hyperedges (in the original hypergraph) might not contain any vertex, which has a degree in the range $\{\alpha_1, .., \alpha_2\}$. Cold start issues, thus, arise for such hyperedges and to remedy that we leverage the auxiliary information in the form of the hypergraph structure. We, therefore, introduce the following penalty and gradient:

$$p_\eta(\mathbf{Z}) = \eta \sum_{r=1}^{d} \mathbf{z}_r^\intercal \mathbf{L_{dual}} \mathbf{z}_r \quad , \quad \frac{\partial p_\eta}{\partial \mathbf{z_r}} = 2\eta \mathbf{L_{dual}} \mathbf{z}_r \tag{12}$$

where $\mathbf{L_{dual}}$ is the dual hypergraph laplacian (see detail in Appendix A). This topology smoothing constraint allows the diffusion of latent embeddings from the *not* cold-start vertices to nearby cold-start vertices, for the later to achieve meaningful non-zero embeddings. Notice that the above laplacian contains the entire hypergraph dual structure, therefore, allowing information exchange (1) across different cardinality hyperedges and (2) also between hyperedges that are not suffering from a cold start. It would be interesting to develop other laplacians that (a) only affect the embeddings of the cold-start hyperedges, (b) only allow hyperedges of the same cardinality to affect each other's embeddings and (c) employ hasse diagram based laplacian [20] which explicitly models the cardinality hierarchy.

Algorithm 1 HTD $(d, m, \alpha_1, \alpha_2, \gamma, \eta, \mathcal{A}^k_{\mathbf{dual}} \forall k \in \{\alpha_1, .., \alpha_2\}, \mathbf{L_{dual}})$

```
1: randomly initialize $\mathbf{Z} \in \mathbb{R}^{m \times d}$ and $\lambda \in \mathbb{R}^d$
2: repeat
3:     for $r = 1$ to $d$ do
4:         Update $\lambda_r$ using Equation 9
5:         for $j = 1$ to $m$ do
6:             Update $\mathbf{z}_{jr}$ using Equation 10
7:         end for
8:         Update $\mathbf{z}_r$ using Equation 11
9:         Update $\mathbf{z}_r$ using Equation 12
10:    end for
11: until fit criteria achieved or max. # of iterations exceeded
12: return $\mathbf{Z}$
```

Putting it all together, the complete optimization objective function is:

$$f_{tot}(\lambda, \mathbf{Z}) = f(\lambda, \mathbf{Z}) + p_\gamma(\mathbf{Z}) + p_\eta(\mathbf{Z}) \tag{13}$$

where the choice of γ is the weight of the penalty on the norm of the columns of $\mathbf{Z}$ and the choice of η determines the penalty on the extent of smoothness (similarity) between the embeddings of hyperedges within neighborhood of each other. We call the algorithm that optimizes the above objective (Equation 13) as **Hypergraph-Tensor-Decomposition (HTD)** (Algorithm 1).

Notice that till now, we have described the process of learning hyperedge embeddings. The same process can also be used for obtaining vertex embedding by calling **HTD** $(d, n, \beta_1, \beta_2, \gamma, \eta, \mathcal{A}^k_{\mathbf{hyp}} \forall k \in \{\beta_1, .., \beta_2\}, \mathbf{L_{hyp}})$. Parameters β_1, β_2 are the limits for the hyperedge cardinalities considered in Equation 6 and $\mathbf{L_{hyp}}$ is the hypergraph laplacian (see Appendix A). We shall jointly refer to the embeddings achieved for nodes and hyperedges via the above tensor decomposition techniques as **t2v**. Lastly, we would like to highlight that the tensors that we have employed are super-symmetric and hence able to capture distribution over sets rather than sequences. But in general, we can employ a k-way tensor, which is not symmetric to even capture sequence. In this sense, tensors are more general purpose.

4 Experiments

4.1 Dataset Description

In this paper, we consider both real-world as well as synthetic datasets. For the former, we make use of five popular real-world datasets from the UCI Machine Learning Repository (http://archive.ics.uci.edu/ml). The five selected datasets have data points whose feature vectors contain mostly boolean-valued features. (From each of the datasets, we removed the very few non-boolean valued features.) We then consider each data point (sample) as a hyperedge with features as vertices. All the features (vertices) which have value one for a given sample (hyperedge) are considered vertices of this sample hyperedge. In short, we treat the data matrix (sample-feature mapping) as the hypergraph incidence matrix (hyperedge-vertex mapping). Below we describe the five datasets:

Data	**Hyperedges** (m)	**Vertices** (n)	**Max. Cardinality**	**Avg. Cardinality**	**Max. Vertex Degree**	**Avg. Vertex Degree**
zoo	101	15	10	6.53	83	44
voter	432	16	13	7.91	272	213.69
autism-child	291	14	13	7.46	217	155
autism-adolo	103	14	12	7.59	82	55.86
autism-adult	681	14	13	5.81	504	282.5
synthetic	7020	80	6	3.3	330	290.3

Table 1: Hypergraph Statistics for various Datasets

1. **zoo**: In this dataset, there are several animals each described with a set of boolean attributes like, for example, does it have a feather, or is it airborne. There are several classes of animals, and the aim is to classify animals correctly into its class.
2. **voter**: In this, the aim is to classify congressman as democrat versus republican based on 16 key votes, where each vote is boolean (yea or nay). Each congressmen's hyperedge contains on "yay" vertices.
3. **autism-child, autism-adolo, autism-adult**: These three datasets contain psychological evaluation on a cohort of children, adolescents, and adults, respectively, for classification into having ASD disorder or not. Attributes are boolean item responses to behavioral questions. We treat each item (psychological evaluation question) as a vertex, and with each positively responded item (vertex) becomes part of the corresponding individual's hyperedge.

For generating **synthetic** hypergraphs, we employ the recently proposed hypergraph stochastic block-model (**hSBM**) [14] which are generalization of the traditional stochastic graph model to a hypergraph setting. This model is fairly straightforward. Here we describe a slightly augmented process as we need labels for hyperedges (rather than partition labels for vertices in hSBM). We start with a set of vertices and divide them into two equal sets (we restrict ourselves to only two vertex partitions but can be easily extended for more). We then randomly generate hyperedges between any vertices with probability p_{inter} and within vertices in the same partition with probability p_{intra}. In order to generate clusters or communities one chooses $p_{inter} < p_{intra}$, resulting in more dense connections within each partition rather than across partitions. For a given cardinality c, we set our probability vector $\mathbf{p}(c) = [p_{inter}, p_{intra}] = [5, 40] \cdot (\log n / n^{(c-1)})$, where n is the number of vertices and order $O(K \cdot \log n / n^{(c-1))}$ is recommended to realize sparse regime i.e. $O(n \log n)$

hyperedges. The particular value of constant K is chosen to realize not too sparse hypergraphs and a sufficiently high number of higher-order hyperedges. We realize 25 hypergraphs using the above process with roughly $15n \log n$ to $25n \log n$ hyperedges. For our experiments we restrict to $[2, 6]$ cardinality hyperedges (average statistics shown in Table 1). We then label hyperedges into three different classes: those consisting of only vertices from first vertex partition, those from an only second partition, and those with a mix of vertices from both partitions. The aim, in this case, is to solve this 3-class classification problem.

4.2 Evaluation Methodology and Experimental Setup

Methods Compared We compare hyperedge embeddings obtained from our proposed method with several baselines. Both the baseline, as well as proposed approaches, can output hyperedge embeddings in two different ways:

1. Each method can directly output hyperedge embeddings. Our proposed approach gives direct hyperedge embedding using the dual tensor as the input. We refer to these embeddings as **t2v-dual**. Similarly, we have two baselines that give hyperedge embeddings directly via eigen-value decomposition of line graph laplacian (herein **e2v-line**) and that of proxy dual hypergraph laplacian (herein **e2v-dual**). Refer to appendix A for details.
2. We can use these methods to output vertex embeddings first and then combine the vertex embeddings of the vertices within a hyperedge. We evaluate only two kinds of combinations - summing the vertex embeddings (**sum**) or taking the mean of vertex embeddings (**mean**). (Note the later is hyperedge cardinality dependent.) We refer hyperedge embeddings obtained from our proposed approach in this manner by first obtaining vertex embeddings via decomposing hypergraph tensor, as **t2v**. Similarly, for the two baselines, we can first obtain vertex embeddings via eigen-value decomposition of graph lapalacian and proxy hypergraph laplacian, which can then be combined to get hyperedge embeddings referred to as **e2v** and **e2v-hyp**, respectively. Refer to appendix A for details.

We therefore, have two different comparisons. First is between embeddings obtained from **e2v-line** and **e2v-dual** with **t2v-dual** embeddings. Second, is between **e2v** and **e2v-hyp** with **t2v**. The later comparison happens further separately for both **sum** and **mean** combining strategies.

Evaluation Tasks and Setup As described in Section 4.1, each dataset has a hypergraph and a corresponding classification task associated with it. We first obtain the hyperedge embeddings for the various datasets. The hyper-parameters specific to the proposed tensor method (γ and η) were determined using grid searches over the search spaces: $\gamma = [0.01, 0.5, 5]$ and $\eta = [0.01, 0.5, 5, 10]$. Also the cardinality inputs $\alpha_1 = \beta_1 = 2$ and $\alpha_2 = \beta_2 = 8$. Further, the hyper-parameter of dimension (d) which is common to all the methods is determined by grid search over: $d = [8, 16, 32, 64]$. 5-fold cross-validation was used to determine all the best hyperparameters.

The obtained hyperedge embeddings for a given dataset are utilized for the hyperedge classification task associated with the dataset. For each classification task, we perform several evaluation runs. In each run, we randomly choose 30% of hyperedges as the test set and train logistic regression classifier using the remaining 70% training hyperedges. We chose the Area Under Curve (AUC) as the evaluation metric (the higher, the better). We take average AUC score across five runs as the final AUC. We performed Logistic regression with l_2-norm regularization whose hyper-parameter was chosen by 5-fold cross-validation using a grid search.

4.3 Results and Discussion

As detailed in prior section 4.2, for each dataset we have two different sets of comparison based on two different ways of obtaining hyperedge embedding: (a) directly obtaining hyperedge embeddings or (b) obtained by aggregating (sum or mean) vertex embeddings. For case (a) we compare **t2v-dual** versus **e2v-line**/**e2v-dual** and for case (b) compare **t2v** versus **e2v**/**e2v-hyp**. Results are reported in Table 2 for various datasets. We refer to methods with **t2v-** prefix jointly as "**t2v-** methods". Similarly, "**e2v-** methods" for all methods with **e2v-** prefix.

One of our central hypothesis is that embeddings obtained from methods which try to retain the higher-order information within hypergraphs are better than those methods which do not. We observe from Table 2 (the best scores for each row are highlighted in bold) that tensor-based methods (**t2v-** methods) consistently outperform graph-based methods (**e2v-** methods), if not worse. This observation supports our hypothesis that tensor-based models, which preserve the joint information within a hyperedge, indeed, are better models for hypergraph representations.

Also while comparing between the two vertex-embedding aggregation functions: sum and average, we observe (see "sum" and "average" rows in Table 2 across **e2v**, **e2v-hyp** and **t2v**) that in most of the cases hyperedge-embeddings obtained via summation of vertex-embeddings, perform better and in some cases those obtained via averaging are performing poorly, especially for **e2v** (like synthetic and autism datasets). Although, summation based aggregation function is performing better in several cases, in a couple of cases, like voter and zoo datasets, its not the case. This observation highlights the fact that the choice of aggregation function is itself a hyper-parameter, which requires tuning in the case of vertex-embedding based methods and a critical drawback. Direct hyperedge-embedding methods, on the other hand, are not subject to any such choices.

Lastly, we note that summation based **t2v** is consistently at least as good as **t2v-dual**, except for autism-adolo and synthetic datasets where it outperforms. Notice that **t2v** employs hypergraph tensors, and the number of non-zeros in them is proportional to the number of hyperedges and **t2v-dual** uses dual-tensor whose non-zeros are proportional to the number of vertices. Given that in all of our datasets, the number of hyperedges is much more than the number of vertices, the dual-tensor becomes very sparse and possibly explains the relatively weaker performance of **t2v-dual**. (*Remark*: But we still might choose to prefer **t2v-dual** because of the criticism we mentioned in the last paragraph.)

		Eigen Decomp.		**Tensor Decomp.**
Dataset	**Embed. Combination**	**graph (e2v)**	**proxy hyp. (e2v-hyp)**	**hyp. tensor (t2v)**
Zoo	**Node Embed Sum**	0.42	0.40	**0.83**
	Node Embed Average	0.60	0.40	**0.78**
		line graph (e2v-line)	**proxy dual (e2v-dual)**	**dual tensor (t2v-dual)**
	Only Hyperedge Embed	0.52	0.60	**0.83**

		Eigen Decomp.		**Tensor Decomp.**
Dataset	**Embed. Combination**	**graph (e2v)**	**proxy hyp. (e2v-hyp)**	**hyp. tensor (t2v)**
Voter	**Node Embed Sum**	0.94	0.94	**0.96**
	Node Embed Average	0.95	0.94	**0.96**
		line graph (e2v-line)	**proxy dual (e2v-dual)**	**dual tensor (t2v-dual)**
	Only Hyperedge Embed	0.95	0.95	**0.96**

		Eigen Decomp.		**Tensor Decomp.**
Dataset	**Embed. Combination**	**graph (e2v)**	**proxy hyp. (e2v-hyp)**	**hyp. tensor (t2v)**
Autism Child	**Node Embed Sum**	0.99	0.98	**0.99**
	Node Embed Average	0.73	**0.98**	**0.98**
		line graph (e2v-line)	**proxy dual (e2v-dual)**	**dual tensor (t2v-dual)**
	Only Hyperedge Embed	0.97	0.71	**0.99**

		Eigen Decomp.		**Tensor Decomp.**
Dataset	**Embed. Combination**	**graph (e2v)**	**proxy hyp. (e2v-hyp)**	**hyp. tensor (t2v)**
Autism Adolescent	**Node Embed Sum**	0.90	0.91	**0.95**
	Node Embed Average	0.65	0.90	**0.91**
		line graph (e2v-line)	**proxy dual (e2v-dual)**	**dual tensor (t2v-dual)**
	Only Hyperedge Embed	0.86	0.65	**0.92**

		Eigen Decomp.		**Tensor Decomp.**
Dataset	**Embed. Combination**	**graph (e2v)**	**proxy hyp. (e2v-hyp)**	**hyp. tensor (t2v)**
Autism Adult	**Node Embed Sum**	**1.00**	**1.00**	**1.00**
	Node Embed Average	0.76	**1.00**	**1.00**
		line graph (e2v-line)	**proxy dual (e2v-dual)**	**dual tensor (t2v-dual)**
	Only Hyperedge Embed	0.98	0.75	**1.00**

		Eigen Decomp.		**Tensor Decomp.**
Dataset	**Embed. Combination**	**graph (e2v)**	**proxy hyp. (e2v-hyp)**	**hyp. tensor (t2v)**
Synthetic	**Node Embed Sum**	0.94	0.94	**0.99**
	Node Embed Average	0.52	0.56	**0.99**
		line graph (e2v-line)	**proxy dual (e2v-dual)**	**dual tensor (t2v-dual)**
	Only Hyperedge Embed	**0.94**	**0.94**	**0.94**

Table 2: Classification AUC Scores of tensor methods compared to baselines

We also note that the difference between **t2v-** methods and the **e2v-** methods is not always similar. For example, this difference is much higher in the zoo dataset as compared to others. Observations such as this and the one described in the last paragraph, points us to several intuitive questions like what kinds of datasets and their hypergraph properties (avg. degree, size, and others) affect the choice of method? Or when would we prefer working using graph methods, and tensor methods might be overkill? Or employ dual-tensor versus hypergraph-tensor? These questions require both theoretical understanding as well as extensive experimentation using a larger variety of data, and hence, we would like to take this as a separate future work. The focus of this work is to introduce higher-order information retaining methodology.

5 Conclusion

In this paper, we have proposed a tensor-based algebraic method to generate representations for both hyperedges as well as hypergraph nodes while pointing out the higher-order information in the hypergraph structure. We introduce the concept of dual tensors corresponding to the hypergraph dual. We then propose a novel higher-order information retaining approach of using factors from the joint decomposition of k-way tensors corresponding to k-uniform sub-hypergraphs, as generic node & hyperedge representations. We demonstrate that our method outperforms several graph-based baselines in terms of accuracy. We, therefore, argue that our advanced tensor methods are principally suited for hypergraphs (and consequently also for graphs) while sustaining accuracy and efficiency.

A Appendix

Hyperedge Embeddings Using Spectral Methods These set of methods extract embeddings as the eigenvectors associated with Laplacian matrices corresponding the following four adjacency matrices. **(1)** Adjacency matrix associated with a hypergraph [24], is defined as: $\mathbf{A_{hyp}} = (\mathbf{H}^T\mathbf{W_e}\mathbf{H} - \mathbf{D_v})$ and laplacian: $\mathbf{L_{hyp}} = (\mathbf{I} - \mathbf{D_v}^{-1/2}\mathbf{A_{hyp}}\mathbf{D_v}^{-1/2})$ where $\mathbf{W_e}$ ($\mathbf{W_e}(i,i) = R(g_i)$) is a diagonal matrix containing the weights of each hyperedge and $\mathbf{D_v}$ is a diagonal matrix containing the degree of each vertex **(2)** $\mathbf{A_{graph}} = \mathbf{H}^T\mathbf{W_e}\mathbf{H}$ is the weighted graph associated with the hypergraph ($\mathbf{A_{hyp}}$) and its laplacian: $\mathbf{L_{graph}} = (\mathbf{I} - \mathbf{D_v}^{-1/2}\mathbf{A_{graph}}\mathbf{D_v}^{-1/2})$ **(3)** we consider the following adjacency matrix $\mathbf{A_{line}} = \mathbf{H}\mathbf{H}^T$ associated with what we refer to as the *line hypergraph*. This inverted hypergraph is a graph (unlike the dual which is a hypergraph) and there is an edge between two nodes if the hyperedges corresponding to the nodes in the original hypergraph have nodes in common. Weight of this edge is the number of common nodes. Its laplacian is: $\mathbf{L_{line}} = (\mathbf{I} - \mathbf{D_v}^{-1/2}\mathbf{A_{line}}\mathbf{D_v}^{-1/2})$ **(4)** The adjacency matrix associated with the hypergraph dual is: $\mathbf{A_{dual}} = (\mathbf{H_{dual}}^T\mathbf{W_v}\mathbf{H_{dual}} - \mathbf{D_e}) = (\mathbf{H}\mathbf{W_v}\mathbf{H}^T - \mathbf{D_e})$ where $\mathbf{W_v}$ is a diagonal matrix containing the weights of each node and $\mathbf{D_e}$ is a diagonal matrix containing the degree of each hyperedge. We assume no weights on the nodes and take $\mathbf{W_v} = \mathbf{I}$. Its laplacian is: $\mathbf{L_{dual}} = (\mathbf{I} - \mathbf{D_e}^{-1/2}\mathbf{A_{dual}}\mathbf{D_e}^{-1/2})$. We get vertex embeddings using of **(1)** and hyperedge embedding using **(3)** via eigen-decomposition, and we refer to as **e2v** and **e2v-line**, respectively. Similarly, we get another pair of vertex embeddings using **(2)** and hyperedge embedding using **(4)** which we refer as **e2v-hyp** and **e2v-dual**, respectively. Note **e2v** and **e2v-hyp** are used to denote the hyperedge embeddings obtained by combining vertex embeddings and not the vertex embeddings themselves.

References

[1] Agarwal, S., Branson, K., Belongie, S.: Higher order learning with graphs. In: Proc. of the 23rd Intl. Conf. on Machine learning (ICML). pp. 17–24. ACM (2006)

[2] Ahmed, I., Brown, C., Pilny, A., Cai, D., Ada, Y.A., Poole, M.S.: Identification of groups in online environments: The twist and turns of grouping groups. In: Third Intl. Conf. on Social Computing (SocialCom). pp. 629–632. IEEE (2011)

[3] Bader, B.W., Kolda, T.G.: Efficient matlab computations with sparse and factored tensors. SIAM Journal on Scientific Computing **30**(1), 205–231 (2007)

[4] Ballard, G., Kolda, T., Plantenga, T.: Efficiently computing tensor eigenvalues on a gpu. In: Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on. pp. 1340–1348. IEEE (2011)

[5] Bengio, Y., Bengio, S.: Modeling high-dimensional discrete data with multi-layer neural networks. In: Adv. in Neural Info. Processing Systems. pp. 400–406 (2000)

[6] Berge, C.: Hypergraphs: combinatorics of finite sets, vol. 45. Elsevier (1984)

[7] Cai, H., Zheng, V.W., Chang, K.C.C.: A comprehensive survey of graph embedding: Problems, techniques and applications. arXiv:1709.07604 (2017)

[8] Chung, F.R., Graham, R.L., Frankl, P., Shearer, J.B.: Some intersection theorems for ordered sets and graphs. Journal of Combinatorial Theory, Series A **43**(1), 23–37 (1986)

[9] Comon, P., Golub, G., Lim, L.H., Mourrain, B.: Symmetric tensors and symmetric tensor rank. SIAM Journal on Matrix Analysis and App. **30**(3), 1254–1279 (2008)

[10] Contractor, N.: Some assembly required: leveraging web science to understand and enable team assembly. Phil. Trans. R. Soc. A **371**(1987), 20120385 (2013)

[11] Cooper, J., Dutle, A.: Spectra of uniform hypergraphs. Linear Algebra and its Applications **436**(9), 3268–3292 (2012)

[12] Deshpande, M., Karypis, G.: Item-based top-n recommendation algorithms. ACM Transactions on Information Systems (TOIS) **22**(1), 143–177 (2004)

[13] Estrada, E., Rodriguez-Velazquez, J.A.: Complex networks as hypergraphs. arXiv preprint physics/0505137 (2005)

[14] Ghoshdastidar, D., Dukkipati, A.: A provable generalized tensor spectral method for uniform hypergraph partitioning. In: International Conference on Machine Learning (ICML). pp. 400–409 (2015)

[15] Klamt, S., Haus, U., Theis, F.: Hypergraphs and cellular networks. PLoS computational biology **5**(5), e1000385 (2009)

[16] Kolda, T.G.: Numerical optimization for symmetric tensor decomposition. Mathematical Programming **151**(1), 225–248 (2015)

[17] Qi, L.: Eigenvalues of a real supersymmetric tensor. Journal of Symbolic Computation **40**(6), 1302–1324 (2005)

[18] Sharma, A.: Hypergraph Analytics: Modeling Higher-order Structures and Probabilities. Ph.D. thesis, University of Minnesota (2020)

[19] Sharma, A., Kuang, R., Srivastava, J., Feng, X., Singhal, K.: Predicting small group accretion in social networks: A topology based incremental approach. In: Intl. Conf. on Advances in Social Networks Analysis and Mining (ASONAM). pp. 408–415. IEEE (2015)

[20] Sharma, A., Moore, T.J., Swami, A., Srivastava, J.: Weighted simplicial complex: A novel approach for predicting small group evolution. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 511–523. Springer (2017)

[21] Sharma, A., Srivastava, J.: Group analysis using machine learning techniques. In: Group Processes, pp. 145–180. Springer (2017)

[22] Shashua, A., Zass, R., Hazan, T.: Multi-way clustering using super-symmetric non-negative tensor factorization. Computer Vision–ECCV 2006 pp. 595–608 (2006)

[23] Tian, Z., Hwang, T., Kuang, R.: A hypergraph-based learning algorithm for classifying gene expression and arraycgh data with prior knowledge. Bioinformatics **25**(21), 2831–2838 (2009)

[24] Zhou, D., Huang, J., Schölkopf, B.: Learning with hypergraphs: Clustering, classification, and embedding. In: Advances in neural information processing systems. pp. 1601–1608 (2006)