Knowledge Hypergraphs: Extending Knowledge Graphs Beyond Binary Relations

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

Abstract

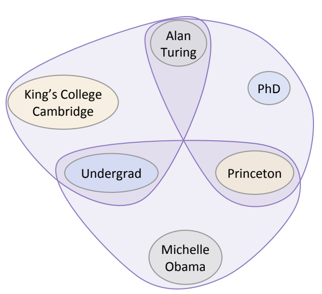
Knowledge graphs store facts using relations between pairs of entities. In this work, we address the question of link prediction in knowledge bases where each relation is defined on *any number* of entities. We represent facts in a *knowledge hypergraph*: a knowledge graph where relations are defined on two or more entities. While there exist techniques (such as reification) that convert the non-binary relations of a knowledge hypergraph into binary ones, current embedding-based methods for knowledge graph completion do not work well out of the box for knowledge graphs obtained through these techniques. Thus we introduce *HypE*, a convolution-based embedding method for knowledge hypergraph completion. We also develop public benchmarks and baselines for our task and show experimentally that HypE is more effective than proposed baselines and existing methods.

1 Introduction

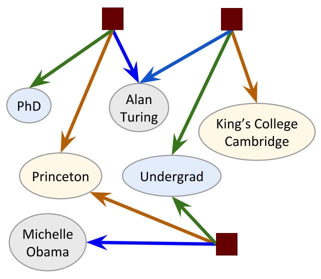
Knowledge Graphs are graph structured knowledge bases that store facts about the world. Such graph structures have applications in several tasks such as search [28] and automatic question answering [12]. A large number of knowledge graphs have been created and are publicly available, such as NELL [5], FREEBASE [2], and Google Knowledge Vault [8]. Since accessing and storing all the facts in the world is difficult, knowledge graphs are incomplete; the goal of *link prediction* (or *knowledge graph completion*) in knowledge graphs is to predict unknown links or relationships between entities based on existing ones. More precisely, knowledge graphs are directed graphs with labeled edges as relations and nodes as entities. These relations are directed from the *head* entity to the *tail* entity. A knowledge graph can be represented as a set of triples, which we denote by $relation(head, tail)$, and that represent information as a collection of binary relations.

Embedding-based models [23, 21, 33] have proved to be effective for knowledge graph completion. These approaches learn embeddings for entities and relations. To find out if $r(h, t)$ is a fact (i.e. is true), such models define a function that embeds relation r and entities h and t, and produces the probability that $r(h, t)$ is a fact. Such embedding-based methods are successful, but they make the strong assumption that all relations are binary (defined between exactly two entities). Wen et al. [35] observe that in Freebase more than $1/3$rd of the entities participate in non-binary relations (defined on more than two entities). We observe in addition, that 61% of the relations in Freebase are non-binary.

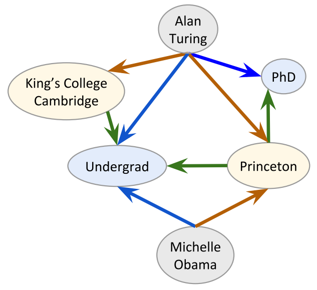
In this paper, we define a *Knowledge Hypergraph* as a generalization of a knowledge graph where relations are defined on two or more entities. We then introduce *HypE*, an embedding-based method for knowledge hypergraph completion that predicts new relations among entities of the hypergraph. HypE uses a new representation for an entity when it appears in a fact based on learned position-dependent convolution filters and entity embeddings. While convolutions are used mainly in vision tasks, Dettmers et al. [7] and Balazevic et al. [1] motivate their use beyond vision by highlighting that



(a) `DEGREE_FROM_UNIVERSITY` defined on three facts.



(b) Reifying non-binary relations with three additional entities.



(c) Converting non-binary relations into cliques.

Figure 1: Converting non-binary relations into binary ones. In this example, the three facts in the original graph (a) show that Turing received his PhD from Princeton and his undergraduate degree from King's College Cambridge. Figures (b) and (c) show two methods of converting this ternary relation into three binary ones.

convolutions are parameter-efficient, fast to compute on a GPU, and have various robust methods to prevent overfitting. We evaluate the proposed method on standard binary and non-binary datasets.

The contributions of this work are: (1) HypE, an embedding-based method for knowledge hypergraph completion that outperforms the baselines for beyond-binary knowledge graphs, (2) a set of baselines for knowledge hypergraph completion, obtained by extending current embedding-based methods and introducing new ones, and (3) two new knowledge hypergraphs obtained from subsets of Freebase, and that can serve as a new benchmark for evaluating knowledge hypergraph completion methods.

2 Motivation and Related Work

Knowledge hypergraph completion is a relatively under-explored area with most of the effective models designed for binary relations. We motivate our current work on link prediction for relations defined on more than two entities by outlining that simply adjusting current approaches to work with hypergraphs do not yield satisfactory results. Existing models can be used in the non-binary setting in either of two possible ways: (1) extending known models to work with non-binary relational data, or (2) converting the non-binary relations into binary ones (using methods such as reification or star-to-clique [35]), and then applying known knowledge graph completion methods.

In the first case, the only known example that extends a known model to work directly with non-binary relations is m-TransH [35], which is an extension of TransH [34]. We describe m-TransH briefly later in this section, and compare its performance to that of our model in Section 6.

The second case is about adjusting the dataset to work with current knowledge graph models. We describe two approaches that could be used to convert the non-binary relations of a knowledge hypergraph into binary. The first approach is *reification*: to *reify* means to "make into an entity". In order to reify a fact with a relation defined on k entities, we first create a new entity and then create k facts, each defining a relation between the new entity and each of the k entities in the given fact. See Figure 1b. The second approach is *star-to-clique*, which converts a fact defined on k entities into $\binom{k}{2}$ facts with distinct relations between all pairwise entities in the fact. See Figure 1c.

Both conversion approaches have their caveats when current embedding-based methods are applied to the resulting graphs. Consider the example in Figure 1. The three facts in this example (Figure 1a) pertain to the relation `DEGREE_FROM_UNIVERSITY`, and show that Turing received his PhD from Princeton and his undergraduate degree from King's College Cambridge, while Michelle Obama received her undergraduate degree from Princeton. When we reify the hypergraph in this example (Figure 1b), we add three new reified entities. Even as this reified knowledge graph has only binary relations and can be used to train an existing model for knowledge graph completion, at test time, we first need to reify the test samples and define a way to embed the newly created entities – about which we have very little information (see Section 5.2 for an example of how we define embeddings for reified entities and how this method compares to our results). On the other hand, when we transform non-binary relations in a knowledge hypergraph into binary through the star-to-clique method, we lose some of the information that we otherwise had in the original hypergraph. In Figure 1c, we can tell that Turing has graduate and undergraduate degrees and that he attended Princeton and King's College Cambridge; but it is no longer clear which degree was granted by which institution.

Other existing methods that relate to our work in this paper can be grouped into three main categories: knowledge graph completion, knowledge hypergraph completion, and learning on hypergraphs. In the remainder of this section, we briefly discuss these approaches.

Knowledge graph completion. Embedding-based models have proved effective for knowledge graphs where all relations are binary. These approaches can be grouped in three main categories: *translational*, *bilinear*, and *deep models*. Translational approaches [3, 34], represent relations as translations in the embedding space. For instance, if a triple is true, then the embedding of the head entity plus the embedding of the relation is close to the embedding of the tail. Bilinear approaches [38, 31, 17] define the score of a triple as the product $v_h^T M_r v_t$ (where v_h and v_t are the vector embeddings of the head and tail entities, and M_r is the matrix embedding of the relation between them). Finally, deep models [22, 29] use neural networks to learn embeddings for each of head, relation, and tail, and compute a score for every triple.

Knowledge hypergraph completion. There is a large family of link prediction models based on soft first-order logic rules [26, 6, 18]. While these models can easily handle variable arity relations and have the advantage of being interpretable, they are known to only learn a subset of patterns that exist in knowledge graphs, and thus are limited in their learning capacity [24]. The model presented in this paper is different from such completion methods, as our method is embedding-based, and consequently is more powerful than soft-rule approaches. The embedding-based work that is closest to our work is m-TransH [35] which extends TransH [34] to knowledge hypergraph completion. Kazemi and Poole [17] (Proposition 2) prove that TransH and other variants of translational approaches are not fully expressive and have restrictions in modeling relations. Similar to TransH, m-TransH is not fully expressive as it inherits the restrictions of TransH in modeling relations. We show in Section 4.1 that our proposed model is fully expressive and compare it to m-TransH in Section 6.

Learning on hypergraphs. Hypergraph learning has been employed to model high-order correlations among data in many computer vision tasks, such as in video object segmentation [14] and in modeling image relationships and image ranking [15]. There is also a line of work extending graph neural network frameworks to hypergraph neural networks [11] and hypergraph convolution networks [37]. On the other hand, graph neural network models are designed for settings where the hypergraph is undirected, with edges that are not labeled (no relations). Knowledge hypergraphs have a different setup, in which predicting a link between (ordered) entities is also a function of the relation combining them. As there is no clear or easy way of extending these graph neural network 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 defined over $\mathcal{E}$ and $\mathcal{R}$. Each tuple in τ is of the form $r(v_1, v_2, \dots, v_k)$ where $r \in \mathcal{R}$ is a relation and each $v_i \in \mathcal{E}$ is an entity, for all $i = 1, 2, \dots, k$. We define the *arity* $|r|$ of a relation r as 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 the entities and relations of the world, and 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 $\mathbf{v_1}, \mathbf{v_2}, \dots, \mathbf{v_k}$ be a set of vectors. The variadic function $concat(\mathbf{v_1}, \dots, \mathbf{v_k})$ outputs the concatenation of its input vectors. The 1D convolution operator $*$ takes as input a vector $\mathbf{v}$ and a convolution weight filter ω, and outputs the convolution of $\mathbf{v}$ with the filters ω. We define the variadic function $\odot()$ to be the sum of the element-wise product of its input vectors, namely $\odot(\mathbf{v_1}, \mathbf{v_2}, \dots, \mathbf{v_k}) = \sum_{i=1}^{\ell} \mathbf{v_1}^{(i)} \mathbf{v_2}^{(i)} \dots \mathbf{v_k}^{(i)}$ where each vector $\mathbf{v_i}$ has the same length, and $\mathbf{v_j}^{(i)}$ is the i-th element of vector $\mathbf{v_j}$.

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 HypE: a Knowledge Hypergraph Embedding Method

In this work we propose HypE, a novel embedding-based method for link prediction in knowledge hypergraphs. The idea at the core of our model is that the way an entity representation is used to make predictions is affected by the role that the entity plays in a given relation. In the example in Figure 1, Turing plays the role of a student at a university, but he may have a different role (e.g. 'professor') in another relation. This means that the way we use Turing's embedding may need to be different for computing predictions for each of these roles.

In several embedding-based methods for knowledge graph completion, such as canonical polyadic [13, 19], ComplEx [31], and SimplE [17], the prediction depends on the position of an entity in a relation. In particular, SimplE learns two embedding vectors $\mathbf{e^{(1)}}$ and $\mathbf{e^{(2)}}$ for an entity e and two embedding vectors $\mathbf{r^{(1)}}$ and $\mathbf{r^{(2)}}$ for a relation r, and computes the score of a triple as $\phi(r(e_1, e_2)) = \odot(\mathbf{r^{(1)}}, \mathbf{e_1^{(1)}}, \mathbf{e_2^{(2)}}) + \odot(\mathbf{r^{(2)}}, \mathbf{e_2^{(1)}}, \mathbf{e_1^{(2)}})$. SimplE can be viewed as a special case of HypE. In what follows, we first provide our formulation of SimplE in terms of convolutions with fixed position-dependent filters; we then build on this formulation to lay out the details of HypE.

SimplE embeds each entity e as a single vector $\mathbf{e} = \mathit{concat}(\mathbf{e^{(1)}}, \mathbf{e^{(2)}})$ and each relation r as a single vector $\mathbf{r} = \mathit{concat}(\mathbf{r^{(1)}}, \mathbf{r^{(2)}})$. It considers four convolutional filters with stride 2 as $\omega_{11} = [1, 0]$ and $\omega_{12} = [0, 1]$ for head and $\omega_{21} = [0, 1]$ and $\omega_{22} = [1, 0]$ for tail. When an entity h appears as head, SimplE uses $\hat{\mathbf{h}} = \mathit{concat}(\mathbf{h} * \omega_{11}, \mathbf{h} * \omega_{12})$ and when an entity t appears as tail, it uses $\hat{\mathbf{t}} = \mathit{concat}(\mathbf{t} * \omega_{21}, \mathbf{t} * \omega_{22})$. In this setting, SimplE computes the score of $r(h, t)$ as $\phi(r(h, t)) = \odot(\mathbf{r}, \hat{\mathbf{h}}, \hat{\mathbf{t}})$.

Instead of using fixed position-dependent filters, HypE learns the filters from the data. The main advantage of learning the filters is that it facilitates extending the formulation to beyond binary, as we can have different (learned) filters for each position. Thus, our model learns embeddings for entities and relations, as well as convolutional weight filters that transform the entity embeddings depending on the position of each in a given relation. For each fact, the transformed entity embeddings are then combined with the embedding of the relation to produce a *score*, *e.g.*, a probability value that the input tuple is true. The architecture of HypE is summarized in Figure 2.

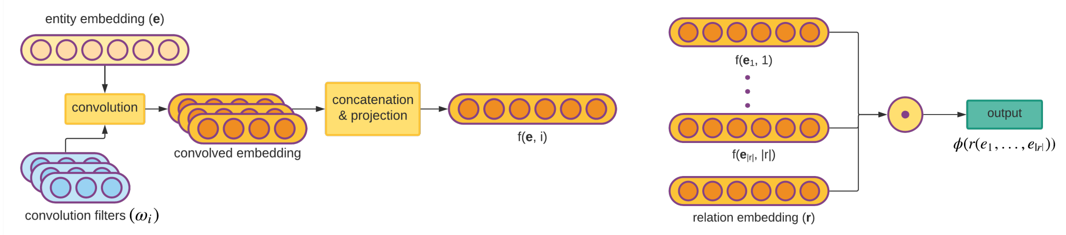
Let n, l, d, and s denote the number of filters per position, the filter-length, the embedding dimension and the stride of the convolution, respectively. Let $\omega_i \in \mathbb{R}^n \times \mathbb{R}^l$ be the convolutional filters associated with an entity at position i, and let $\omega_{ij} \in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

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

4.1 Full Expressivity

Full expressivity of models has been the focus of several studies [10, 32, 36]. A model that is not fully expressive can easily overfit to the training data. The following theorem establishes the full expressivity of HypE. We defer its proof to the *Appendix*.

Theorem 1 (Expressivity) *Let τ be a set of true tuples defined over entities $\mathcal{E}$ and relations $\mathcal{R}$, and let $\delta = \max_{r \in \mathcal{R}}(|r|)$ be the maximum arity of the relations in $\mathcal{R}$. There exists a HypE model with embedding vectors of size at most $\max(\delta|\tau|, \delta)$ that assigns 1 to the tuples in τ and 0 to others.*


entity embedding (**e**)
convolution
convolution filters (ω_i)
convolved embedding
concatenation & projection
f(**e**, i)
f(**e**$_1$, 1)
f(**e**$_{|r|}$, |r|)
relation embedding (**r**)
output
$\phi(r(e_1,\ldots,e_{|r|}))$

(a) Function $f(\mathbf{e}, i)$ (b) Function $\phi(r(e_1, ..., e_{|r|}))$

Figure 2: Visualization of HypE architecture. (a) function $f(\mathbf{e}, i)$ gets an entity embedding and the position the entity appears in the given tuple and returns a vector. (b) function $\phi(r(e_1, \ldots, e_{|r|}))$ gets as input a tuple. The transformed entity embeddings are then combined with $\mathbf{r}$.

4.2 Objective Function and Training

To learn a HypE model, we use stochastic gradient descent with mini-batches. In each learning iteration, we iteratively take in a batch of positive tuples from the knowledge hypergraph. As we only have positive instances available, we need also to train our model on negative instances. For this purpose, for each positive instance, we produce a set of negative instances. For negative sample generation, we follow the contrastive approach of Bordes et al. [3] 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 generate a set of negative samples through the process described above. We define the following cross entropy loss, which is a combination of softmax and negative log likelihood loss, and has been shown to be effective for link prediction [16]:

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

Here, $\{\mathbf{r}\}$ represents relation embeddings, $\{\mathbf{e}\}$ represents entity embeddings, and ϕ is the function given by equation (1) that maps a tuple to a score.

5 Experimental Setup

In this section, we introduce the datasets and the baselines we use to compare to our proposed model. At the end of the section, we discuss the evaluation metrics and implementation details.

5.1 Datasets

We conduct experiments on a total of 5 different datasets (two containing only binary relations, and three with relations of arity 2 to 7). For the experiments on datasets with binary relations, we use two standard benchmarks for knowledge graph completion: WN18 [4] and FB15k [3]. WN18 is a subset of WORDNET [20] and FB15k is a subset of FREEBASE [2]. We use the train, validation, and test split proposed by Bordes et al. [3].

The experiments on knowledge hypergraph completion are conducted on three datasets. The first is JF17K proposed by Wen et al. [35]; 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. Note first that FREEBASE is a reified dataset; that is, it is created from a knowledge base having facts with relations defined on two or more entities. To obtain a knowledge hypergraph H from FREEBASE, we perform an inverse reification process by following the steps below. Table 1 summarizes the statistics of the datasets.

(a) From FREEBASE, remove the facts that have relations defined on a single entity, or that contain numbers or enumeration as entities.

Table 1: Statistics on the datasets.

Dataset	$\|\mathcal{E}\|$	$\|\mathcal{R}\|$	#train	#valid	#test	arities
WN18	40,943	18	141,442	5,000	5,000	{2}
FB15k	14,951	1,345	483,142	50,000	59,071	{2}
JF17K	29,177	327	77,733	–	24,915	{2, 3, 4, 5, 6, 7}
FB-AUTO	3,410	8	6,778	2,255	2,180	{2, 4, 5}
M-FB15K	10,314	71	415,375	39,348	38,797	{2, 3, 4, 5}

(b) Convert the triples in FREEBASE that share the same entity into facts in H. For example, the triples $r_0(id123, e_i)$, $r_1(id123, e_j)$, and $r_2(id123, e_k)$, which were originally created by the addition of the (unique) reified entity $id123$, now represent fact $r(e_i, e_j, e_k)$ in H.

(c) Create the FB-AUTO dataset by selecting the facts from H whose subject is 'automotive'.

(d) Create the M-FB15K dataset by following a strategy similar to that proposed by Bordes et al. [3]: select the facts in H that pertain to entities present in the Wikilinks database [27].

(e) Split the facts in each of FB-AUTO and M-FB15K randomly into train, test, and validation sets.

5.2 Baselines

To compare our results to that of existing work, we first need to come up with some baselines for knowledge hypergraph completion. We achieve this by either extending current models on knowledge graph completion or by introducing new ones. The baselines we introduce in this work are grouped into the following three categories: (1) methods that work with binary relations and that are easily extendable to higher-arity: r-SimplE, m-DistMult, and m-CP; (2) simple (but non-trivial) extensions of current methods: m-SimplE, Shift1Left; and (3) existing methods that can handle higher-arity relations: m-TransH. Below we give some details about these proposed baselines:

r-SimplE: To test how well a model trained on reified data performs in practice, we converted higher-arity relations in the train set to binary relations through reification. We then use the SimplE model (that we call r-SimplE) to train and test 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 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 being replaced by three facts: $r_1(id123, e_1)$, $r_2(id123, e_2)$, and $r_3(id123, e_3)$. When predicting the tail entity of $r_1(id123, ?)$, we use the other two reified facts to learn an embedding for entity $id123$. Because $id123$ is added only to help represent the higher-arity relations as a set of binary relations, we only do tail prediction for reified relations.

m-DistMult: DistMult [38] 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 score function as $\phi(r(e_i, \ldots, e_j)) = \odot(\mathbf{r}, \mathbf{e_i}, \ldots, \mathbf{e_j})$.

m-CP: Canonical Polyadic (CP) decomposition [13] embeds each entity e as two vectors $\mathbf{e^{(1)}}$ and $\mathbf{e^{(2)}}$, and each relation r as a single vector $\mathbf{r}$. CP 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 a variant (m-CP) that accommodates non-binary relations, and which embeds each entity e as δ different vectors $\mathbf{e^{(1)}}, .., \mathbf{e}^{(\delta)}$, where $\delta = \max_{r \in \mathcal{R}}(|r|)$. m-CP computes the score of a tuple as $\phi(r(e_i, \ldots, e_j)) = \odot(\mathbf{r}, \mathbf{e_i^{(1)}}, ..., \mathbf{e_j^{(|r|)}})$.

m-SimplE: SimplE [17] embeds each entity e as two vectors $\mathbf{e^{(1)}}$ and $\mathbf{e^{(2)}}$, and each relation r as two vectors $\mathbf{r^{(1)}}$ and $\mathbf{r^{(2)}}$. We reformulate SimplE as embedding each e and r as vectors $\mathbf{e}$ and $\mathbf{r}$, and defining the score as $\phi(r(e_i, e_j)) = \odot(\mathbf{r}, \mathbf{e_i}, shift(\mathbf{e_j}, len(\mathbf{e_j})/2))$. Here, $shift(\mathbf{a}, x)$ shifts vector $\mathbf{a}$ to the left by x steps and $len(\mathbf{e})$ returns length of vector $\mathbf{e}$. The encoding of SimplE is a special instance of the above encoding, with $\mathbf{e} = concat(\mathbf{e}^1, \mathbf{e}^2)$ and $\mathbf{r} = concat(\mathbf{r}^1, \mathbf{r}^2)$. The m-SimplE score function is defined as $\phi(r(e_i, e_j, \ldots, e_k)) = \odot(\mathbf{r}, \mathbf{e_i}, shift(\mathbf{e_j}, len(\mathbf{e_j})/\delta), ..., shift(\mathbf{e_k}, len(\mathbf{e_k}) * (\delta - 1)/\delta)$ where $\delta = \max_{r \in \mathcal{R}}(|r|)$.

Shift1Left: Shift1Left works similar to m-SimplE. Shift1Left shifts entity embeddings to the left and computes the score with $\phi(r(e_i, e_j, ..., e_k) = \odot(\mathbf{r}, \mathbf{e_i}, shift(\mathbf{e_j}, 1), ..., shift(\mathbf{e_k}, |r| - 1))$.

Table 2: 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. Our method outperforms the baselines on all datasets.

	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	0.490	0.398	0.539	0.659	0.789	0.751	0.816	0.850	0.700	0.627	0.736	0.841
m-CP	0.521	0.443	0.560	0.665	0.797	0.768	0.815	0.855	0.744	0.687	0.769	0.857
Shift1Left	0.528	0.444	0.571	0.685	0.799	0.770	0.826	0.851	0.729	0.664	0.760	0.854
m-SimplE	0.530	0.447	0.573	0.687	0.800	0.771	0.820	0.851	0.727	0.661	0.759	0.856
m-TransH [35]	0.775	0.770	0.782	0.783	0.808	0.775	0.812	0.862	0.723	0.718	0.720	0.725
HypE (Ours)	**0.827**	**0.799**	**0.858**	**0.859**	**0.907**	**0.839**	**0.963**	**1.00**	**0.795**	**0.730**	**0.815**	**0.875**

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 $r(e_1, \dots, e_k)$ 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 $r(e_1, \dots, e_i^c, \dots, e_k)$ where $e_i^c \in \mathcal{E} \setminus \{e_i\}$. Let the set of corrupted tuples, plus $r(e_1, \dots, e_k)$, be denoted by $\theta_i(r(e_1, \dots, e_k))$. Let $\mathrm{rank}_i(r(e_1, \dots, e_k))$ be the ranking of $r(e_1, \dots, e_k)$ within $\theta_i(r(e_1, \dots, e_k))$ based on the score $\phi(x)$ for each $x \in \theta_i(r(e_1, \dots, e_k))$. In an ideal knowledge hypergraph completion method, the rank $\mathrm{rank}_i(r(e_1, \dots, e_k))$ is 1 among all corrupted tuples. We compute the MRR as $\frac{1}{K} \sum_{r(e_1,\dots,e_k) \in \tau'_{test}} \sum_{i=1}^{k} \frac{1}{\mathrm{rank}_i r(e_1,\dots,e_k)}$ where $K = \sum_{r(e_1,\dots e_k) \in \tau'_{test}} |r|$ is the number of prediction tasks. Hit@t measures the proportion of tuples in τ'_{test} that rank among top t in their corresponding corrupted sets. We follow Bordes et al. [3] and remove all corrupted tuples that are in τ' from our computation of MRR and Hit@t.

5.4 Implementation Details

We implement HypE and the baselines in PyTorch [25]. We use Adagrad [9] as the optimizer and dropout [30] to regularize our model and baselines. We tune our hyperparameters over the validation set, and fix the maximum number of epochs to 500 and batch size to 128. We set the embedding size and negative ratio to 200 and 10 respectively. We compute the MRR of models over the validation set every 50 epochs and select the epoch that results the best. The learning rate and dropout rate of all models are tuned. HypE has n, l and s as hyperparameters. We select the hyperparameters of HypE and baselines via the same grid search based on MRR on the validation. The code of the proposed model, the baselines, and the datasets will be available upon acceptance of the paper.

6 Experiments

In this section, we evaluate HypE on binary and non-binary facts. Our method clearly outperforms m-TransH and the proposed baselines. As ablation study, we compute the breakdown performance of the models on different arities. To assess the performance of our method on binary relations, we evaluate it on WN18 and FB15K. We test our model on WN18 and FB15K, as the methods we compare against report results on only these datasets.

6.1 Knowledge Hypergraph Completion Results

Table 2 shows the knowledge hypergraph completion results for the proposed baselines and HypE across three datasets. Our model outperforms the proposed baselines on JF17K, FB-AUTO, and M-FB15K by a large margin. These results represent the clear advantage of HypE when higher-arity relations are available. The results also show that reification for the r-SimplE model does not work well; this is probably because the reification process introduces auxiliary entities that appear in very few facts, based on which the model is not able to learn an appropriate embedding. Comparing

Table 3: Breakdown performance of Hit@10 across relations with different arities on JF17K.

	Arity					
Model	2	3	4	5	6	All
r-SimplE	0.478	0.025	0.015	0.022	0.000	0.168
m-DistMult	0.484	0.680	0.851	0.960	0.792	0.659
m-CP	0.465	0.705	0.840	0.968	0.969	0.665
Shift1Left	0.497	0.725	0.856	0.974	0.271	0.685
m-SimplE	0.498	0.718	0.857	**0.976**	0.583	0.687
m-TransH [35]	0.748	0.865	0.744	0.964	0.803	0.783
HypE (Ours)	**0.906**	**0.870**	**0.972**	0.863	**1.00**	**0.859**

Table 4: Knowledge graph completion results on WN18 and FB15K for baselines and our model. Our model performs similar to the best baselines for knowledge graphs with binary relations.

	WN18				FB15k			
Model	MRR	Hit@1	Hit@3	Hit@10	MRR	Hit@1	Hit@3	Hit@10
CP [13]	0.074	0.049	0.080	0.125	0.326	0.219	0.376	0.532
TransH [34]	-	-	-	0.867	-	-	-	0.585
m-TransH [35]	0.671	0.495	0.839	0.923	0.351	0.228	0.427	0.559
DistMult [38]	0.822	0.728	0.914	0.936	0.654	0.546	0.733	0.824
SimplE [17]	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

the results on r-SimplE and m-SimplE we can also see that extending a model works better than reification when higher-arity relations are present.

6.2 Ablation Study on Different Arities

For each of the baselines and HypE, we break down the performance across relations with different arities. Table 3 shows Hit@10 of the models for each arity in JF17K. We observe that HypE outperforms the baselines in all arities except arity 5. Its improved performance for relations with arity more than 2 may be one reason why its performance on binary relations improves as well. Note that HypE is designed to handle relations of any arity.

6.3 Knowledge Graph Completion Results

Table 4 shows link prediction results on WN18 and FB15K. Baseline results are taken from the original papers except that of m-TransH, which we implement it ourself. To be fair when comparing our model to the baselines, we follow the Kazemi and Poole [17] setup with the same grid search approach: we set $n = 2$, $l = 2$, and $s = 2$ so our models have the same number of parameters. This makes our our results directly comparable to knowledge graph completion methods, which sohw that HypE outperforms m-TransH on WN18 and FB15K. As we show in Section 4, SimplE can be formulated as a special case of HypE. This is also reflected in the results, as SimplE and HypE get comparable outcomes in the binary setting when they have the same number of parameters.

7 Conclusion and Future Work

In this paper, we represent facts as a knowledge hypergraph: a graph where labeled edges are defined on two or more nodes. We propose HypE, a knowledge hypergraph completion method that embeds entities and relations, and predicts new links in a knowledge hypergraph. We introduce baselines for completing knowledge hypergraphs by extending current methods and introducing new ones. We also introduce two datasets for evaluating knowledge hypergraph completion methods by compiling subsets of Freebase. Based on our benchmarks, HypE achieves results comparable to the baselines for knowledge graph completion (having only binary relations), and outperforms the state of the art by a large margin for knowledge hypergraphs (having relations that are defined on two or more entities).

References

[1] Ivana Balazevic, Carl Allen, and Timothy M Hospedales. Hypernetwork knowledge graph embeddings. *arXiv preprint arXiv:1808.07018*, 2018.

[2] 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.

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

[4] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259, 2014.

[5] 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.

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

[7] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, 2018.

[8] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *ACM SIGKDD*, 2014.

[9] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 12(Jul):2121–2159, 2011.

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

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

[12] 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*, 31(3):59–79, 2010.

[13] 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.

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

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

[16] Rudolf Kadlec, Ondrej Bajgar, and Jan Kleindienst. Knowledge base completion: Baselines strike back. In *RepL4NLP*, 2017.

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

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

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

[20] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11): 39–41, 1995.

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

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

[23] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing yago: scalable machine learning for linked data. In *ACM WWW*, 2012.

[24] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[25] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch, 2017.

[26] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2): 107–136, 2006.

[27] Sameer Singh, Amarnag Subramanya, Fernando Pereira, and Andrew McCallum. Wikilinks: A large-scale cross-document coreference corpus labeled via links to wikipedia. *University of Massachusetts, Amherst, Tech. Rep. UM-CS-2012*, 15, 2012.

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

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

[30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

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

[32] Théo Trouillon, Christopher R Dance, Éric Gaussier, Johannes Welbl, Sebastian Riedel, and Guillaume Bouchard. Knowledge graph completion via complex tensor factorization. *JMLR*, 18(1):4735–4772, 2017.

[33] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *TKDE*, 29(12):2724–2743, 2017.

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

[35] 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.

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

[37] 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.

[38] 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.

A Appendix

Proof of Theorem 1

Theorem 1 (Expressivity) Let τ be a set of true tuples defined over entities $\mathcal{E}$and relations $\mathcal{R}$, and let $\delta = \max_{r \in \mathcal{R}}(|r|)$ be the maximum arity of the relations in $\mathcal{R}$. Then there exists a HypE model with embedding vectors each of size at most $\max(\delta|\tau|, \delta)$ that assigns 1 to the tuples in τ and 0 to tuples not in τ.

Proof: 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 is as follows:

$$\phi(x) = \begin{cases} 0 & \text{if } x \in \tau \\ 1 & \text{otherwise} \end{cases}$$

We begin the proof by first describing the embeddings of each of the entities and relations in HypE; we then proceed to show that with such an embedding, HypE can represent any world accurately.

Let us first assume that $|\tau| > 0$ and let f_p be the pth fact in τ. We let each entity $e \in \mathcal{E}$ be represented with a vector of length $\delta|\tau|$ in which the pth block of δ-bits is the one-hot representation of e in fact f_p: if e appears in fact f_p at position i, then the ith bit of the pth block is set to 1, and to 0 otherwise. Each relation $r \in \mathcal{R}$ is then represented as a vector of length τ whose pth bit is equal to 1 if fact f_p is defined on relation r, and 0 otherwise.

HypE defines different convolutional weight filters for each entity position within a tuple. As we have at most δ possible positions, we define each convolutional filter ω_i as a vector of length δ where the ith bit is set to 1 and all others to 0, for each $i = 1, 2, \ldots, \delta$. When the scoring function ϕ is applied to some tuple x, for each entity position i in x, convolution filter ω_i is applied to the entity at position i in the tuple as a first step; the $\odot()$ function is then applied to the resulting vector and the relation embedding to obtain a score.

Given any tuple x, we want to show that $\phi(x) = 1$ if $x \in \tau$ and 0 otherwise.

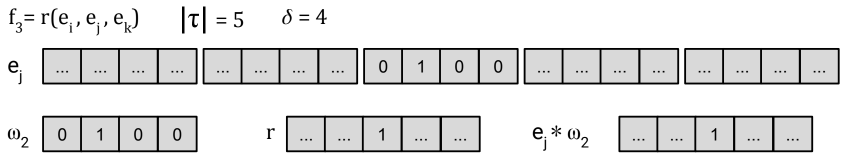


Figure 3: An example of an embedding where $|\tau| = 5$, $\delta = 4$ and f_3 is the third fact in τ

First assume that $x = f_p$ is the pth fact in τ that is defined on relation r and entities where e_i is the entity at position i. Convolving each e_i with ω_i results in a vector of length $|\tau|$ where the pth bit is equal to 1 (since both ω_i and the pth block of e_i have a 1 at the ith position) (See Figure 3. Then, as a first step, function $\odot()$ computes the element-wise multiplication between the embedding of relation r (that has 1 at position p) and all of the convolved entity vectors (each having 1 at position p); this results in a vector of length $|\tau|$ where the pth bit is set to 1 and all other bits set to 0. Finally, $\odot(())$ sums the outcome of the resulting products to give us a score of 1.

To show that $\phi(x) = 0$ when $x \notin \tau$, we prove the contrapositive, namely that if $\phi(x) = 1$, then x must be a fact in τ. We proceed by contradiction. Assume that there exists a tuple $x \notin \tau$ such that $\phi(x) = 1$. This means that at the time of computing the element-wise product in the $\odot()$ function, there was a position j at which all input vectors to $\odot()$ had a value of 1. This can happen only when (1) applying the convolution filter w_j to each of the entities in x produces a vector having 1 at position j, and (2) the embedding of relation $r \in x$ has 1 at position j.

The first case can happen only if all entities of x appear in the jth fact $f_j \in \tau$; the second case happens only if relation $r \in x$ appears in f_j. But if all entities of x as well as its relation appear in fact f_j, then $x \in \tau$, contradicting our assumption. Therefore, if $\phi(x) = 1$, then x must be a fact in τ.

To complete the proof, we consider the case when $|\tau| = 0$. In this case, since there are no facts, all entities and relations are represented by zero-vectors of length δ. Then, for any tuple x, $\phi(x) = 0$. This completes the proof. $\square$