Knowledge Hypergraphs: Prediction Beyond Binary Relations*

Bahare Fatemi[1,2†], **Perouz Taslakian**[2], **David Vazquez**[2] and **David Poole**[1]
[1]University of British Columbia
[2]Element AI
{bfatemi, poole}@cs.ubc.ca, {perouz,dvazquez}@elementai.com,

Abstract

Knowledge graphs store facts using relations between two entities. In this work, we address the question of link prediction in knowledge hypergraphs where relations are defined on *any number* of entities. While techniques exist (such as reification) that convert non-binary relations into binary ones, we show that current embedding-based methods for knowledge graph completion do not work well out of the box for knowledge graphs obtained through these techniques. To overcome this, we introduce *HSimplE* and *HypE*, two embedding-based methods that work directly with knowledge hypergraphs. In both models, the prediction is a function of the relation embedding, the entity embeddings and their corresponding positions in the relation. We also develop public datasets, benchmarks and baselines for hypergraph prediction and show experimentally that the proposed models are more effective than the baselines.

1 Introduction

Knowledge hypergraphs are graph structured knowledge bases that store facts about the world in the form of relations among any number of entities. They can be seen as one generalization of *knowledge graphs* in which relations are defined on at most two entities. Since accessing and storing all the facts in the world is difficult, knowledge bases are incomplete; the goal of *link prediction* in knowledge (hyper)graphs (or *knowledge (hyper)graph completion*) is to predict unknown links or relationships between entities based on existing ones. In this work, we are interested in the problem of link prediction in knowledge hypergraphs. Our motivation for studying link prediction in these more sophisticated knowledge structures is based on the fact that most knowledge in the world has inherently complex compositions.

Link prediction in knowledge graphs is a problem that is studied extensively, and has applications in several tasks such as automatic question answering [Ferrucci *et al.*, 2010]. In these studies, knowledge graphs are defined as directed graphs having nodes as entities and labeled edges as relations; edges are directed from the *head* entity to the *tail* entity. The common data structure for representing knowledge graphs is a set of triples $relation(head, tail)$ that represents information as a collection of binary relations. There exist a large number of knowledge graphs that are publicly available, such as FREEBASE [Bollacker *et al.*, 2008]. Wen *et al.* (2016) observe that in the original FREEBASE more than $1/3$rd of the entities participate in non-binary relations (i.e., defined on more than two entities). We observe, in addition, that 61% of the relations in the original Freebase are non-binary.

Embedding-based models [Nguyen, 2017; Kazemi *et al.*, 2020] have proved to be effective for knowledge graph completion. These approaches learn embeddings for entities and relations. To find out if a triple $relation(head, tail)$ is true, such models define a function that uses embeddings of $relation$, $head$, and $tail$ and output the probability of the triple. While successful, such embedding-based methods make the strong assumption that all relations are binary.

Knowledge hypergraph completion is a relatively underexplored area. We motivate our work by outlining that converting non-binary relations into binary ones using methods such as reification or star-to-clique [Wen *et al.*, 2016], and then applying known link prediction methods does not yield satisfactory results. *Reification* is one common approach of converting higher-arity relations into binary ones. In order to reify a tuple having a relation defined on k entities $e_1, \ldots, e_k$, we form k new binary relations, one for each position in this relation, and a new entity e for this tuple and connect e to each of the k entities that are part of the given tuple using the k binary relations. Another conversion approach is *Star-to-clique*, which converts a tuple defined on k entities into $\binom{k}{2}$ tuples with distinct relations between all pairwise entities in the tuple.

Both conversion approaches have their caveats when current link prediction models are applied to the resulting graphs. The example in Figure 1a shows three facts that pertain to the relation *flies_between*. When we reify the hypergraph in this example (Figure 1b), we add three reified entities. In terms of representation, the binary relations created are equivalent to the original representation and reification does not lose information during conversion. A problem with reification, however, arises at test time: because we introduce new enti-

*This paper with the supplementary material can be found at https://arxiv.org/abs/1906.00137.

†Contact Author

ties that the model never encounters during training, we do not have a learned embedding for these entities; and current embedding-based methods require an embedding for each entity in order to be able to make a prediction.

Applying the star-to-clique method to a hypergraph does not yield better results, as star-to-clique conversion loses information. Figure 1c shows the result of applying star-to-clique to the original hypergraph (Figure 1a), in which the tuple *flies_between(Air Canada, New York, Los Angeles)* might be interpreted as being true (since the corresponding entities are connected by edges), whereas looking at the original hypergraph, it is clear that Air Canada does not fly from New York to Los Angeles.

In this work, we introduce two embedding-based models that perform link prediction directly on knowledge hypergraphs without converting them to graphs. Both proposed models are based on the idea that predicting the existence of a relation between a set of entities depends on the position of the entities in the relation; otherwise, the relation is symmetric. On the other hand, learning entity embeddings for each position independently does not work well either, as this does not let the information flow between the embeddings for the different positions of the same entity. The first model we propose is *HSimplE*. For a given entity, *HSimplE* shifts the entity embedding by a value that depends on the position of the entity in the given relation. Our second model is *HypE*, which in addition to learning entity embeddings, learns positional (convolutional) embeddings; these positional embeddings are disentangled from entity representations and are used to transform the representation of an entity based on its position in a relation. This makes HypE more robust to changes in the position of an entity within a tuple. We show that both HSimplE and HypE are fully expressive. To evaluate our models, we introduce two new datasets from subsets of FREEBASE, and develop baselines by extending existing models on knowledge graphs to work with hypergraphs.

The contributions of this paper are: (1) showing that current techniques to convert a knowledge hypergraph to knowledge graph do not yield satisfactory results for the link prediction task, (2) introducing HypE and HSimplE, two models for knowledge hypergraph completion, (3) a set of baselines for knowledge hypergraph completion, and (4) two new datasets containing multi-arity relations obtained from subsets of FREEBASE, which can serve as new evaluation benchmarks for knowledge hypergraph completion methods. We also show that our proposed methods outperform baselines.

2 Related Work

Existing methods that relate to our work in this paper can be grouped into the following three main categories.

Knowledge graph completion. Embedding-based models for knowledge graph completion such as *translational* [Bordes *et al.*, 2013; Wang *et al.*, 2014], *bilinear* [Yang *et al.*, 2015; Kazemi and Poole, 2018], and *deep models* [Socher *et al.*, 2013] have proved to be effective for knowledge graphs where all relations are binary. In Section 6 we extend some of the models in this category to knowledge hypergraphs, and compare their performance with the proposed methods.

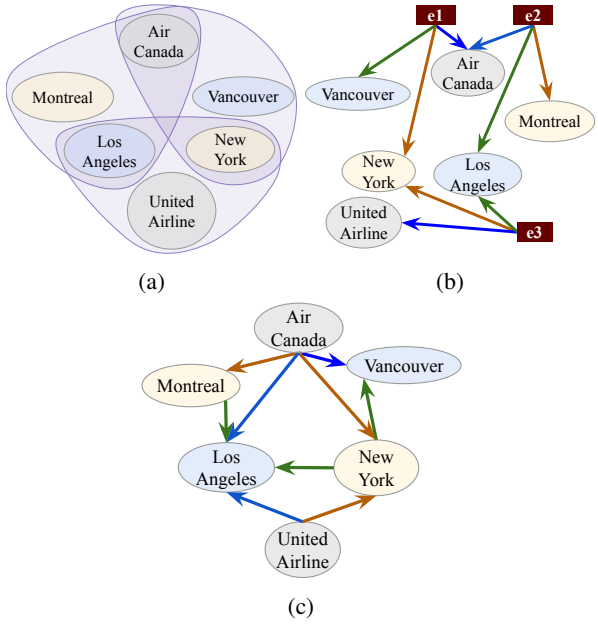


Figure 1: (a) Relation *flies_between* with arity 3 defined on three tuples. (b) Reifying non-binary relations by creating three additional entities e_1, e_2, and e_3. (c) Converting non-binary relations into cliques using star-to-clique.

Knowledge hypergraph completion. Soft-rule models [De Raedt *et al.*, 2007; Kazemi *et al.*, 2014] can easily handle variable arity relations and have the advantage of being interpretable. However, they can only learn a subset of patterns [Nickel *et al.*, 2016]. Guan *et al.* (2019) propose an embedding-based method based on the star-to-clique approach. The caveats of this approach are discussed earlier. m-TransH [Wen *et al.*, 2016] extends TransH [Wang *et al.*, 2014] to knowledge hypergraph completion. Kazemi and Poole (2018) prove that TransH, and consequently m-TransH, are not fully expressive and have restrictions in modeling relations. In contrast, we prove that our proposed models are fully expressive.

Learning on hypergraphs. Hypergraph learning has been employed to model high-order correlations among data in many tasks, such as in video object segmentation [Huang *et al.*, 2009] and in modeling image relationships and image ranking [Huang *et al.*, 2010]. There is also a line of work extending graph neural networks to hypergraph neural networks [Feng *et al.*, 2019] and hypergraph convolution networks [Yadati *et al.*, 2018]. These models are designed for undirected hypergraphs, with edges that are not labeled (no relations), while knowledge hypergraphs are directed and labeled graphs. As there is no clear or easy way of extending these models to our knowledge hypergraph setting, we do not consider them as baselines for our experiments.

3 Definition and Notation

A world consists of a finite set of entities $\mathcal{E}$, a finite set of relations $\mathcal{R}$, and a set of tuples τ where each tuple in τ is of the form $r(e_1, e_2, \ldots, e_k)$ where $r \in \mathcal{R}$ is a relation and

each $e_i \in \mathcal{E}$ is an entity, for all $i = 1, 2, \ldots, k$. The *arity* $|r|$ of a relation r is the number of arguments that the relation takes and is fixed for each relation. A world specifies what is true: all the tuples in τ are true, and the tuples that are not in τ are false. A knowledge hypergraph consists of a subset of the tuples $\tau' \subseteq \tau$. Link prediction in knowledge hypergraphs is the problem of predicting the missing tuples in τ', that is, finding the tuples $\tau \setminus \tau'$.

An *embedding* is a function that converts an entity or a relation into a vector (or sometimes a higher order tensor) over a field (typically the real numbers). We use bold lower-case for vectors, that is, $\mathbf{e} \in \mathbb{R}^k$ is an embedding of entity e, and $\mathbf{r} \in \mathbb{R}^l$ is an embedding of a relation r.

Let $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}$ be a set of vectors. The variadic function $\text{concat}(\mathbf{v_1}, \ldots, \mathbf{v_k})$ outputs the concatenation of its input vectors. We define the variadic function $\odot()$ to be the sum of the element-wise product of its input vectors, namely $\odot(\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}) = \sum_{i=1}^{\ell} \mathbf{v_1}^{(i)} \mathbf{v_2}^{(i)} \ldots \mathbf{v_k}^{(i)}$ where each vector $\mathbf{v_i}$ has the same length, and $\mathbf{v_j}^{(i)}$ is the i-th element of vector $\mathbf{v_j}$. The 1D convolution operator $*$ takes as input a vector $\mathbf{v}$, a convolution weight filter ω, and a stride s and outputs the standard 1D convolution as defined in torch.nn.Conv1D function in PyTorch [Paszke *et al.*, 2019].

For the task of knowledge graph completion, an embedding-based model defines a function ϕ that takes a tuple x as input, and generates a prediction, *e.g.*, a probability (or score) of the tuple being true. A model is *fully expressive* if given any complete world (full assignment of truth values to all tuples), there exists an assignment of values to the embeddings of the entities and relations that accurately separates the tuples that are true in the world from those that are false.

4 Knowledge Hypergraph Completion: Proposed Methods

The idea at the core of our methods is that the way an entity representation is used to make predictions is affected by the role (or position) that the entity plays in a given relation. In the example in Figure 1a, Montreal is the departure city; but it may appear in a different position (e.g., arrival city) in another tuple. This means that the way we use Montreal's embedding for computing predictions may need to vary based on the position it appears in within the tuple. In general, when the embedding of an entity does not depend on its position in the tuple during prediction, then the relation has to be symmetric (which is not the case for most relations). On the other hand, when entity embeddings are based on position but are learned independently, information about one position will not interact with that of others. It should be noted that in several embedding-based methods for knowledge graph completion, such as canonical polyadic [Hitchcock, 1927; Lacroix *et al.*, 2018], ComplEx [Trouillon *et al.*, 2016], and SimplE [Kazemi and Poole, 2018], the prediction depends on the position of each entity in the tuple.

In what follows, we propose two embedding-based methods for link prediction in knowledge hypergraphs. The first model is inspired by SimplE and has its roots in knowledge graph completion; the second model takes a fresh look at knowledge completion as a multi-arity problem, without first setting it up within the frame of binary relation prediction.

HSimplE. HSimplE is an embedding-based method for link prediction in knowledge hypergraphs that is inspired by SimplE [Kazemi and Poole, 2018]. SimplE learns two embedding vectors $\mathbf{e^{(1)}}$ and $\mathbf{e^{(2)}}$ for an entity e (one for each possible position of the entity), and two embedding vectors $\mathbf{r^{(1)}}$ and $\mathbf{r^{(2)}}$ for a relation r (with one relation embedding as the inverse of the other). It then computes the score of a triple as $\phi(r(e_1, e_2)) = \odot(\mathbf{r^{(1)}}, \mathbf{e_1^{(1)}}, \mathbf{e_2^{(2)}}) + \odot(\mathbf{r^{(2)}}, \mathbf{e_2^{(1)}}, \mathbf{e_1^{(2)}})$.

In HSimplE, we adopt the idea of having different representations for an entity based on its position in a relation and updating all these representations from a single training tuple. We do this by representing each entity e as a single vector $\mathbf{e}$ (instead of multiple vectors as in SimplE), and each relation r as a single vector $\mathbf{r}$. Conceptually, each $\mathbf{e}$ can be seen as the concatenation of the different representations of e based on every possible position. For example, in a knowledge hypergraph where the relation with maximum arity is α, an entity can appear in α different positions; hence $\mathbf{e}$ will be the concatenation of α vectors, one for each possible position. HSimplE scores a tuple using the following function.

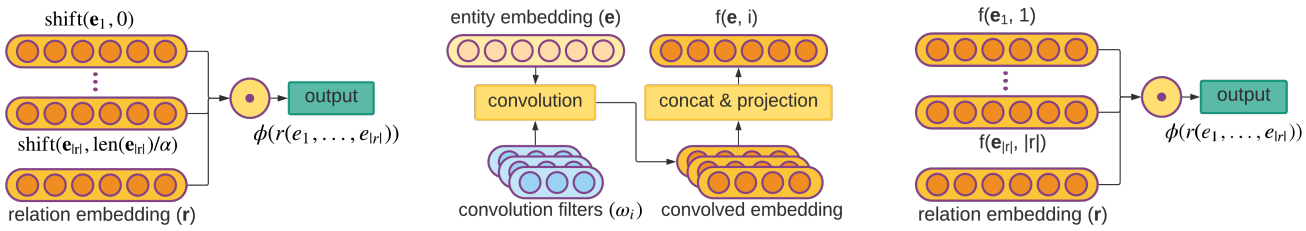
$$\phi(r(e_i, e_j, \ldots, e_k)) = \odot(\mathbf{r}, \mathbf{e_i}, \text{shift}(\mathbf{e_j}, \text{len}(\mathbf{e_j})/\alpha), \ldots, \text{shift}(\mathbf{e_k}, \text{len}(\mathbf{e_k}) \cdot (\alpha - 1)/\alpha))) \quad (1)$$

Here, $\text{shift}(\mathbf{v}, x)$ shifts vector $\mathbf{v}$ to the left by x steps, $\text{len}(\mathbf{e})$ returns length of vector $\mathbf{e}$, and $\alpha = \max_{r \in \mathcal{R}}(|r|)$. We observe that for knowledge graphs ($\alpha = 2$), SimplE is a special instance of HSimplE, with $\mathbf{e} = \text{concat}(\mathbf{e}^{(1)}, \mathbf{e}^{(2)})$ and $\mathbf{r} = \text{concat}(\mathbf{r}^{(1)}, \mathbf{r}^{(2)})$. The architecture of HSimplE is summarized in Figure 2a.

HypE. HypE learns a single representation for each entity, a single representation for each relation, and positional convolutional weight filters for each possible position. When an entity appears in a specific position, the appropriate positional filters are first used to transform the embedding of each entity in the given fact; these transformed entity embeddings are then combined with the embedding of the relation to produce a *score*, *i.e.*, the probability of the input tuple to be true. The architecture of HypE is summarized in Figures 2b and 2c.

Let n denote the number of filters per position, l the filter-length, d the embedding dimension, and s the stride of a convolution. Let $\omega_i \in \mathbb{R}^{n \times l}$ be the convolutional filters associated with position i, and let $\omega_{ij} \in \mathbb{R}^l$ be the jth row of ω_i. We denote by $P \in \mathbb{R}^{nq \times d}$ the projection matrix, where $q = \lfloor (d - l)/s \rfloor + 1$ is the feature map size. For a given tuple, define $f(\mathbf{e}, i) = \text{concat}(\mathbf{e} * \omega_{\text{i1}}, \ldots, \mathbf{e} * \omega_{\text{in}})\text{P}$ to be a function that returns a vector of size d based on the entity embedding $\mathbf{e}$ and its position i in the tuple. Thus, each entity embedding $\mathbf{e}$ appearing at position i in a given tuple is convolved with the set of position-specific filters ω_i to give n feature maps of size q. All n feature maps corresponding to an entity are concatenated into a vector of size nq and projected to the embedding space through multiplication by P. The projected vectors of entities and the embedding of the relation are combined by inner-product to define ϕ:

$$\phi(r(e_1, \ldots, e_{|r|})) = \odot(\mathbf{r}, f(\mathbf{e_1}, 1), \ldots, f(\mathbf{e_{|r|}}, |r|)). \quad (2)$$



(a) (b) (c)

Figure 2: Visualization of HSimplE and HypE architectures. (a) ϕ for HSimplE transforms entity embeddings by shifting them based on their position and combining them with the relation embedding. (b) $f(\mathbf{e}, i)$ for HypE takes an entity embedding and the position the entity appears in the given tuple, and returns a vector. (c) ϕ takes as input a tuple and outputs the score of HypE for the tuple.

The advantage of learning positional filters disentangled from entity embeddings is two-folds: On one hand, learning a single vector per entity keeps entity representations simple and disentangled from its position in a given fact. On the other hand, unlike HSimplE, HypE learns positional filters from all entities that appear in the given position; overall, this separation of representations for entities, relations, and positions facilitates the representation of knowledge bases having facts of arbitrary number of entities. It also gives HypE an additional robustness, such as in the case when we test a trained HypE model on a tuple that contains an entity in a position never seen before at train time. We discuss this in Section 6.1.

Both HSimplE and HypE are fully expressive — an important property that has been the focus of several studies [Fatemi *et al.*, 2019]. A model that is not fully expressive can embed assumptions that may not be reflected in reality.

Theorem 1 (Expressivity)**.** *For any ground truth over entities $\mathcal{E}$ and relations $\mathcal{R}$ containing $|\tau|$ true tuples and $\alpha = \max_{r \in \mathcal{R}}(|r|)$, there exists a HypE and a HSimplE model with embedding vectors of size $\max(\alpha|\tau|, \alpha)$ that represents that ground truth.*

Proof Sketch. To prove the theorem, we show an assignment of embedding values for each of the entities and relations in τ such that the scoring function of HypE and HSimplE gives 1 for $t \in \tau$ and 0 otherwise. □

4.1 Objective Function and Training

Both HSimplE and HypE are trained using stochastic gradient descent with mini-batches. In each learning iteration, we take a batch of positive tuples from the knowledge hypergraph. As we only have positive instances available, we need to also train our model on negative instances; thus, for each positive instance, we produce a set of negative instances. For negative sample generation, we follow the contrastive approach of [Bordes *et al.*, 2013] for knowledge graphs and extend it to knowledge hypergraphs: for each tuple, we produce a set of negative samples of size $N|r|$ by replacing each of the entities with N random entities in the tuple, one at a time. Here, N is the ratio of negative samples in our training set and is a hyperparameter.

Given a knowledge hypergraph defined on τ', we let τ'_{train}, τ'_{test}, and τ'_{valid} denote the train, test, and validation sets, respectively, so that $\tau' = \tau'_{train} \cup \tau'_{test} \cup \tau'_{valid}$. For any tuple x in τ', we let $T_{neg}(x)$ be a function that generates a set of negative samples through the process described above. Let $\mathbf{r}$ and $\mathbf{e}$ represent relation and entity embeddings respectively. We define the following cross entropy loss:

$$\mathcal{L}(\mathbf{r}, \mathbf{e}) = \sum_{x' \in \tau'_{train}} -log\left(\frac{e^{\phi(x')}}{e^{\phi(x')} + \sum_{x \in T_{neg}(x')} e^{\phi(x)}}\right)$$

5 Experimental Setup

5.1 Datasets

The experiments on knowledge hypergraph completion are conducted on three datasets. The first is JF17K proposed by Wen *et al.* (2016); as no validation set is proposed for JF17K, we randomly select 20% of the train set as validation. We also create two datasets FB-AUTO and M-FB15K from FREEBASE. For the experiments on datasets with binary relations, we use two standard benchmarks for knowledge graph completion: WN18 [Bordes *et al.*, 2014] and FB15k [Bordes *et al.*, 2013]. See Table 2 for statistics of the datasets.

5.2 Baselines

To compare our results to that of existing work, we first design simple baselines that extend current models to work with knowledge hypergraphs. We only consider models that admit a simple extension to higher-binary relations for the link prediction task. The baselines for this task are grouped into the following categories: (1) methods that work with binary relations and that are easily extendable to higher-arity, namely r-SimplE, m-DistMult, and m-CP; (2) existing methods that can handle higher-arity relations, namely m-TransH. Below we give some details about methods in category (1).

r-SimplE. To test the performance of a model trained on reified data, we convert higher-arity relations in the train set to binary relations through reification. We then use SimplE (that we call r-SimplE) on this reified data. In this setting, at test time higher-arity relations are first reified to a set of binary relations; this process creates new auxiliary entities for which the model has no learned embeddings. To embed the

	JF17K				FB-AUTO				M-FB15K			
Model	MRR	Hit@1	Hit@3	Hit@10	MRR	Hit@1	Hit@3	Hit@10	MRR	Hit@1	Hit@3	Hit@10
r-SimplE	0.102	0.069	0.112	0.168	0.106	0.082	0.115	0.147	0.051	0.042	0.054	0.070
m-DistMult	0.463	0.372	0.510	0.634	0.784	0.745	0.815	0.845	0.705	0.633	0.740	0.844
m-CP	0.391	0.298	0.443	0.563	0.752	0.704	0.785	0.837	0.680	0.605	0.715	0.828
m-TransH [Wen *et al.*, 2016]	0.444	0.370	0.475	0.581	0.728	0.727	0.728	0.728	0.623	0.531	0.669	0.809
HSimplE (Ours)	0.472	0.378	0.520	0.645	0.798	0.766	0.821	0.855	0.730	0.664	0.763	0.859
HypE (Ours)	**0.494**	**0.408**	**0.538**	**0.656**	**0.804**	**0.774**	**0.823**	**0.856**	**0.777**	**0.725**	**0.800**	**0.881**

Table 1: Knowledge hypergraph completion results on JF17K, FB-AUTO and M-FB15K for baselines and the proposed method. The prefixes 'r' and 'm' in the model names stand for *reification* and *multi-arity* respectively. Both our methods outperform the baselines on all datasets.

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	#train	#valid	#test
WN18	40,943	18	141,442	5,000	5,000
FB15k	14,951	1,345	483,142	50,000	59,071
JF17K	29,177	327	77,733	–	24,915
FB-AUTO	3,410	8	6,778	2,255	2,180
M-FB15K	10,314	71	415,375	39,348	38,797

Table 2: Dataset Statistics.

auxiliary entities for the prediction step, we use the observation we have about them at test time. For example, a higher-arity relation $r(e_1, e_2, e_3)$ is reified at test time by adding a new entity e' and converting the higher-arity tuple to three binary facts: $r_1(e', e_1)$, $r_2(e', e_2)$, and $r_3(e', e_3)$. When predicting the tail entity of $r_1(e', ?)$, we use the other two reified facts to learn an embedding for entity e'. Because e' is added only to help represent the higher-arity relations as a set of binary relations, we only need to do tail prediction for reified relations. Note that we do not reify binary relations.

m-DistMult. DistMult [Yang *et al.*, 2015] defines a score function $\phi(r(e_i, e_j)) = \odot(\mathbf{r}, \mathbf{e_i}, \mathbf{e_j})$. To accommodate non-binary relations, we redefine this function as $\phi(r(e_i, \ldots, e_j)) = \odot(\mathbf{r}, \mathbf{e_i}, \ldots, \mathbf{e_j})$.

m-CP. Canonical Polyadic decomposition [Hitchcock, 1927] is a tensor decomposition approach. We refer to the version that only handles binary relations as *CP*. CP embeds each entity e as two vectors $\mathbf{e^{(1)}}$ and $\mathbf{e^{(2)}}$, and each relation r as a single vector $\mathbf{r}$; it defines the score function $\phi(r(e_i, e_j)) = \odot(\mathbf{r}, \mathbf{e_i^{(1)}}, \mathbf{e_j^{(2)}})$. We extend CP to *m-CP*, which accommodates relations of any arity. m-CP embeds each entity e as α different vectors $\mathbf{e^{(1)}}, .., \mathbf{e^{(\alpha)}}$, where $\alpha = \max_{r \in \mathcal{R}}(|r|)$; it computes the score of a tuple as $\phi(r(e_i, \ldots, e_j)) = \odot(\mathbf{r}, \mathbf{e_i^{(1)}}, ..., \mathbf{e_j^{(|r|)}})$.

5.3 Evaluation Metrics

Given a knowledge hypergraph on τ', we evaluate various completion methods using a train and test set τ'_{train} and τ'_{test}. We use two evaluation metrics: Hit@t and Mean Reciprocal Rank (MRR). Both these measures rely on the *ranking* of a tuple $x \in \tau'_{test}$ within a set of *corrupted* tuples. For each tuple $r(e_1, \ldots, e_k)$ in τ'_{test} and each entity position i in the tuple, we generate $|\mathcal{E}| - 1$ corrupted tuples by replacing the entity e_i with each of the entities in $\mathcal{E} \setminus \{e_i\}$. For example, by corrupting entity e_i, we would obtain a new tuple $r(e_1, \ldots, e_i^c, \ldots, e_k)$ where $e_i^c \in \mathcal{E} \setminus \{e_i\}$. Let the set of corrupted tuples, plus $r(e_1, \ldots, e_k)$, be denoted by $\theta_i(r(e_1, \ldots, e_k))$. Let $\text{rank}_i(r(e_1, \ldots, e_k))$ be the ranking of $r(e_1, \ldots, e_k)$ within $\theta_i(r(e_1, \ldots, e_k))$ based on the score $\phi(x)$ for each $x \in \theta_i(r(e_1, \ldots, e_k))$. We compute the MRR as $\frac{1}{K} \sum_{r(e_1,\ldots,e_k) \in \tau'_{test}} \sum_{i=1}^{k} \frac{1}{\text{rank}_i r(e_1,\ldots,e_k)}$ where $K = \sum_{r(e_1,\ldots e_k) \in \tau'_{test}} |r|$ is the number of prediction tasks. Hit@t measures the proportion of tuples in τ'_{test} that rank among the top t in their corresponding corrupted sets. We follow [Bordes *et al.*, 2013] and remove all corrupted tuples that are in τ' from our computation of MRR and Hit@t.

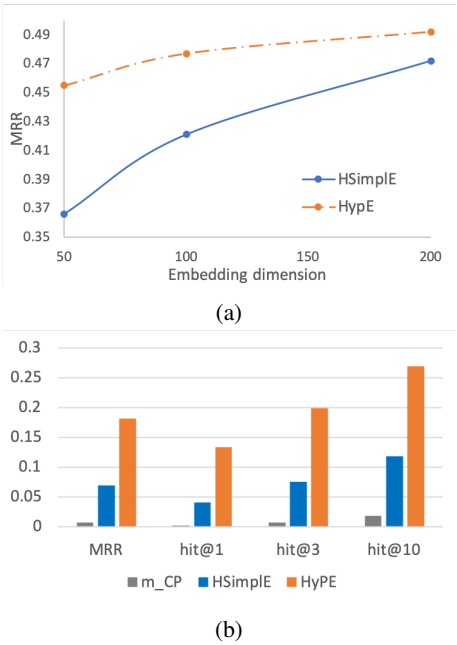


Figure 3: These experiments show that HypE outperforms HSimplE when trained with fewer parameters, and when tested on samples that contain at least one entity in a position never encountered during training. (a) MRR of HypE and HSimplE for different embedding dimensions. (b) Results of m-CP, HSimplE, and HypE on the *missing positions* test set (containing 1,806 test samples).

6 Experiments

This section summarizes our experiments [1].

[1] Code and data available at https://github.com/ElementAI/HypE.

	WN18				FB15k			
Model	MRR	Hit@1	Hit@3	Hit@10	MRR	Hit@1	Hit@3	Hit@10
CP [Hitchcock, 1927]	0.074	0.049	0.080	0.125	0.326	0.219	0.376	0.532
TransH [Wang *et al.*, 2014]	-	-	-	0.867	-	-	-	0.585
m-TransH [Wen *et al.*, 2016]	0.671	0.495	0.839	0.923	0.351	0.228	0.427	0.559
DistMult [Yang *et al.*, 2015]	0.822	0.728	0.914	0.936	0.654	0.546	0.733	0.824
HSimplE (Ours) and SimplE [Kazemi and Poole, 2018]	**0.942**	**0.939**	**0.944**	**0.947**	**0.727**	**0.660**	0.773	0.838
HypE (Ours)	0.934	0.927	0.940	0.944	0.725	0.648	**0.777**	**0.856**

Table 3: Knowledge graph completion results on WN18 and FB15K for baselines and HypE. Note that in knowledge graphs (binary relations), HSimplE and SimplE are equivalent, both theoretically and experimentally. The results show that our methods outperform the baselines.

	Arity			
Model	2	3	4-5-6	All
r-SimplE	0.478	0.025	0.017	0.168
m-DistMult	0.495	0.648	0.809	0.634
m-CP	0.409	0.563	0.765	0.560
m-TransH [Wen *et al.*, 2016]	0.411	0.617	0.826	0.596
HSimplE (Ours)	**0.497**	**0.699**	0.745	0.645
HypE (Ours)	0.466	0.693	**0.858**	**0.656**
# train tuples	36,293	18,846	6,772	61,911
# test tuples	10,758	10,736	3,421	24,915

Table 4: Breakdown performance of Hit@10 across relations with different arities on JF17K dataset along with their statistics.

6.1 Knowledge Hypergraph Completion Results

The results of our experiments, summarized in Table 1, show that both HSimplE and HypE outperform the proposed baselines across the three datasets JF17K, FB-AUTO, and M-FB15K. They further demonstrate that reification for the r-SimplE model does not work well; this is because the reification process introduces auxiliary entities for which the model does not learn appropriate embeddings because these auxiliary entities appear in very few facts. Comparing the results of r-SimplE against HSimplE, we can also see that extending a model to work with hypergraphs works better than reification when high-arity relations are present.

The ability of knowledge sharing through the learned position-dependent convolution filters suggests that HypE would need a lower number of parameters than HSimplE to obtain good results. To test this, we train both models with different embedding dimensions. Figure 3a shows the MRR on the test set for each model with different embedding sizes. Based on the MRR result, we can see that HypE outperforms HSimplE by 24% for embedding dimension 50, implying that HypE works better under a constrained budget.

Disentangling the representations of entity embeddings and positional filters enables HypE to better learn the role of position within a relation because the learning process considers the behavior of all entities that appear in a given position at the time of training. This becomes especially important in the case when some entities never appear in certain positions in the train set, but you still want to be able to reason about them no matter what position they appear in at test time. In order to test the effectiveness of our models in this more challenging scenario, we created a *missing positions* test set by selecting the tuples from our original test set that contain at least one entity in a position it never appears in within the training dataset. The results on these experiments (Figure 3b) show that (1) both HSimplE and HypE outperform m-CP (which learns different embeddings for each entity-position pair), and more importantly, (2) HypE significantly outperforms HSimplE for this challenging test set, leading us to believe that disentangling entity and position representations may be a better strategy for this scenario.

6.2 Knowledge Graph Completion Results

To show that HSimplE and HypE work well also on the more common knowledge graphs, we evaluate them on WN18 and FB15K. Table 3 shows link prediction results on WN18 and FB15K. Baseline results are taken from the original papers except that of m-TransH, which we implement ourselves. Instead of tuning the parameters of HypE to get potentially better results, we follow the Kazemi and Poole (2018) setup with the same grid search approach by setting $n = 2$, $l = 2$, and $s = 2$. This results in all models in Table 3 having the same number of parameters, and thus makes them directly comparable to each other. Note that for knowledge graph completion (all binary relations) HSimplE is equivalent to SimplE, both theoretically and experimentally (as shown in Section 4). The results show that on WN18 and FB15K, HSimplE and HypE outperform all baselines.

6.3 Ablation Study on Different Arities

We break down the performance of models across different arities. As the number of test tuples in higher arities (4-5-6) is much less than in smaller arities (2-3), we used equivalent size bins to show the decomposed results for a reasonable number of test tuples. Table 4 shows the Hit@10 results of the models for bins of arity 2, 3, and 4-5-6 in JF17K. The proposed models outperform the state-of-the-art and the baselines in all arities. We highlight that r-SimplE and HSimplE are quite different models for relations having arity > 2.

7 Conclusions

Knowledge hypergraph completion is an important problem that has received little attention. Having introduced two new knowledge hypergraph datasets, baselines, and two new methods for link prediction in knowledge hypergraphs, we hope to kindle interest in the problem. Unlike graphs, hypergraphs have a more complex structure that opens the door to more challenging questions such as: how do we effectively predict the missing entities in a given (partial) tuple?

References

[Bollacker *et al.*, 2008] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *ACM ICMD*, 2008.

[Bordes *et al.*, 2013] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.

[Bordes *et al.*, 2014] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 2014.

[Carlson *et al.*, 2010] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

[De Raedt *et al.*, 2007] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. *IJCAI*, 2007.

[Fatemi *et al.*, 2019] Bahare Fatemi, Siamak Ravanbakhsh, and David Poole. Improved knowledge graph embedding using background taxonomic information. In *AAAI*, 2019.

[Feng *et al.*, 2019] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *AAAI*, 2019.

[Ferrucci *et al.*, 2010] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 2010.

[Guan *et al.*, 2019] Saiping Guan, Xiaolong Jin, Yuanzhuo Wang, and Xueqi Cheng. Link prediction on n-ary relational data. In *WWW*, 2019.

[Hitchcock, 1927] Frank L Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

[Huang *et al.*, 2009] Yuchi Huang, Qingshan Liu, and Dimitris Metaxas. Video object segmentation by hypergraph cut. In *CVPR*, 2009.

[Huang *et al.*, 2010] Yuchi Huang, Qingshan Liu, Shaoting Zhang, and Dimitris N Metaxas. Image retrieval via probabilistic hypergraph ranking. In *CVPR*, 2010.

[Kazemi and Poole, 2018] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *NIPS*, 2018.

[Kazemi *et al.*, 2014] Seyed Mehran Kazemi, David Buchman, Kristian Kersting, Sriraam Natarajan, and David Poole. Relational logistic regression. In *KR*, 2014.

[Kazemi *et al.*, 2020] Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation Learning for Dynamic Graphs: A Survey. In *JMLR*, 2020.

[Lacroix *et al.*, 2018] Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. *ICML*, 2018.

[Nguyen, 2017] Dat Quoc Nguyen. An overview of embedding models of entities and relationships for knowledge base completion. *arXiv preprint arXiv:1703.08098*, 2017.

[Nickel *et al.*, 2011] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, 2011.

[Nickel *et al.*, 2016] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *IEEE*, 2016.

[Paszke *et al.*, 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*. 2019.

[Singhal, 2012] Amit Singhal. Introducing the knowledge graph: things, not strings, 2012.

[Socher *et al.*, 2013] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*, 2013.

[Trouillon *et al.*, 2016] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *ICML*, 2016.

[Trouillon *et al.*, 2017] Théo Trouillon, Christopher R Dance, Éric Gaussier, Johannes Welbl, Sebastian Riedel, and Guillaume Bouchard. Knowledge graph completion via complex tensor factorization. *JMLR*, 2017.

[Wang *et al.*, 2014] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, 2014.

[Wen *et al.*, 2016] Jianfeng Wen, Jianxin Li, Yongyi Mao, Shini Chen, and Richong Zhang. On the representation and embedding of knowledge bases beyond binary relations. In *IJCAI*, 2016.

[Xu *et al.*, 2018] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[Yadati *et al.*, 2018] Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Anand Louis, and Partha Talukdar. Hypergcn: Hypergraph convolutional networks for semi-supervised classification. *arXiv preprint arXiv:1809.02589*, 2018.

[Yang *et al.*, 2015] Bishan Yang, Wen tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *CoRR*, abs/1412.6575, 2015.