HYPEREDGE2VEC: DISTRIBUTED REPRESENTATIONS FOR HYPEREDGES

Anonymous authors
Paper under double-blind review

ABSTRACT

Data structured in form of overlapping or non-overlapping sets is 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 sets; and for various types of analysis 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 important task in the fields of machine learning and statistics, research on "set-like" data still remains less explored. Relationships between pairs of elements can be modeled as edges in a graph. However, modeling relationships that involve all members of a set, a hyperedge is a more natural representation for the set. In this work, we focus on the problem of embedding hyperedges in a hypergraph (a network of overlapping sets) to a low dimensional vector space. We present a number of new models, some of which extend existing node-level embedding models to the hyperedge-level, as well as other novel methods that directly work on the hypergraph topology. We propose both probabilistic as well as tensor-based models to leverage the hypergraph structure. Our central focus is to highlight the connection between hypergraphs (topology), tensors (algebra) and probabilistic models. The performance of these models is evaluated with a network of social groups and a network of word phrases. Our results demonstrate the effectiveness of our approach.

1 INTRODUCTION

The Internet has made it possible to record as well as share data about various entities, which may be related or interacting in a sophisticated manner. Complex networks Strogatz (2001) are often used to study such multi-faceted interactions among objects. Among the various models for complex networks, graphs are the most commonly used representations (Strogatz, 2001). Graphs consist of set of nodes representing the entities under study, and edges between node-pairs capture the relationship between the entity pair. However, data in many applications has more complex structure, involving relations between multiple entities simultaneously, unlike simple pair-wise relationships modeled by graphs. In fact, such data is more abundantly found in the real world than has been usually studied (Estrada & Rodriguez-Velazquez, 2005). Figure 1 shows such examples for networks of groups, sentences, and item sets.

Hypergraph (Berge, 1984), which is a generalization of graphs, is a popular model to naturally capture higher-order relationships between sets of objects (Figure 2). A specialization of this model is the Simplicial complex (Munkres, 1984) (Figure 2), in which additionally each hyperedge has the subset closure property, i.e., each subset of hyperedge is also a valid hyperedge. Within machine learning, algorithms guided by the structure of such higher order networks (Zhou et al., 2006) have found applications in a variety of domains (Gao et al., 2013; Tian et al., 2009; Li & Li, 2013; Sharma et al., 2015). More recently, the success of feature (or representation) learning in natural language processing or NLP (Mikolov et al., 2013a) has stimulated interest in its application for network analysis by learning low dimensional representation for each network node (Grover & Leskovec, 2016; Tang et al., 2015b; Perozzi et al., 2014). These learned representations can then be provided as input to supervised learning algorithms to perform various prediction tasks related to link prediction or node classification. However, this new line of research is limited to simple graphs, as they extract

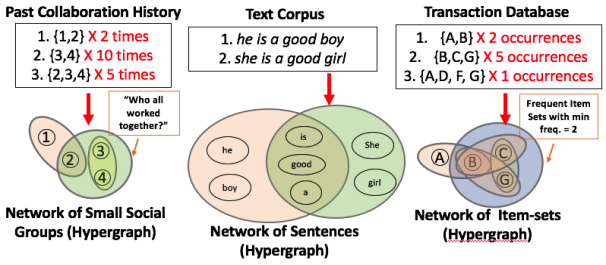

Figure 1: Example illustrating "set-like" structures from various domains.

features for the graph nodes or edges by designing an objective function that only takes into account the dyadic relationship modeled by a simple edge.

In this paper we focus on developing methods to learn representations for not just nodes but also for set of nodes (hyperedges or simplex faces). The proposed method takes into account the higher-order network structure while learning the embeddings. These learned features (embeddings) can then be employed by a supervised algorithm to perform various predictive tasks pertaining to the set of nodes. For example performance prediction of a team (set of individuals) engaged in a collaborative task or sentiment analysis of a sentence (set of words).

Traditional graph embedding methods such as spectral clustering have been extended to hypergraphs (Zhou et al., 2006). However, these embeddings are learned for nodes, not for hyperedges, although the algorithm takes into account the hypergraph structure. Also, (Agarwal et al., 2006) have criticized that such representations can be learned by constructing graphs, which are proxies for the hypergraph structure. In fact, we are unaware of any work within network analysis that considers embeddings at the set-level. On the contrary, in the natural language processing (NLP) literature, methods for learning embeddings for higher-order entities like sentences and paragraphs have been proposed (Le & Mikolov, 2014), but they do not consider the language network structure. Our paper, therefore, aims to fill this gap by learning representations at the set-level itself, while taking into account the topological connectivity between the various sets in a hypergraph.

Another distinguishing feature of our work is that we leverage the existing structure within the network data. Recent attempts along these lines in network literature (Grover & Leskovec, 2016; Tang et al., 2015b; Perozzi et al., 2014) argue that language models have ready-made context in the form of sentences or paragraphs to train the model, which are not available in networks, and therefore, they propose different ways to generate this context. By contrast, we focus on networks where the context is already present, e.g. collaboration networks where collaborative teams are hyperedges, or language hyper-networks where sentences are hyperedges.

We propose several methods, some of which use the existing node-level method to generate set-level embeddings, and other novel models optimization criterion which directly learn hyperedge-level representations. We have experimented with social group dataset (network of teams as a hypergraph) from an online game for team (hyperedge) performance prediction as well as language networks (sentence/phrase hypergraphs) for phrase-level sentiment analysis. Our results demonstrate the effectiveness of our approach. In summary the main contributions of our paper are:

- We propose several methods to generate higher-order representations for both hyperedges (set-level) and hypergraph nodes (that also take into account the hypergraph structure).
- We build feature learning models that leverage the existing structure present in network data as context.
- We highlight the connection between higher order tensor methods and higher order probabilistic models in NLP.
- We argue that tensor/matrix based methods can be used to construct embeddings which have a natural graph/hypergraph theoretic interpretation. Therefore, the embedding can be better interpreted unlike the neural network based black-box approaches.
- We also propose the novel concept of a dual tensor corresponding to the hypergraph dual. In general, we consider using factors from decomposition of N-way tensors corresponding

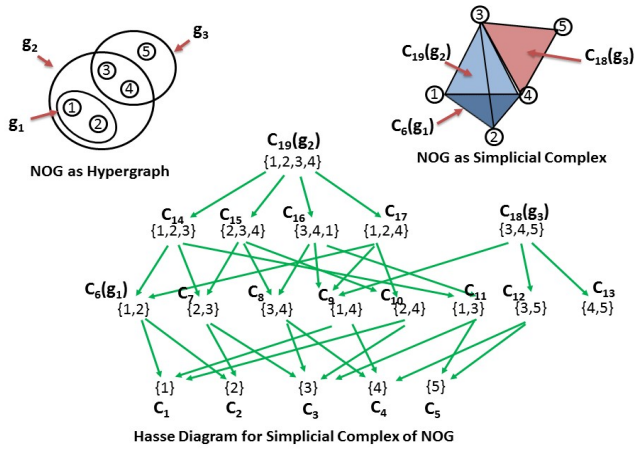

Figure 2: Example illustrating *network of groups (NOG)* which is the *hypergraph (HG)* (top left), the corresponding *simplicial complex (SC)* (top right) and the *hasse diagram (HD)* (bottom) corresponding to the simplicial complex, for a scenario where the actors $\{1,2,3,4,5\}$ have collaberated in past as groups: $g_1 = \{1, 2\}$, $g_2 = \{1, 2, 3, 4\}$ and $g_3 = \{3, 4, 5\}$.

to hypergraph, for node & hyperedge embeddings, as a novel approach. We are unaware of any such works or applications employing this approach.

Following is the outline for the rest of the paper. Section 5 provides an overview of the related literature followed by Section 2 where we describe the problem definition and statement. In Section 3 we describe in detail the various methods proposed in this paper. Section 4 describes the datasets, experimental tasks & settings for the models.

2 PRELIMINARIES

In this paper we consider the 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 sets defined over these elements, denoted by $G = \{g_1, g_2, ..., g_m\}$, where $g_i \subseteq V$ represents the ith 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 Berge (1984), 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 have a graph associated with the hypergraph, $N_a = (V, E)$, where $E = \{e_1, .., e_w\}$ are the dyadic edges defined over V. The adjacency matrix $\mathbf{A} \in \{0, 1\}^{|V|\times|V|}$ for N_a has elements $\mathbf{A}(v_1, v_2) = 1$ for $v_1 \in V$, $v_2 \in V$, such that $\exists i, \{v_1, v_2\} \subseteq g_i$ else 0.

2.1 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 the following subsections, we describe our methods for hyperedge embedding. As we had mentioned before there are several methods to learn representation of nodes in a graph. Given that a hyperedge is a set of nodes, a natural question arises is that, can we combine the node level embeddings (learned using existing methods) within a given hyperedge to find a suitable representation of that hyperedge? However, there are large number of possible ways one can combine the node embeddings. Therefore, we should expect that a method for hyperedge embedding should learn the embeddings for hyperedges directly in a more principled manner. In fact, in each of the methods below we shall find both node as well as hyperedge level embeddings. In our experiments we will evaluate the various combinations of both types of embeddings.

3.1 HYPEREDGE2VEC USING DENOISING AUTOENCODER

An autoencoder (Bengio et al., 2009), takes an input vector $\mathbf{x} \in [0,1]^n$ and maps it to a latent representation $\mathbf{y} \in [0,1]^d$. This is typically done using an affine mapping followed by a non-linearity (more so when the input, like in our case, is binary (Vincent et al., 2010)): $f_{en}(\mathbf{x}) = s(\mathbf{W}\mathbf{x} + \mathbf{b})$, with parameters $\theta = \{\mathbf{W}, \mathbf{b}\}$. Here, $\mathbf{W}$ is a $n \times d$ weight matrix and $\mathbf{b}$ is the offset. This latent representation is reconstructed into a vector $\mathbf{z} = f_{de}(\mathbf{y}) = s(\mathbf{W}'\mathbf{y} + \mathbf{b}')$, in the input space, $\mathbf{z} \in [0,1]^n$ and parameters $\theta' = \{\mathbf{W}', \mathbf{b}'\}$. The mappings f_{en} and f_{de} are referred as the **encoder** and **decoder**, respectively. The representation $\mathbf{y}$ is learned by minimizing the following reconstruction error:

$$\theta^*, \theta'^* = \underset{\theta^*, \theta'^*}{\arg\min} \frac{1}{m} \sum_{i=1}^{m} L(\mathbf{x}(i), \mathbf{z}(i)) = \underset{\theta^*, \theta'^*}{\arg\min} \frac{1}{m} \sum_{i=1}^{m} L(\mathbf{x}(i), g_{\theta'}(f_\theta(\mathbf{x}(i)))) \tag{1}$$

where L is a loss function, which in case of binary or bit probabilities is often chosen as the *cross-entropy loss*:

$$L(\mathbf{x}, \mathbf{z}) = \sum_{j=1}^{n} [\mathbf{x}(j) \log \mathbf{z}(j) + (1 - \mathbf{x}(j)) \log(1 - \mathbf{z}(j))] \tag{2}$$

In their paper, Vincent et al. (2010) have shown that minimizing reconstruction amounts to maximizing the lower bound on the mutual information between input (x) and the representation $\mathbf{y}$. However, they have further argued (Vincent et al. (2008)) that $\mathbf{y}$ retaining information about input $\mathbf{x}$ is insufficient. They further, propose the idea that the learned representation should be able to recover (*denoising*) the original input even after being trained with corrupted input (*adding noise*). They generate the corrupted input ($\tilde{\mathbf{x}}$), using a stochastic mapping $q(\tilde{\mathbf{x}}|\mathbf{x})$. Choice of noise is usually either *Gaussian* for real inputs and *Salt-and-pepper noise* for discrete inputs. The *denoising autoencoder* then learns the representation for each input, $\mathbf{x}$, same as Equation 1, but with the following modified loss function: $L(\mathbf{x}(i), g_{\theta'}(f_\theta(\tilde{\mathbf{x}}(i))))$.

We leverage the denoising autoencoder for learning representation for j^{th} hyperedge, by treating each hyperedge as an input, $\mathbf{x} = \mathbf{H}(j,:)$. The size of this input vector for each hyperedge is n, which is the number of vertices in the hypergraph. In most natural hypergraphs, specially social networks, n can be quiet high ranging from thousands to millions or even billions (like Facebook for example). Therefore, randomly using a discrete noise like *salt-and-pepper*, might not be reasonable, as there are large number of possible permutations (as size n is large) and not all of them are related. Random addition of 1s or deletion of existing 1s from $\mathbf{x}$, amounts to randomly adding or deleting vertices to the hyperedge corresponding to $\mathbf{x}$. This might end up in new hyperedges that are completely unrelated to the given hyperedge ($\mathbf{x}$). For example, users (nodes) in a social network from completely different regions of the network suddenly form a group (hyperedge). Such anomalous scenarios rarely happen in practice and social groups evolve in a gradual fashion via simple processes (Sharma et al., 2017; 2015).

Rather, we take advantage of the hypergraph structure to systematically guide us in generating this noise. A hypergraph can be defined by its corresponding *hasse lattice* (Sharma et al., 2017). For a given hyperedge ($\mathbf{x}$), we consider the sub-lattice consisting of only those hyperedges that are a distance h from it in the complete lattice. On this sub-lattice we sample hyperedges (nodes in sub-lattice) by performing random walk starting at the given hyperedge's node (see Figure 3). Our stochastic mapping $q(\tilde{\mathbf{x}}|\mathbf{x})$ is therefore, a random walk on the sub-lattice of hyperdge ($\mathbf{x}$) containing

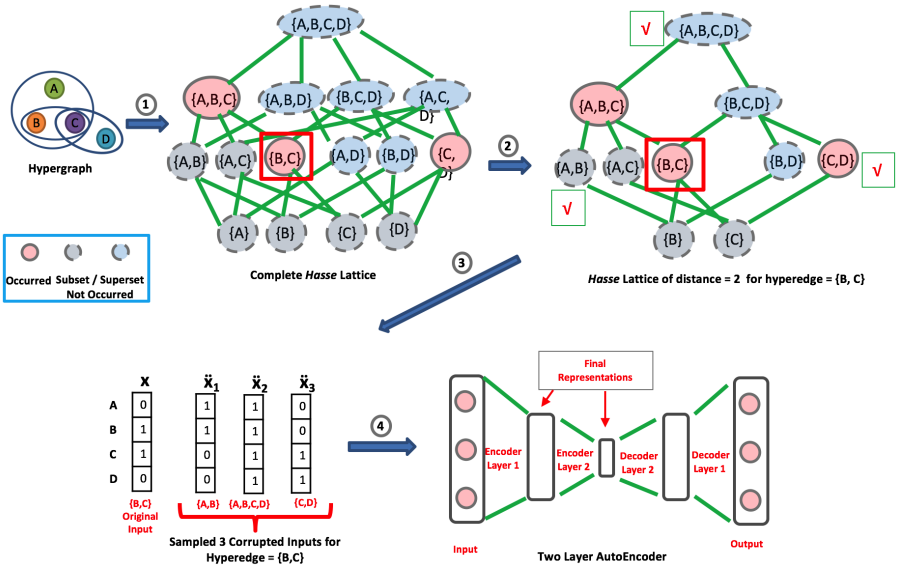

Figure 3: For the given hypergraph between four nodes (A,B,C,D) we consider the complete *hasse* lattice. For a given hyperedge {B,C} (square box) we then construct the sub-lattice made of hyperedges with distance $h = 2$ from {B,C}. We perform random walk starting from the node corresponding to hyperedge {B,C} and sample $p = 3$ hyperedges (nodes visited by the random walk; shown with a check-mark). Finally, we train the autoencoder to reconstruct the original hyperedge from these p noisy hyperedges.

hyperedges at distance h from it. Intuitively, the hyperedges coming within a reasonable distance will affect each others representations and will have more similar representations. We will refer to the hyperedge representations learned by the above autoencoder technique, as **h2v-auto**.

3.2 Hyperedge2vec Using N-way Tensor Decomposition

In this section, we develop tensor (higher-order matrices) based linear algebraic methods that learn node as well as hyperedge embedding by taking into account the joint probability over a hyperedge. 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 Cooper & Dutle (2012). Corresponding to this k-uniform hypergraph, we can define a kth order n-dimensional symmetric tensor Qi (2005) $\mathcal{A}^{\mathbf{k}}_{\mathbf{hyp}} = (a_{p_1,p_2,..,p_k}) \in \mathbb{R}^{[k,n]}$ whose elements are initialized as follows:

$$a_{p_1,p_2,..,p_k} = R(g_i) \tag{3}$$

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 value of element $a_{p_1,p_2,..,p_k}$ is invariable under any permutation of its indices $(p_1, p_2, .., p_k)$. Rest all the elements in the tensor are zeros.

In a similar manner we can also define a *dual tensor*, corresponding to *hypergraph dual* where the roles of nodes and hyperedges are interchanged. We consider all the hyperedges in the hypergraph dual that are of cardinality k. This basically corresponds to all the vertices in the original hypergraph which have a degree of 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 kth order m-dimensional symmetric dual tensor $\mathcal{A}^{\mathbf{k}}_{\mathbf{dual}} = (a_{q_1,q_2,..,q_k}) \in \mathbb{R}^{[k,m]}$ 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 $|\{r|\mathbf{H}(r,j) = 1\}| = k, \forall j \in \{1, ..., n\}$. Note that this tensor is also symmetric and rest all the elements in the tensor are zeros.

To realize our aim of learning node and hyperedge embeddings we perform joint CP Tensor Decomposition Kolda & Bader (2009) (of the tensors we just described) across different cardinality

hyperedges simultaneously. Specifically, for the node embeddings we solve the following optimization problem:

$$\min_{\widehat{\mathcal{A}^{\mathbf{k}}_{\mathbf{hyp}}}} \sum_{k=c_{min}}^{c_{max}} \|\mathcal{A}^{\mathbf{k}}_{\mathbf{hyp}} - \widehat{\mathcal{A}^{\mathbf{k}}_{\mathbf{hyp}}}\| \tag{5}$$

where,

$$\widehat{\mathcal{A}^{\mathbf{k}}_{\mathbf{hyp}}} = \sum_{r=1}^{d} \lambda_r {}_k\mathbf{u}_r^{(1)} \circ {}_k\mathbf{u}_r^{(2)} ... \circ {}_k\mathbf{u}_r^{(k)} \equiv \langle \lambda | {}_k\mathbf{U}^{(\mathbf{1})}, {}_k\mathbf{U}^{(\mathbf{2})}, ..., {}_k\mathbf{U}^{(\mathbf{k})} \rangle \tag{6}$$

with ${}_k\mathbf{u}_r^{(j)} \in \mathbb{R}^n$, $\lambda_r \in \mathbb{R}_+$, ${}_k\mathbf{U}^{(\mathbf{j})} \in \mathbb{R}^{n \times d}$, ${}_k\mathbf{U}^{(\mathbf{j})}(:, r) = {}_k\mathbf{u}_r^{(j)}$ and $j \in \{1, .., k\}$. Notice, that equation 6 is the standard CP decomposition but equation 5 is summation of reconstruction errors in different tensor decomposition for different cardinality hyperedges. Each error term learns latent factor matrices corresponding to various ${}_k\mathbf{U}^{(\mathbf{j})}$ using the empirical observed k-uniform sub-hypergraph stored in $\mathcal{A}^{\mathbf{k}}_{\mathbf{hyp}}$ tensor. However, we wish to learn common representations for all the nodes learned from different cardinality k-uniform sub-hypergraph. We therefore, wish to add an additional constraint that ${}_k\mathbf{U}^{(\mathbf{j})}$ are same for all $j \in \{1, .., k\}$ and for all $k \in [c_{min}, c_{max}]$. We add these constrains by augmenting the standard CP Decomposition into the **Hypergraph-CP-ALS** algorithm 1. Lines (4-8) is the standard CP Decomposition (see reference (Kolda & Bader, 2009) for details) and lines (9-15) is where we force all representations for all cardinalities to be average of the ${}_k\mathbf{U}^{(\mathbf{j})}$ achieved by the last decomposition that occurred in lines (4-8). We make repeated pass through the entire hypergraph (by learning via different k-uniform sub-hypergraph (line 3)) until the objective (equation 5) converges. The same algorithm 1 is used to get the hyperedge embeddings, by just passing $\mathcal{A}_{\mathbf{dual}}$ instead of $\mathcal{A}_{\mathbf{hyp}}$. We shall jointly refer to the embeddings achieved for nodes and hyperedges via the above tensor decomposition techniques as **t2v**.

Algorithm 1 Hypergraph-CP-ALS $(\mathcal{A}_{\mathbf{hyp}}, c_{min}, c_{max})$

1: randomly initialize ${}_k\mathbf{U}^{(\mathbf{j})}, \forall k \in [c_{min}, c_{max}], \forall j \in \{1, ..., k\}$
2: **repeat**
3: **for** $k = c_{min}$ to c_{max} **do**
4: **for** $j = 1$ to k **do**
5: $\mathbf{V} \leftarrow {}_k\mathbf{U}^{(\mathbf{1})\intercal} {}_k\mathbf{U}^{(\mathbf{1})} * ... * {}_k\mathbf{U}^{(\mathbf{j-1})\intercal} {}_k\mathbf{U}^{(\mathbf{j-1})} * {}_k\mathbf{U}^{(\mathbf{j+1})\intercal} {}_k\mathbf{U}^{(\mathbf{j+1})} * ... * {}_k\mathbf{U}^{(\mathbf{j})\intercal} {}_k\mathbf{U}^{(\mathbf{j})}$
6: ${}_k\mathbf{U}^{(\mathbf{j})} \leftarrow (\mathcal{A}^{\mathbf{k}}_{\mathbf{hyp}})^{(j)} ({}_k\mathbf{U}^{(\mathbf{k})} \odot ... \odot {}_k\mathbf{U}^{(\mathbf{j+1})} \odot {}_k\mathbf{U}^{(\mathbf{j-1})} \odot ... \odot {}_k\mathbf{U}^{(\mathbf{1})}) \mathbf{V}^{\dagger}$
7: normalize columns of ${}_k\mathbf{U}^{(\mathbf{j})}$ (and store norms as λ)
8: **end for**
9: $\mathbf{Z} \leftarrow \frac{1}{k} \sum_{j=1}^{k} {}_k\mathbf{U}^{(\mathbf{j})}$
10: **for** $p = c_{min}$ to c_{max} **do**
11: **for** $j = 1$ to k **do**
12: ${}_p\mathbf{U}^{(\mathbf{j})} \leftarrow \mathbf{Z}$
13: **end for**
14: **end for**
15: **end for**
16: **until** fit criteria achieved or maximum number of iterations exceeded
17: **return** $\mathbf{Z}$

We would like to highlight a few points regarding the tensor methods. The tensors that we have employed are super-symmetric and hence able to capture distribution over sets rather than sequence. 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. Another point one can observe that when we initialize the hypergraph tensor $\mathcal{A}^{\mathbf{k}}_{\mathbf{hyp}}$, we have initialized all the permutations of vertices corresponding to a given hyperedge (Eq. 3). Moreover, we intialize it by the repetition counts. This process seems to have direct correspondence with the proxy text based scheme that we developed in Section 3.3. This should give us confidence in this proxy text scheme that its more that just a heuristic. In fact the whole reason of developing a proxy text based scheme was to deliberately augment a model tailored for sequences to learn embeddings for sets as well.

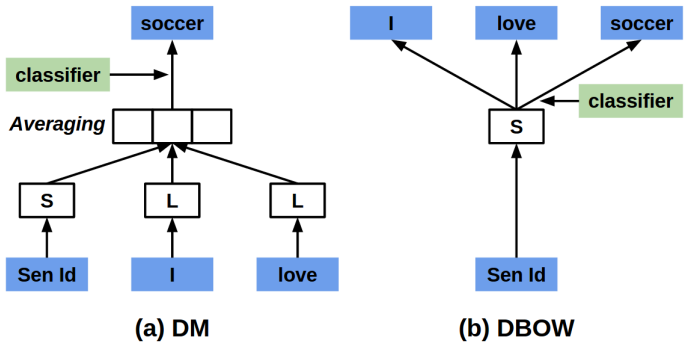

Figure 4: Distributed memory (DM) and Distributed bag of words (DBOW) versions of Sen2Vec for a sentence "I love soccer".

3.3 HYPEREDGE2VEC USING SEN2VEC

As mentioned before, the most commonly used model for studying complex interactions in networks is graphs, where each edge represents a dyadic interaction between nodes Strogatz (2001). Therefore, even if the original interaction is not dyadic like a set of researchers (3 or more) collaborating on a publication, we shall break down this joint interaction into dyadic interactions. Even though a large number of complex network data naturally occurs as hypergraphs Estrada & Rodriguez-Velazquez (2005), the popularity of "think like dyadic edges" and not like sets seems to us as a hindrance, specifically when the end aim is to model the joint distribution at the level of sets. In this paper, as it is obvious, we go by "think like set" paradigm, specifically, when the data is naturally occurring as network of sets (i.e., a hypergraph). Therefore, if we think of hyperedge as a set, a natural question arises is that are there any existing techniques that learn embeddings at set level? We, henceforth, explore the most popular discrete data on which representation learning has been applied is text data.

Recently, Le & Mikolov (2014) proposed two representation learning methods for sentences as shown in Figure 4: a) a distributed memory (**DM**) model, and b) a distributed bag of words (**DBOW**) model. In the DM model, every sentence in the dataset G is represented by a d dimensional vector in a shared lookup matrix $S \in^{|G|\times d}$. Similarly, every word in the vocabulary Ω is represented by a d dimensional vector in another shared lookup matrix $L \in^{|\Omega|\times d}$. Given an input sentence $\mathbf{v} = (v_1, v_2 \cdots v_m)$, the corresponding sentence vector from S and the word vectors from L are averaged to predict the next word in a context. More formally, let ϕ denote the mapping from sentence and word ids to their respective vectors in S and L, the DM model minimizes the following objective:

$$J(\phi) = -\sum_{t=k}^{m-k} \log P(v_t|\mathbf{v}; v_{t-k+1}, \cdots, v_{t-1}) \tag{7}$$

$$= -\sum_{t=k}^{m-k} \log \frac{\exp(\omega(v_t)^T \mathbf{z})}{\sum_i \exp(\omega(v_i)^T \mathbf{z})} \tag{8}$$

where $\mathbf{z}$ is the average of $\phi(\mathbf{v}), \phi(v_{t-k+1}), \cdots, \phi(v_{t-1})$ *input* vectors, and $\omega(v_t)$ is the *output* vector representation of word $v_t \in \Omega$. The sentence vector $\phi(\mathbf{v})$ is shared across all (sliding window) contexts extracted from the same sentence, thus acts as a distributed memory.

Instead of predicting the next word in the context, the DBOW model predicts the words in the context independently given the sentence id as input. More formally, DBOW minimizes the following objective:

$$J(\phi) = -\sum_{t=k}^{m-k} \sum_{j=t-k+1}^{t} \log P(v_j|\mathbf{v}) \tag{9}$$

$$= -\sum_{t=k}^{m-k} \sum_{j=t-k+1}^{t} \log \frac{\exp(\omega(v_j)^T \phi(\mathbf{v}))}{\sum_i \exp(\omega(v_i)^T \phi(\mathbf{v}))} \tag{10}$$

Both the methods take sentences as input and return embeddings for words as well as the sentence. To apply the Sen2Vec models directly to hyperedges, as our first method we generate a proxy sentence for each hyperedge by leveraging the contextual information in the hyperedge.

A proxy sentence $\mathbf{v}_i$ is formed for each hyperedge $g_i \in G$ as a sequence made by concatenating all the permutations of the nodes (as words) in the hyperedge and further repeating this sequence as many time this hyperedge occurred. For example, for a three node hyperedge $g_i = \{1, 4, 7\}$ which has occurred two times we make the following sentence $\mathbf{v}_i$: {1,4,7,1,7,4,7,1,4,7,4,1,4,1,7,4,7,1,1,4,7,1,7,4,7,1,4,7,4,1,
4,1,7,4,7,1} of length 12 (6 permutations times 2 occurrence). In this scheme we have permutations and repetitions. We take all the permutations hoping that it should fool the sequence based model to get a sequence independent embedding. On the other hand, repetition of the permutations acts similarly to observing the same sequence of words several times in the text corpus. Of course one can think of other alternative schemes, but the point we are trying to make is that naturally observed hyperedges captures important contextual information that can be leveraged to achieve better representations. We refer to the node-hyperedge embedding pairs resulting from **DM**, **DBOW** models as **h2v-dm** and **h2v-dbow**, respectively.

3.4 HYPEREDGE2VEC USING NODE2VEC

Grover & Leskovec (2016) propose a representation method for nodes in a graph called Node2Vec, which uses the skip-gram model Mikolov et al. (2013b) with the intuition that neighborhood nodes within a graph should have similar representations. The objective of the skip-gram model for graphs can be defined as:

$$J(\phi) = -\sum_{v \in V} \log P(N(v)|\phi(v)) \tag{11}$$

$$= -\sum_{v \in V} \sum_{n_i \in N(v)} \log P(n_i|\phi(v)) \tag{12}$$

$$= -\sum_{v \in V} \sum_{n_i \in N(v)} \log \frac{\exp(\omega(n_i)^T \phi(v))}{\sum_{x \in V} \exp(\omega(x)^T \phi(v))} \tag{13}$$

where as before, ϕ and ω denote the *input* and the *output* vector representations of the nodes. The neighboring nodes $N(v)$ form the *context* for node v. Node2Vec uses a biased random walk which adaptively combines breadth first search (BFS) and depth first search (DFS) to find the neighborhood of a node. The walk attempts to capture two properties of a graph often used for prediction tasks in networks: i) homophily and ii) structural equivalence. According to homophily, nodes in the same group or community should have similar representations. Structural equivalence suggests that nodes with similar structural roles (hub, bridge) should have similar representations. In a real-world network, nodes exhibit mixed properties.

Our aim is to find both node and hyperedge level embeddings by taking into account the hypergraph structure. For the former, we leverage the adjacency matrix associated with a hypergraph Zhou et al. (2006), which is defined as:

$$\mathbf{A_{hyp}} = \mathbf{H}^T \mathbf{W_e} \mathbf{H} - \mathbf{D_v} \tag{14}$$

where $\mathbf{W_e}$ is a diagonal matrix containing the weights of each hyperedge and $\mathbf{D_v}$ is a diagonal matrix containing the degree of each vertex. We take the weight of a hyperedge as its occurrence number, i.e. $\mathbf{W_e}(i,i) = R(g_i), \forall\{i \in \{1, ..., n\}\}$. The adjacency matrix ($\mathbf{A_{hyp}} \in^{|V| \times |V|}$) associates a weight between a pair of nodes while taking into account the weights of all the hyperedges that encompass a pair of nodes. The weighted graph associated with $\mathbf{A_{hyp}}$ in some sense serves as a proxy to the actual hypergraph structure. We can provide $\mathbf{A_{hyp}}$ as input to Node2Vec and hypothesize that the random walk over this proxy hypergraph should allow the skip-gram model to learn more meaningful node level embeddings which can be combined to construct hyperedge level embeddings. We refer to these node level embeddings as **N2V-hyp**.

We still have not met our second objective to learn hyperedge embeddings directly. We again wish to leverage Node2Vec for our purpose. However, Node2Vec only works for graphs by performing random walk over nodes. Therefore, we ask ourselves the question, that can we treat hyperedges as nodes? There can be other ways of doing so, but here we suggest two techniques. In the

first technique, we simply consider the *hypergraph dual* Berge (1984), whose incidence matrix is $\mathbf{H_{dual}} = \mathbf{H}^T$. The adjacency matrix associated with the hypergraph dual is:

$$\mathbf{A_{dual}} = \mathbf{H_{dual}}^T \mathbf{W_v} \mathbf{H_{dual}} - \mathbf{D_e} = \mathbf{H} \mathbf{W_v} \mathbf{H}^T - \mathbf{D_e} \tag{15}$$

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 $\mathbf{W_v} = \mathbf{I}$. The matrix $\mathbf{A_{dual}} \in^{|G| \times |G|}$ represents another hypergraph, but the roles of nodes and hyperedges have now switched. For example, in case of words and sentences, the hyperedges were sentences, but in the hypergraph dual the words become hyperedges and the nodes within a word's hyperedge represent all the sentences in which the word has appeared. We again give $\mathbf{A_{dual}}$ (i.e., a graph proxy for the dual hypergraph) as input to Node2Vec, but this time we get output as the embeddings for the hyperedges associated with the nodes in dual. We refer to these hyperedge embeddings as **h2v-dual**.

In the second technique, we consider the following adjacency matrix $\mathbf{A_{inv}} \in^{|G| \times |G|}$:

$$\mathbf{A_{inv}} = \mathbf{H} \mathbf{H}^T \tag{16}$$

associated with what we refer to as the *inverted 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. We again give $\mathbf{A_{inv}}$ as input to Node2Vec to get embeddings for the hyperedges associated with the nodes in inverted hypergraph. We refer to these hyperedge embeddings as **h2v-inv**.

3.5 Hyperedge2vec Using Spectral Embeddings

These set of methods extract embeddings as the eigenvectors associated with Laplacian matrices corresponding to the various adjacency matrices discussed in the previous section. We consider the following graph Laplacians:

$$\mathbf{L_{graph}} = \mathbf{I} - \mathbf{D_v}^{-1/2} \mathbf{A_{graph}} \mathbf{D_v}^{-1/2} \tag{17}$$

where $\mathbf{A_{graph}} = \mathbf{H}^T \mathbf{W_e} \mathbf{H}$ is the weighted graph associated with the graph corresponding to adjacency matrix $\mathbf{A}$,

$$\mathbf{L_{hyp}} = \mathbf{I} - \mathbf{D_v}^{-1/2} \mathbf{A_{hyp}} \mathbf{D_v}^{-1/2} \tag{18}$$

$$\mathbf{L_{inv}} = \mathbf{I} - \mathbf{D_e}^{-1/2} \mathbf{A_{inv}} \mathbf{D_e}^{-1/2} \tag{19}$$

$$\mathbf{L_{dual}} = \mathbf{I} - \mathbf{D_e}^{-1/2} \mathbf{A_{dual}} \mathbf{D_e}^{-1/2} \tag{20}$$

We get the d eigenvectors associated with the smallest d eigenvalues of the above graph Laplacians as the embeddings. We get vertex embeddings using Eq. 18 and hyperedge embedding using Eq. 20, and we refer to them together as **e2v-hyp**. Similarly, we get another pair of vertex embeddings (using Eq. 17) and hyperedge embedding (using Eq. 20). We refer to the later pair as **e2v**.

3.6 Discussion

Information Loss Many methods that we have developed so far make use of only pair-wise information. For example, the methods developed using Node2Vec in Section 3.4 are based on the skip-gram model, which learns embedding of nodes while maximizing the conditional probability of a node given another node in a context (Eq. 13). Similarly, the spectral methods (Section 3.5) are inherently two dimensional as they are based on matrix. Same is the case with skip-gram based Sen2Vec (**h2v-dbow**). However, Sen2Vec based on the **DM** architecture (**h2v-dm**) maximizes the conditional probability of a word given the previous set of words (*context*) without breaking this context (by concatenating or averaging the embedding of the previous words). However, the tensors principally capture the joint distribution over k cardinality hyperedges unlike conditional distribution like Sen2vec (**h2v-dm**) which are more appropriate for sequences.

Limitations There are a few limitations we can see with tensor approach as compared to **h2v-dm** (which also takes into account higher-order probability distribution). The embeddings for hyperedge and nodes are learned by decomposing the dual and the hypergraph tensors, again in a separate

manner. In contrary, in Sen2Vec, the sentence embeddings are learned for variable size sentences and also jointly with the word embeddings. However, learning joint embeddings by possibly doing joint tensor decomposition, is a task we leave as a future work. Another point to mention is that when we extract a sub-hypergraph of a given k size hyperedges, it is possible that a significant part of nodes are left out (as they might not be a part of any k size hyperedge). This can possibly cause a cold start problem in the corresponding tensor. However, this does not mean that we will not get embedding for those nodes. It simply means that we might not achieve embeddings for these nodes from the decomposition of k-way tensors. But we can get embedding for these nodes from other tensors for a different k. However, we can mitigate the cold start problem in a principled manner using a graph regularization term ($\mathbf{U}^{(\mathrm{j})^T}\mathbf{L}\mathbf{U}^{(\mathrm{j})}$) in our tensor decomposition objective function 5. Here, $\mathbf{L}$ is the graph laplacian of our choice. For example we can use $\mathbf{L_{hyp}}$ or $\mathbf{L_{graph}}$ when we perform tensor decomposition for node embeddings and use $\mathbf{L_{inv}}$ or $\mathbf{L_{dual}}$ for hyperedge embeddings. Lastly, the hypergraph autoencoder can only produce hyperedge embeddings and not the node embeddings. In this sense it is feeble to other methods.

4 EXPERIMENTS

4.1 DATASET DESCRIPTION

As the first dataset, we use group interaction log-data of the Sony's Online multi-player game EverQuest II (**EQ II**) (www.everquest2.com) for a time period of nine months. In this game, several players work together as a team to perform various tasks. Each team earn points after completion of each task, and as the teams progress by earning points, they are escalated to different levels of game play. The interestingness of the game increases with each level. The points earned by the teams are treated as a measure of group performance. Each set of players who played together is treated as a hyperedge. When the same set of players play together again, we treat it as hyperedge re-occurrence ($R(g_i)$). Players can participate in several teams over time, therefore, resulting in a hypergraph with overlapping hyperedges. We consider hyperedges of cardinality $\in [2, 6]$. The resulting dataset contains a total of 5964 hyperedges (teams) among 6536 nodes (players).

Second dataset, is the fully labeled and publicly available sentiment analysis corpus of *Stanford Sentiment Treebank* (**LangNet**) (Socher et al., 2013). This dataset is based on the reviews from a movie review website (www.rottentomatoes.com) and contains 215,154 unique phrases. Each of the phrases are labeled with a sentiment score (a real number $\in [0, 1]$, larger value indicates positive sentiment) by human annotators. Each phrase is a set or hyperedge of words. Given that words are shared across various phrases, these common words connect the phrase hyperedges, resulting in a phrase hypergraph with overlapping phrase hyperedges. Again, we only consider hyperedges of cardinality $\in [2, 6]$. After applying this cardinality filter we are left with 141,410 hyperedges (phrases) and 21,122 nodes (words).

4.2 EVALUATION METHODOLOGY AND EXPERIMENTAL SETUP

4.2.1 METHODS COMPARED

We seven different methods: **h2v-DM**, **h2v-DBOW**, **h2v-inv**, **h2v-dual**, **e2v**, **e2v-hyp**, **t2v** and **h2v-auto**. Except for **h2v-auto**, each of these methods result in node as well as hyperedge embeddings of dimension $d = 128$. We further combine the node and hyperedge embedding using five different strategies: i) node embedding summation (dimension $d = 128$), ii) node embedding summation and concatenation with hyperedge embedding (dimension $2 \times d = 256$), iii) node embedding averaging (dimension $d = 128$), iv) node embedding averaging and concatenation with hyperedge embedding (dimension $2 \times d = 256$), and v) only hyperedge embedding (dimension $d = 128$). **h2v-auto** only produces hyperedge embeddings of dimension $d = 128$. But it builds the embeddings using three different scenarios as mentioned in next section. Therefore, in total we have 38(35+3) different scenarios each resulting in a different hyperedge embedding.

4.2.2 EVALUATION TASKS AND SETUP

We perform two regression based tasks for the two datasets. In **EQII** dataset each team (hyperedge) has a team performance score associated with it. This team performance score is a real number,

Table 1: RMSE Scores for **EQ II** team performance

	Embedding Methods						
	Sen2Vec based		Node2Vec based		Spectral methods		Tensor method
Embed Combination	**h2v-DM**	**h2v-DBOW**	**h2v-inv**	**h2v-dual**	**e2v**	**e2v-hyp**	**t2v**
Node Embed Sum	0.79308	0.79567	0.80418	0.79956	0.81183	0.81405	0.81341
Node Embed Sum + Hyperedge Embed	0.79651	0.80241	0.81362	0.80636	0.8113	0.81652	0.81299
Node Embed Average	0.81584	0.81733	0.82407	0.82281	0.81234	0.81369	0.81303
Node Embed Avg + Hyperedge Embed	0.8182	0.82077	0.83378	0.82896	0.81223	0.81608	0.8127
Only Hyperedge Embed	0.81203	0.81522	0.82189	0.81984	0.81233	0.81608	0.81341

equal to the number of points earned by the team while performing one or more tasks within a gaming session. We treat the embedding learned for a given team (hyperedge) as its feature vector which is associated with a real number (team performance). We therefore, perform on regression over all the hyperedges (teams) with team performance as the dependent variable.

Similarly, in **LangNet** dataset each phrase (hyperedge) has a sentiment score associated with it, which again is a real number. Similar to the team dataset above, we treat the embedding learned for a given phrase (hyperedge) as its feature vector which is associated with a real number (sentiment score). We therefore, treat this as a regression task with sentiment score as the dependent variable and perform regression using the feature matrix containing embeddings of all the phrases.

For both the tasks we just described, we perform several evaluation runs. In each run we randomly choose 30% of hyperedges (teams or phrases) as the test set and learn ridge regression parameters using the remaining 70% training hyperedges for each of the 38 different embedding scenarios. Root mean squared error (RMSE) was chosen as the evaluation metric (the lower, the better). RMSE was calculated for each of the 35 scenarios and for each run. Final RMSE score was taken as the average RMSE score across five runs. Ridge regression's hyper-parameter was chosen by 5-fold cross-validation.

For the autoencoder method (**h2v-auto**) we consider three scenarios: (1) single hidden layer ($L1$) of $d = 128$; (2) two hidden layers ($L1$ & $L2$) with size of $L1 : d = 96$ and of $L2 : d = 32$. We concatenate these embedding to get a single $d = 128$ size embedding; and (3) two hidden layers ($L1$ & $L2$) with size of $L1 : d = 512$ and of $L2 : d = 128$. We use the output of $L2$, which is of dimension $d = 128$, as the embedding. For sampling, we use the distance parameter $h = 2$ for generating the sub-lattice for both datasets. Also, $p = 10$ & $p = 5$ number of hyperedges are sampled (corresponding to each hyperedge) from **EQII** and **LangNet**, respectively.

4.3 Results and Discussion

Tables 1 &2 show the RMSE scores for the tasks of team performance prediction and sentiment score prediction, respectively. These tables contain scores for all the 38 different scenarios: columns represent 7 different models while rows represent combination strategies. The scores for *Autoencoder* are shown in the separate Table 3 as autoencoder only generates hyperedge embeddings.

As we can observe that for our dataset and for both the regression tasks, almost all the embeddings are performing very similarly in terms of accuracy. However, we can observer in Table 4 that simple spectral methods based on the hypergraph and its dual (**e2v-hyp**) have significantly less running times for both the datasets. This makes these simple methods based on well understood eigen value decomposition techniques more attractive than the sophisticated and complex Sen2Vec or Node2vec based probabilistic models.

Over all, we observe that **e2v**, **e2v-hgraph** and **t2v**, all of which are matrix or tensor based algebraic techniques, perform comparable to state of the art embedding techniques from NLP. Therefore, it provides motivation to further explore algebraic techniques, which are easily interpretable in graph theoretic terms, as a reasonable alternative for learning representations.

Another thing we notice, is that in case of **LangNet** dataset, that Node2Vec based **h2v-inv** and **h2v-dual** methods are simply unable to run, as the $\mathbf{A_{inv}}$ and $\mathbf{A_{dual}}$ are quiet large (see N/A in Table 2). It seems that in **LangNet** and possibly in text data in general the hypergraph dual, which contain phrase to phrase edges turns out be containing significantly more edges than the number of node to

Table 2: RMSE Scores for **LangNet** Sentiment Analysis

	Embedding Methods						
	Sen2Vec based		Node2Vec based		Spectral methods		Tensor method
Embed Combination	**h2v-DM**	**h2v-DBOW**	**h2v-inv**	**h2v-dual**	**e2v**	**e2v-hyp**	**t2v**
Node Embed Sum	0.14081	0.14029	N/A	N/A	0.14633	0.14854	0.14194
Node Embed Sum + Hyperedge Embed	0.14028	0.13883	N/A	N/A	0.14627	0.14845	0.14144
Node Embed Average	0.14245	0.14115	N/A	N/A	0.14665	0.14852	0.14381
Node Embed Avg + Hyperedge Embed	0.14178	0.14007	N/A	N/A	0.14661	0.14845	0.14333
Only Hyperedge Embed	0.14194	0.14147	N/A	N/A	0.14744	0.14844	0.1482

Table 3: *Hypergraph AutoEncoder* RMSE Scores & Run-times for **LangNet** (Sentiment Analysis) & **EQ II** (Team Performance)

	EQ II			LangNet		
Layer Sizes	**L1:128**	**L1:96/L2:32**	**L1:512/L2:128**	**L1:128**	**L1:96/L2:32**	**L1:512/L2:128**
RMSE	0.81104	0.81512	0.81635	0.14568	0.14529	0.14784
Run Time	52 min	40 min	1 hr 20 min	2 hr 10 min	3 hr 20 min	6 hr

Table 4: Average Runtime (seconds) of various methods across datasets

	Embedding Methods						
	Sen2Vec based		Node2Vec based		Spectral methods		Tensor method
Dataset	**h2v-DM**	**h2v-DBOW**	**h2v-inv**	**h2v-dual**	**e2v**	**e2v-hyp**	**t2v**
EQ2	455.84	103.47	90.05	93.41	128.03	12.01	213.37
LangNet	80.61	62.31	211.97*	207.86*	221.46	47.12	483.81

***** these are average time taken for learning vertex embeddings only

node edges in $\mathbf{A_{hyp}}$. Performing context generation over $\mathbf{A_{inv}}$ and $\mathbf{A_{dual}}$ graphs turns out be very costly and we were unable to get hyperedge embeddings for **h2v-inv** and **h2v-dual** methods. Note, we however do get vertex embedding (as mentioned with a * mark below Table 4). But as you can see that even the time time taken for vertex embedding is similar or more than the total time taken by Spectral methods for learning both hyperedge as well as vertex embedding, together. This indicates the robustness of the Spectral methods, specially the **e2v-hyp** for different kinds of hypergraph data and edge densities (sparsity).

Furthermore, we also observe that even simple element-wise summation or averaging of node embeddings for the nodes (in a given hyperdge) also perform comparably when compared to hyperedge embedding alone. From this we can infer that depending upon the dataset, if we have less hyperedges and more nodes, than we would rather prefer to simply learn the hyperedge embedding directly rather than learning node embeddings and then performing aggregating operation over them. Aggregation might turn out be costly specially if average hyperedge size is large and the choice of aggregation function is an issue. Therefore, learning hyperedge embeddings directly seems to be escape the problem of choosing the aggregation function all together.

5 Related Works

In Representation learning (Bengio et al., 2013) features or (geometric) representations of the data (topology) are used to build models. Machine learning algorithms make use of these features for prediction. Traditionally these features are readily available within the data-set or are engineered manually. However, this is a tedious labor-intensive process. Alternatively, the representations can be learned by the algorithm automatically in a task-dependent supervised, or a task-independent unsupervised manner. In this work we focus on unsupervised algorithms for learning network embeddings. Unsupervised feature learning using latent models has been done across the field of machine learning, e.g. LSA, LDA [3] for language models and Matrix Factorization (Ahmed et al., 2013)/ Community detection (Tang & Liu, 2011) based techniques for networks (Roweis & Saul, 2000; Belkin & Niyogi, 2001; Tenenbaum et al., 2000; Cox & Cox, 2000). In each case there is a vector of features learned for a word or node, each of whose entries reflects association with some latent dimension or community in the case of networks. This type of latent vector is learned for each node from the graphs and reflect, in a sense, the node's association with the various communities existing within the network. For words the input is a word co-occurence matrix of words residing in the same document. Word feature vector learned reflects the latent topics or classes of words. However, these embeddings reflect representation for node or words *globally* rather than *locally*. However, they are based on the spectral properties of the graph/ affinity matrix and mostly involve some form of eigen-decomposition, which can be costly.

A number of embedding methods have been proposed in the neural network literature. Unsupervised models learn embeddings by predicting neighbors in a context. The context can be defined by nodes in a graph (Grover & Leskovec, 2016) or by words in a sentence (Mikolov et al., 2013b; Le & Mikolov, 2014). Supervised algorithms learn embeddings which are optimal for the specific task at hand. This results in high accuracy but incurs significant computational cost for training. Recently, several supervised learning algorithms have been proposed for network analysis (Tian et al., 2014; Xiaoyi et al., 2013) and for text networks in a semi-supervised setting (Tang et al., 2015a).

Representation learning for sets using neural networks has been proposed recently (Vinyals et al., 2016), where a memory network is used to compose features sequentially but in an order invariant manner.

6 Conclusion

In this paper we have proposed several methods to generate higher-order representations for both hyperedges (representing sets of nodes) and hypergraph nodes (that also take into account the hypergraph structure). We build feature learning models that leverage the existing structure present in network data as context. We highlight the connection between higher order tensor methods and higher order probabilistic models in NLP. We argue that tensor/matrix based methods can be used to construct embeddings which have a natural graph/hypergraph theoretic interpretation. Therefore, the embedding can be better interpreted, unlike neural network based black-box approaches. While

introducing a new idea of a dual tensors corresponding to the hypergraph dual, we also highlight the novel approach of using factors from decomposition of N-way tensors corresponding to hypergraph, as generic node & hyperedge representations.

REFERENCES

Sameer Agarwal, Kristin Branson, and Serge Belongie. Higher order learning with graphs. In *Proceedings of the 23rd international conference on Machine learning*, pp. 17–24. ACM, 2006.

Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pp. 37–48. ACM, 2013.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, volume 14, pp. 585–591, 2001.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

Claude Berge. *Hypergraphs: combinatorics of finite sets*, volume 45. Elsevier, 1984.

Joshua Cooper and Aaron Dutle. Spectra of uniform hypergraphs. *Linear Algebra and its Applications*, 436(9):3268–3292, 2012.

Trevor F Cox and Michael AA Cox. *Multidimensional scaling*. CRC press, 2000.

Ernesto Estrada and Juan A Rodriguez-Velazquez. Complex networks as hypergraphs. *arXiv preprint physics/0505137*, 2005.

Shenghua Gao, Ivor Wai-Hung Tsang, and Liang-Tien Chia. Laplacian sparse coding, hypergraph laplacian sparse coding, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):92–104, 2013.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 855–864, 2016. doi: 10.1145/2939672.2939754.

Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3): 455–500, 2009.

Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, volume 14, pp. 1188–1196, 2014.

Lei Li and Tao Li. News recommendation via hypergraph learning: encapsulation of user behavior and news content. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pp. 305–314. ACM, 2013.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, NIPS'13, pp. 3111–3119, 2013b.

James R Munkres. *Elements of algebraic topology*, volume 2. Addison-Wesley Menlo Park, 1984.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710. ACM, 2014.

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

Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

Ankit Sharma, Rui Kuang, Jaideep Srivastava, Xiaodong Feng, and Kartik Singhal. Predicting small group accretion in social networks: A topology based incremental approach. In *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 408–415. IEEE, 2015.

Ankit Sharma, Terrence J Moore, Ananthram Swami, and Jaideep Srivastava. 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.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng, and Christopher Potts. Parsing With Compositional Vector Grammars. In *EMNLP*. 2013.

Steven H Strogatz. Exploring complex networks. *Nature*, 410(6825):268–276, 2001.

Jian Tang, Meng Qu, and Qiaozhu Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1165–1174. ACM, 2015a.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pp. 1067–1077. ACM, 2015b.

Lei Tang and Huan Liu. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478, 2011.

Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. Learning deep representations for graph clustering. In *AAAI*, pp. 1293–1299, 2014.

Ze Tian, TaeHyun Hwang, and Rui Kuang. A hypergraph-based learning algorithm for classifying gene expression and arraycgh data with prior knowledge. *Bioinformatics*, 25(21):2831–2838, 2009.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. In *ICLR*, 2016.

Li Xiaoyi, Li Hui Du Nan, et al. A deep learning approach to link prediction in dynamic networks. In *Proceedings of the 2013 SIAM International Conference on Data Mining. Philadelphia, PA, USA: SIAM*, 2013.

Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *Advances in neural information processing systems*, pp. 1601–1608, 2006.