Hypernetwork Knowledge Graph Embeddings

Ivana Balažević[1], Carl Allen[1], and Timothy M. Hospedales[1,2]

[1] School of Informatics, University of Edinburgh, UK
[2] Samsung AI Centre, Cambridge, UK
{ivana.balazevic, carl.allen, t.hospedales}@ed.ac.uk

Abstract. Knowledge graphs are graphical representations of large databases of facts, which typically suffer from incompleteness. Inferring missing relations (links) between entities (nodes) is the task of link prediction. A recent state-of-the-art approach to link prediction, ConvE, implements a convolutional neural network to extract features from concatenated subject and relation vectors. Whilst results are impressive, the method is unintuitive and poorly understood. We propose a hypernetwork architecture that generates simplified relation-specific convolutional filters that (i) outperforms ConvE and all previous approaches across standard datasets; and (ii) can be framed as tensor factorization and thus set within a well established family of factorization models for link prediction. We thus demonstrate that convolution simply offers a convenient computational means of introducing sparsity and parameter tying to find an effective trade-off between non-linear expressiveness and the number of parameters to learn.

1 Introduction

Knowledge graphs, such as WordNet, Freebase, and Google Knowledge Graph, are large graph-structured databases of facts, containing information in the form of triples (e_1, r, e_2), with e_1 and e_2 representing subject and object entities and r a relation between them. They are considered important information resources, used for a wide variety of tasks ranging from question answering to information retrieval and text summarization. One of the main challenges with existing knowledge graphs is their incompleteness: many of the links between entities in the graph are missing. This has inspired substantial work in the field of *link prediction*, i.e. the task of inferring missing links in knowledge graphs.

Until recently, many approaches to link prediction have been based on different factorizations of a 3-moded binary tensor representation of the training triples [12,17,23,22]. Such approaches are shallow and linear, with limited expressiveness. However, attempts to increase expressiveness with additional fully connected layers and non-linearities often lead to overfitting [12,17]. For this reason, Dettmers et al. introduce ConvE, a model that uses 2D convolutions over reshaped and concatenated entity and relation embeddings [3]. They motivate the use of convolutions by being parameter efficient and fast to compute on a GPU, as well as having various robust methods from computer vision to prevent

overfitting. Even though results achieved by ConvE are impressive, it is highly unintuitive that convolution – particularly 2D convolution – should be effective for extracting information from 1D entity and relation embeddings.

In this paper, we introduce HypER, a model that uses a *hypernetwork* [5] to generate convolutional filter weights for each relation. A hypernetwork is an approach by which one network generates weights for another network, that can be used to enable weight-sharing across layers and to dynamically synthesize weights given an input. In our context, we generate relation-specific filter weights to process input entities, and also achieve *multi-task knowledge sharing* across relations in the knowledge graph. Our proposed HypER model uses a hypernetwork to generate a set of 1D relation-specific filters to process the subject entity embeddings. This simplifies the interaction between subject entity and relation embeddings compared to ConvE, in which a global set of *2D* filters are convolved over *reshaped and concatenated* subject entity and relation embeddings, which is unintuitive as it suggests the presence of 2D structure in word embeddings. Moreover, interaction between subject and relation in ConvE depends on an arbitrary choice about how they are reshaped and concatenated. In contrast, HypER's hypernetwork generates relation-specific filters, and thus extracts *relation-specific features* from the subject entity embedding. This necessitates no 2D reshaping, and allows entity and relation to interact more completely, rather than only around the concatenation boundary. We show that this simplified approach, in addition to improving link prediction performance, can be understood in terms of tensor factorization, thus placing HypER within a well established family of factorization models. The apparent obscurity of using convolution within word embeddings is thereby explained as simply a convenient computational means of introducing *sparsity* and *parameter tying*.

We evaluate HypER against several previously proposed link prediction models using standard datasets (FB15k-237, WN18RR, FB15k, WN18, YAGO3-10), across which it consistently achieves state-of-the-art performance. In summary, our key contributions are:

- proposing a new model for link prediction (HypER) which achieves state-of-the-art performance across all standard datasets;
- showing that the benefit of using convolutional instead of fully connected layers is due to restricting the number of dimensions that interact (i.e. explicit regularization), rather than finding higher dimensional structure in the embeddings (as implied by ConvE); and
- showing that HypER in fact falls within a broad class of tensor factorization models despite the use of convolution, which serves to provide a good trade-off between expressiveness and number of parameters to learn.

2 Related Work

Numerous matrix factorization approaches to link prediction have been proposed. An early model, RESCAL [12], tackles the link prediction task by optimizing a scoring function containing a bilinear product between vectors for each

of the subject and object entities and a full rank matrix for each relation. DistMult [23] can be viewed as a special case of RESCAL with a diagonal matrix per relation type, which limits the linear transformation performed on entity vectors to a stretch. ComplEx [22] extends DistMult to the complex domain. TransE [1] is an affine model that represents a relation as a translation operation between subject and object entity vectors.

A somewhat separate line of link prediction research introduces Relational Graph Convolutional Networks (R-GCNs) [15]. R-GCNs use a convolution operator to capture locality information in graphs. The model closest to our own and which we draw inspiration from, is ConvE [3], where a convolution operation is performed on the subject entity vector and the relation vector, after they are each reshaped to a matrix and lengthwise concatenated. The obtained feature maps are flattened, put through a fully connected layer, and the inner product is taken with all object entity vectors to generate a score for each triple. Advantages of ConvE over previous approaches include its expressiveness, achieved by using multiple layers of non-linear features, its scalability to large knowledge graphs, and its robustness to overfitting. However, it is not intuitive why convolving across concatenated and reshaped subject entity and relation vectors should be effective.

The proposed HypER model does no such reshaping or concatenation and thus avoids both implying any inherent 2D structure in the embeddings and restricting interaction to the concatenation boundary. Instead, HypER convolves *every* dimension of the subject entity embedding with *relation-specific* convolutional filters generated by the hypernetwork. This way, entity and relation embeddings are combined in a non-linear (quadratic) manner, unlike the linear combination (weighted sum) in ConvE. This gives HypER more expressive power, while also reducing parameters.

Interestingly, we find that the differences in moving from ConvE to HypER in fact bring the factorization and convolutional approaches together, since the 1D convolution process is equivalent to multiplication by a highly sparse tensor with tied weights (see Figure 2). The multiplication of this "convolutional tensor" (defined by the relation embedding and hypernetwork) and other weights gives an implicit relation matrix, corresponding to those in e.g. RESCAL, DistMult and ComplEx. Other than the method of deriving these relation matrices, the key difference to existing factorization approaches is the ReLU non-linearity applied prior to interaction with the object embedding.

3 Link Prediction

In link prediction, the aim is to learn a scoring function ϕ that assigns a score $s = \phi(e_1, r, e_2) \in \mathbb{R}$ to each input triple (e_1, r, e_2), where $e_1, e_2 \in \mathcal{E}$ are subject and object entities and $r \in \mathcal{R}$ a relation. The score indicates the strength of prediction that the given triple corresponds to a true fact, with positive scores meaning true and negative scores, false. Link prediction models typically map entity pair e_1, e_2 to their corresponding distributed embedding representations $\mathbf{e}_1, \mathbf{e}_2 \in \mathbb{R}^{d_e}$ and a score is assigned using a *relation-specific* function,

Table 1. Scoring functions of state-of-the-art link prediction models, the dimensionality of their relation parameters, and their space complexity. d_e and d_r are the dimensions of entity and relation embeddings respectively, $\overline{\mathbf{e}}_2 \in \mathbb{C}^{d_e}$ denotes the complex conjugate of $\mathbf{e}_2$, and $\underline{\mathbf{e}}_1, \underline{\mathbf{w}}_r \in \mathbb{R}^{d_w \times d_h}$ denote a 2D reshaping of $\mathbf{e}_1$ and $\mathbf{w}_r$ respectively. $*$ is the convolution operator, $\mathbf{F}_r = \text{vec}^{-1}(\mathbf{w}_r\mathbf{H})$ the matrix of relation specific convolutional filters, vec is a vectorization of a matrix and vec^{-1} its inverse, f is a non-linear function, and n_e and n_r respectively denote the number of entities and relations.

Model	Scoring Function	Relation Parameters	Space Complexity
RESCAL [12]	$\mathbf{e}_1^\top \mathbf{W}_r \mathbf{e}_2$	$\mathbf{W}_r \in \mathbb{R}^{{d_e}^2}$	$\mathcal{O}(n_e d_e + n_r d_e^2)$
TransE [1]	$\lVert \mathbf{e}_1 + \mathbf{w}_r - \mathbf{e}_2 \rVert$	$\mathbf{w}_r \in \mathbb{R}^{d_e}$	$\mathcal{O}(n_e d_e + n_r d_e)$
NTN [17]	$\mathbf{u}_r^\top f(\mathbf{e}_1 \mathbf{W}_r^{[1..k]} \mathbf{e}_2 + \mathbf{V}_r \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \end{bmatrix} + \mathbf{b}_r)$	$\mathbf{W}_r \in \mathbb{R}^{{d_e}^2 k}, \mathbf{V}_r \in \mathbb{R}^{2{d_e}k}$, $\mathbf{u}_r \in \mathbb{R}^k, \mathbf{b}_r \in \mathbb{R}^k$	$\mathcal{O}(n_e d_e + n_r {d_e}^2 k)$
DistMult [23]	$\langle \mathbf{e}_1, \mathbf{w}_r, \mathbf{e}_2 \rangle$	$\mathbf{w}_r \in \mathbb{R}^{d_e}$	$\mathcal{O}(n_e d_e + n_r d_e)$
ComplEx [22]	$\text{Re}(\langle \mathbf{e}_1, \mathbf{w}_r, \overline{\mathbf{e}}_2 \rangle)$	$\mathbf{w}_r \in \mathbb{C}^{d_e}$	$\mathcal{O}(n_e d_e + n_r d_e)$
ConvE [3]	$f(\text{vec}(f([\underline{\mathbf{e}}_1; \underline{\mathbf{w}}_r] * w))\mathbf{W})\mathbf{e}_2$	$\mathbf{w}_r \in \mathbb{R}^{d_r}$	$\mathcal{O}(n_e d_e + n_r d_r)$
HypER (ours)	$f(\text{vec}(\mathbf{e}_1 * \text{vec}^{-1}(\mathbf{w}_r\mathbf{H}))\mathbf{W})\mathbf{e}_2$	$\mathbf{w}_r \in \mathbb{R}^{d_r}$	$\mathcal{O}(n_e d_e + n_r d_r)$

$s = \phi_r(\mathbf{e}_1, \mathbf{e}_2)$. The majority of link prediction models apply the logistic sigmoid function $\sigma(\cdot)$ to the score to give a probabilistically interpretable prediction $p = \sigma(s) \in [0, 1]$ as to whether the queried fact is true. The scoring functions for models from across the literature and HypER are summarized in Table 1, together with the dimensionality of their relation parameters and the significant terms of their space complexity.

4 Hypernetwork Knowledge Graph Embeddings

In this work, we propose a novel hypernetwork model for link prediction in knowledge graphs. In summary, the hypernetwork projects a vector embedding of each relation via a fully connected layer, the result of which is reshaped to give a set of convolutional filter weight vectors for each relation. We explain this process in more detail below. The idea of using convolutions on entity and relation embeddings stems from computer vision, where feature maps reflect patterns in the image such as lines or edges. Their role in the text domain is harder to interpret, since little is known of the meaning of a single dimension in a word embedding. We believe convolutional filters have a regularizing effect when applied to word embeddings (compared to the corresponding full tensor), as the filter size restricts which dimensions of embeddings can interact. This allows nonlinear expressiveness while limiting overfitting by using few parameters. A visualization of HypER is given in Figure 1.

4.1 Scoring Function and Model Architecture

The relation-specific scoring function for the HypER model is:

$$\begin{aligned} \phi_r(\mathbf{e}_1, \mathbf{e}_2) &= f(\text{vec}(\mathbf{e}_1 * \mathbf{F}_r)\mathbf{W})\mathbf{e}_2 \\ &= f(\text{vec}(\mathbf{e}_1 * \text{vec}^{-1}(\mathbf{w}_r\mathbf{H}))\mathbf{W})\mathbf{e}_2, \end{aligned} \tag{1}$$

where the vec^{-1} operator reshapes a vector to a matrix, and non-linearity f is chosen to be a rectified linear unit (ReLU).

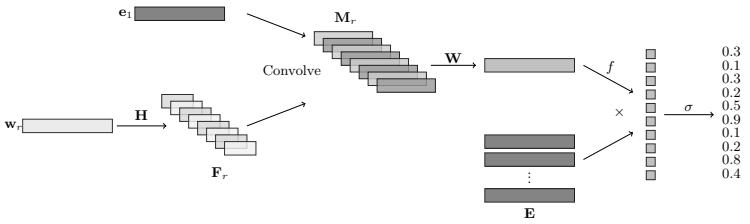

Fig. 1. Visualization of the HypER model architecture. Subject entity embedding $\mathbf{e}_1$ is convolved with filters $\mathbf{F}_r$, created by the hypernetwork $\mathbf{H}$ from relation embedding $\mathbf{w}_r$. The obtained feature maps $\mathbf{M}_r$ are mapped to d_e-dimensional space via $\mathbf{W}$ and the non-linearity f applied before being combined with all object vectors $\mathbf{e}_2 \in \mathbf{E}$ through an inner product to give a score for each triple. Predictions are obtained by applying the logistic sigmoid function to each score.

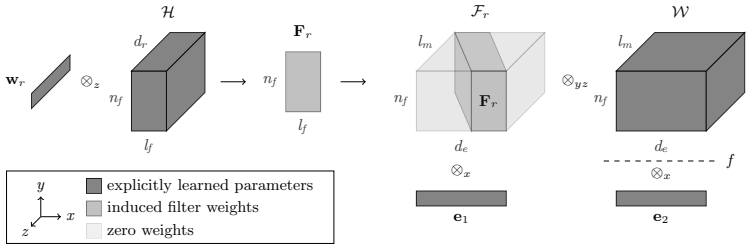

Fig. 2. Interpretation of the HypER model in terms of tensor operations. Each relation embedding $\mathbf{w}_r$ generates a set of filters $\mathbf{F}_r$ via the hypernetwork $\mathcal{H}$. The act of convolving $\mathbf{F}_r$ over $\mathbf{e}_1$ is equivalent to multiplication of $\mathbf{e}_1$ by a tensor $\mathcal{F}_r$ (in which $\mathbf{F}_r$ is diagonally duplicated and zero elsewhere). The tensor product $\mathcal{F}_r \otimes_{yz} \mathcal{W}$ gives a $d_e \times d_e$ matrix specific to each relation. Axes labels indicate the modes of tensor interaction (via inner product).

In the feed-forward pass, the model obtains embeddings for the input triple from the entity and relation embedding matrices $\mathbf{E} \in \mathbb{R}^{n_e \times d_e}$ and $\mathbf{R} \in \mathbb{R}^{n_r \times d_r}$. The hypernetwork is a fully connected layer $\mathbf{H} \in \mathbb{R}^{d_r \times l_f n_f}$ (l_f denotes filter length and n_f the number of filters per relation, i.e. *output channels* of the convolution) that is applied to the relation embedding $\mathbf{w}_r \in \mathbb{R}^{d_r}$. The result is reshaped to generate a matrix of convolutional filters $\mathbf{F}_r = \text{vec}^{-1}(\mathbf{w}_r \mathbf{H}) \in \mathbb{R}^{l_f \times n_f}$. Whilst the overall dimensionality of the filter set is $l_f n_f$, the rank is restricted to d_r to encourage parameter sharing between relations.

The subject entity embedding $\mathbf{e}_1$ is convolved with the set of relation-specific filters $\mathbf{F}_r$ to give a 2D feature map $\mathbf{M}_r \in \mathbb{R}^{l_m \times n_f}$, where $l_m = d_e - l_f + 1$ is the feature map length. The feature map is vectorized to $\text{vec}(\mathbf{M}_r) \in \mathbb{R}^{l_m n_f}$, and projected to d_e-dimensional space by the weight matrix $\mathbf{W} \in \mathbb{R}^{l_m n_f \times d_e}$. After applying a ReLU activation function, the result is combined by way of inner product with each and every object entity embedding ${\mathbf{e}_2}^{(i)}$, where i varies over all entities in the dataset (of size n_e), to give a vector of scores. The logistic sigmoid is applied element-wise to the score vector to obtain the predicted probability of each prospective triple being true $\mathbf{p}_i = \sigma(\phi_r(\mathbf{e}_1, {\mathbf{e}_2}^{(i)}))$.

4.2 Understanding HypER as Tensor Factorization

Having described the HypER architecture, we can view it as a series of tensor operations by considering the hypernetwork $\mathbf{H}$ and weight matrix $\mathbf{W}$ as tensors $\mathcal{H} \in \mathbb{R}^{d_r \times l_f \times n_f}$ and $\mathcal{W} \in \mathbb{R}^{l_m \times n_f \times d_e}$ respectively. The act of convolving $\mathbf{F}_r = \mathbf{w}_r \otimes \mathcal{H}$ over the subject entity embedding $\mathbf{e}_1$ is equivalent to the multiplication of $\mathbf{e}_1$ by a sparse tensor $\mathcal{F}_r$ within which $\mathbf{F}_r$ is diagonally duplicated with zeros elsewhere (see Figure 2). The result is multiplied by $\mathcal{W}$ to give a vector, which is subject to ReLU before the final dot product with $\mathbf{e}_2$. Linearity allows the product $\mathcal{F}_r \otimes \mathcal{W}$ to be considered separately as generating a $d_e \times d_e$ matrix for each relation. Further, rather than duplicating entries of $\mathbf{F}_r$ within $\mathcal{F}_r$, we can generalize $\mathcal{F}_r$ to a relation-agnostic sparse 4 moded tensor $\mathcal{F} \in \mathbb{R}^{d_r \times d_e \times n_f \times l_m}$ by replacing entries with d_r-dimensional strands of $\mathcal{H}$. Thus, the HypER model can be described explicitly as tensor multiplication of $\mathbf{e}_1, \mathbf{e}_2$ and $\mathbf{w}_r$ with a core tensor $\mathcal{F} \otimes \mathcal{W} \in \mathbb{R}^{d_e \times d_e \times d_r}$, where $\mathcal{F}$ is heavily constrained in terms of its number of free variables. This insight allows HypER to be viewed in a very similar light to the family of factorization approaches to link prediction, such as RESCAL, DistMult and ComplEx.

4.3 Training Procedure

Following the training procedure introduced by [3], we use *1-N scoring* with the Adam optimizer [8] to minimize the binary cross-entropy loss:

$$\mathcal{L}(\mathbf{p}, \mathbf{y}) = -\frac{1}{n_e} \sum_i (\mathbf{y}_i \log(\mathbf{p}_i) + (1 - \mathbf{y}_i) \log(1 - \mathbf{p}_i)), \tag{2}$$

where $\mathbf{y} \in \mathbb{R}^{n_e}$ is the label vector containing ones for true triples and zeros otherwise, subject to *label smoothing*. **Label smoothing** is a widely used technique shown to improve generalization [20,14]. Label smoothing changes the ground-truth label distribution by adding a uniform prior to encourage the model to be less confident, achieving a regularizing effect. **1-N scoring** refers to simultaneously scoring $(e_1, r, \mathcal{E})$, i.e. for all entities $e_2 \in \mathcal{E}$, in contrast to *1-1 scoring*, the practice of training individual triples (e_1, r, e_2) one at a time. As shown by [3], 1-N scoring offers a significant speedup (3x on train and 300x on test time) and improved accuracy compared to 1-1 scoring. A potential extension of the HypER model described above would be to apply convolutional filters to *both* subject and object entity embeddings. However, since this is not trivially implementable with 1-N scoring and wanting to keep its benefits, we leave this to future work.

4.4 Number of Parameters

Table 2 compares the number of parameters of ConvE and HypER (for the FB15k-237 dataset, which determines n_e and n_r). It can be seen that, overall, HypER has fewer parameters (4.3M) than ConvE (5.1M) due to the way HypER directly transforms relations to convolutional filters.

Table 2. Comparison of number of parameters for ConvE and HypER on FB15k-237. h_m and w_m are height and width of the ConvE feature maps respectively.

Model	**E**	**R**	Filters	**W**
ConvE	$n_e \times d_e$	$n_r \times d_r$	$l_f n_f$	$h_m w_m n_f \times d_e$
	$2.9M$	$0.1M$	$0.0M$	$2.1M$
HypER	$n_e \times d_e$	$n_r \times d_r$	$d_r \times l_f n_f$	$l_m n_f \times d_e$
	$2.9M$	$0.1M$	$0.1M$	$1.2M$

5 Experiments

5.1 Datasets

We evaluate our HypER model on the standard link prediction task using the following datasets (see Table 3):

FB15k [1] a subset of Freebase, a large database of facts about the real world.

WN18 [1] a subset of WordNet, containing lexical relations between words.

FB15k-237 created by [21], noting that the validation and test sets of FB15k and WN18 contain the inverse of many relations present in the training set, making it easy for simple models to do well. FB15k-237 is a subset of FB15k with the inverse relations removed.

WN18RR [3] a subset of WN18, created by removing the inverse relations.

YAGO3-10 [3] a subset of YAGO3 [10], containing entities which have a minimum of 10 relations each.

Table 3. Summary of dataset statistics.

Dataset	Entities (n_e)	Relations (n_r)
FB15k	14,951	1,345
WN18	40,943	18
FB15k-237	14,541	237
WN18RR	40,943	11
YAGO3-10	123,182	37

5.2 Experimental Setup

We implement HypER in PyTorch [13] and make our code publicly available.[1]

Implementation Details We train our model with 200 dimension entity and relation embeddings ($d_e = d_r = 200$) and 1-N scoring. Whilst the relation embedding dimension does not have to equal the entity embedding dimension, we set $d_r = 200$ to match ConvE for fairness of comparison.

To accelerate training and prevent overfitting, we use batch normalization [6] and dropout [18] on the input embeddings, feature maps and the hidden layer. We perform a hyperparameter search and select the best performing model by mean reciprocal rank (MRR) on the validation set. Having tested the values $\{0., 0.1, 0.2, 0.3\}$, we find that the following combination of parameters works

[1] `https://github.com/ibalazevic/HypER`

well across all datasets: input dropout 0.2, feature map dropout 0.2, and hidden dropout 0.3, apart from FB15k-237, where we set input dropout to 0.3. We select the learning rate from $\{0.01, 0.005, 0.003, 0.001, 0.0005, 0.0001\}$ and exponential learning rate decay from $\{1., 0.99, 0.995\}$ for each dataset and find the best performing learning rate and learning rate decay to be dataset-specific. We set the convolution stride to 1, number of feature maps to 32 with the filter size 3×3 for ConvE and 1×9 for HypER, after testing different numbers of feature maps $n_f \in \{16, 32, 64\}$ and filter sizes $l_f \in \{1 \times 1, 1 \times 2, 1 \times 3, 1 \times 6, 1 \times 9, 1 \times 12\}$ (see Table 9). We train all models using the Adam optimizer with batch size 128. One epoch on FB15k-237 takes approximately 12 seconds on a single GPU compared to 1 minute for e.g. RESCAL, largely due to 1-N scoring.

Evaluation Results are obtained by iterating over all triples in the test set. A particular triple is evaluated by replacing the object entity e_2 with all entities $\mathcal{E}$ while keeping the subject entity e_1 fixed and vice versa, obtaining scores for each combination. These scores are then ranked using the "filtered" setting only, i.e. we remove all true cases other than the current test triple [1].

We evaluate HypER on five different metrics found throughout the link prediction literature: mean rank (MR), mean reciprocal rank (MRR), hits@10, hits@3, and hits@1. Mean rank is the average rank assigned to the true triple, over all test triples. Mean reciprocal rank takes the average of the reciprocal rank assigned to the true triple. Hits@k measures the percentage of cases in which the true triple appears in the top k ranked triples. Overall, the aim is to achieve high mean reciprocal rank and hits@k and low mean rank. For a more extensive description of how each of these metrics is calculated, we refer to [3].

5.3 Results

Link prediction results for all models across the five datasets are shown in Tables 4, 5 and 6. Our key findings are:

- whilst having fewer parameters than the closest comparator ConvE, HypER consistently outperforms all other models across all datasets, thereby achieving state-of-the-art results on the link prediction task; and
- our filter dimension study suggests that no benefit is gained by convolving over reshaped 2D entity embeddings in comparison with 1D entity embedding vectors and that most information can be extracted with very small convolutional filters (Table 9).

Overall, HypER outperforms all other models on all metrics apart from mean reciprocal rank on WN18 and mean rank on WN18RR, FB15k-237, WN18, and YAGO3-10. Given that mean rank is known to be highly sensitive to outliers [11], this suggests that HypER correctly ranks many true triples in the top 10, but makes larger ranking errors elsewhere.

Given that most models in the literature, with the exception of ConvE, were trained with 100 dimension embeddings and 1-1 scoring, we reimplement previous models (DistMult, ComplEx and ConvE) with 200 dimension embeddings

and 1-N scoring for fair comparison and report the obtained results on WN18RR in Table 7. We perform the same hyperparameter search for every model and present the mean and standard deviation of each result across five runs (different random seeds). This improves most previously published results, except for ConvE where we fail to replicate some values. Notwithstanding, HypER remains the best performing model overall despite better tuning of the competitors.

Table 4. Link prediction results on WN18RR and FB15k-237. The RotatE [19] results are reported without their self-adversarial negative sampling (see Appendix H in the original paper) for fair comparison, given that it is not specific to that model only.

	WN18RR					FB15k-237				
	MR	MRR	H@10	H@3	H@1	MR	MRR	H@10	H@3	H@1
DistMult [23]	5110	.430	.490	.440	.390	254	.241	.419	.263	.155
ComplEx [22]	5261	.440	.510	.460	.410	339	.247	.428	.275	.158
Neural LP [24]	–	–	–	–	–	–	.250	.408	–	–
R-GCN [15]	–	–	–	–	–	–	.248	.417	.264	.151
MINERVA [2]	–	–	–	–	–	–	–	.456	–	–
ConvE [3]	**4187**	.430	.520	.440	.400	244	.325	.501	.356	.237
M-Walk [16]	–	.437	–	.445	.414	–	–	–	–	–
RotatE [19]	–	–	–	–	–	**185**	.297	.480	.328	.205
HypER (ours)	5798	**.465**	**.522**	**.477**	**.436**	250	**.341**	**.520**	**.376**	**.252**

Table 5. Link prediction results on WN18 and FB15k.

	WN18					FB15k				
	MR	MRR	H@10	H@3	H@1	MR	MRR	H@10	H@3	H@1
TransE [1]	**251**	–	.892	–	–	125	–	.471	–	–
DistMult [23]	902	.822	.936	.914	.728	97	.654	.824	.733	.546
ComplEx [22]	–	.941	.947	.936	.936	–	.692	.840	.759	.599
ANALOGY [9]	–	.942	.947	.944	.939	–	.725	.854	.785	.646
Neural LP [24]	–	.940	.945	–	–	–	.760	.837	–	–
R-GCN [15]	–	.819	**.964**	.929	.697	–	.696	.842	.760	.601
TorusE [4]	–	.947	.954	.950	.943	–	.733	.832	.771	.674
ConvE [3]	374	.943	.956	.946	.935	51	.657	.831	.723	.558
SimplE [7]	–	.942	.947	.944	.939	–	.727	.838	.773	.660
HypER (ours)	431	**.951**	.958	**.955**	**.947**	**44**	**.790**	**.885**	**.829**	**.734**

Table 6. Link prediction results on YAGO3-10.

	YAGO3-10				
	MR	MRR	H@10	H@3	H@1
DistMult [23]	5926	.340	.540	.380	.240
ComplEx [22]	6351	.360	.550	.400	.260
ConvE [3]	**1676**	.440	.620	.490	.350
HypER (ours)	2529	**.533**	**.678**	**.580**	**.455**

To ensure that the difference between reported results for HypER and ConvE is not simply due to HypER having a reduced number of parameters (implicit regularization), we trained ConvE reducing the number of feature maps to 16 instead of 32 to have a comparable number of parameters to HypER (explicit

Table 7. Link prediction results on WN18RR; all models trained with 200 dimension embeddings and 1-N scoring.

	WN18RR				
	MR	MRR	H@10	H@3	H@1
DistMult [23]	**4911 ± 109**	.434 ± .002	.508 ± .002	.447 ± .001	.399 ± .002
ComplEx [22]	5930 ± 125	.446 ± .001	**.523 ± .002**	.462 ± .001	.409 ± .001
ConvE [3]	**4997± 99**	.431 ± .001	.504 ± .002	.443 ± .002	.396 ± .001
HypER (ours)	5798 ± 124	**.465 ± .002**	**.522 ± .003**	**.477 ± .002**	**.436 ± .003**

regularization). This showed no improvement in ConvE results, indicating HypER's architecture does more than merely reducing the number of parameters.

Table 8. Results with and without hypernetwork on WN18RR and FB15k-237.

	WN18RR		FB15k-237	
	MRR	H@10	MRR	H@10
HypER	**.465 ± .002**	**.522 ± .003**	**.341 ± .001**	**.520 ± .002**
HypER (no **H**)	.459 ± .002	.511 ± .002	.338 ± .001	.515 ± .001

Hypernetwork Influence To test the influence of the hypernetwork and, thereby, knowledge sharing between relations, we compare HypER results on WN18RR and FB15k-237 with the hypernetwork component removed, i.e. without the first fully connected layer and with the relation embeddings directly corresponding to a set of convolutional filters. Results presented in Table 8 show that the hypernetwork component improves performance, demonstrating the value of multi-task learning across different relations.

Filter Dimension Study Table 9 shows results of our study investigating the influence of different convolutional filter sizes on the performance of HypER. The lower part of the table shows results for 2D filters convolved over reshaped (10×20) 2D subject entity embeddings. It can be seen that reshaping the embeddings is of no benefit, especially on WN18RR. These results indicate that the purpose of convolution on word embeddings is not to find patterns in a 2D embedding (as with images), but perhaps to limit the number of dimensions that can interact with each other, thereby avoiding overfitting. In the upper part of the table, we vary the length of 1D filters, showing that comparable results can be achieved with filter sizes 1×6 and 1×9, with diminishing results for smaller (e.g. 1×1) and larger (e.g. 1×12) filters.

Label Smoothing Contrary to the ablation study of [3], showing the influence of hyperparameters on mean reciprocal rank for FB15k-237, from which they deem label smoothing unimportant, we find label smoothing to give a significant improvement in prediction scores for WN18RR. However, we find it does have a negative influence on the FB15k scores and as such, exclude label smoothing from our experiments on that dataset. We therefore recommend evaluating the influence of label smoothing on a per dataset basis and leave to future work analysis of the utility of label smoothing in the general case.

Table 9. Influence of different filter dimension choices on prediction results.

	WN18RR		FB15k-237	
Filter Size	MRR	H@1	MRR	H@1
1×1	.455	.422	.337	.248
1×2	.458	.428	.337	.248
1×3	.457	.427	.339	.250
1×6	.459	.429	.340	.251
1×9	**.465**	**.436**	**.341**	**.252**
1×12	.457	.428	**.341**	**.252**
2×2	.456	.429	.340	.250
3×3	.458	.430	.339	.250
5×5	.452	.423	.340	**.252**

6 Conclusion

In this work, we introduce HypER, a hypernetwork model for link prediction on knowledge graphs. HypER generates relation-specific convolutional filters and applies them to subject entity embeddings. The hypernetwork component allows information to be shared between relation vectors, enabling multi-task learning across relations. To our knowledge, HypER is the first link prediction model that creates non-linear interaction between entity and relation embeddings by convolving relation-specific filters over the entity embeddings.

We show that no benefit is gained from 2D convolutional filters over 1D, dispelling the suggestion that 2D structure exists in entity embeddings implied by ConvE. We also recast HypER in terms of tensor operations showing that, despite the convolution operation, it is closely related to the established family of tensor factorization models. Our results suggest that convolution provides a good trade-off between expressiveness and parameter number compared to a dense network. HypER is fast, robust to overfitting, has relatively few parameters, and achieves state-of-the-art results across almost all metrics on multiple link prediction datasets.

Future work might include expanding the current architecture by applying convolutional filters to both subject and object entity embeddings. We may also analyze the influence of label smoothing and explore the interpretability of convolutional feature maps to gain insight and potentially improve the model.

Acknowledgements

We thank Ivan Titov for helpful discussions on this work. Ivana Balažević and Carl Allen were supported by the Centre for Doctoral Training in Data Science, funded by EPSRC (grant EP/L016427/1) and the University of Edinburgh.

References

1. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating Embeddings for Modeling Multi-relational Data. In: Advances in Neural Information Processing Systems (2013)

2. Das, R., Dhuliawala, S., Zaheer, M., Vilnis, L., Durugkar, I., Krishnamurthy, A., Smola, A., McCallum, A.: Go for a Walk and Arrive at the Answer: Reasoning over Paths in Knowledge Bases Using Reinforcement Learning. In: International Conference on Learning Representations (2018)
3. Dettmers, T., Minervini, P., Stenetorp, P., Riedel, S.: Convolutional 2D Knowledge Graph Embeddings. In: Association for the Advancement of Artificial Intelligence (2018)
4. Ebisu, T., Ichise, R.: TorusE: Knowledge Graph Embedding on a Lie Group. In: Association for the Advancement of Artificial Intelligence (2018)
5. Ha, D., Dai, A., Le, Q.V.: Hypernetworks. In: International Conference on Learning Representations (2017)
6. Ioffe, S., Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: International Conference on Machine Learning (2015)
7. Kazemi, S.M., Poole, D.: SimplE Embedding for Link Prediction in Knowledge Graphs. In: Advances in Neural Information Processing Systems (2018)
8. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. In: International Conference on Learning Representations (2015)
9. Liu, H., Wu, Y., Yang, Y.: Analogical Inference for Multi-relational Embeddings. In: International Conference on Machine Learning (2017)
10. Mahdisoltani, F., Biega, J., Suchanek, F.M.: Yago3: A Knowledge Base from Multilingual Wikipedias. In: Conference on Innovative Data Systems Research (2013)
11. Nickel, M., Rosasco, L., Poggio, T.A.: Holographic Embeddings of Knowledge Graphs. In: Association for the Advancement of Artificial Intelligence (2016)
12. Nickel, M., Tresp, V., Kriegel, H.P.: A Three-Way Model for Collective Learning on Multi-Relational Data. In: International Conference on Machine Learning (2011)
13. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic Differentiation in PyTorch. In: NIPS-W (2017)
14. Pereyra, G., Tucker, G., Chorowski, J., Kaiser, Ł., Hinton, G.: Regularizing neural networks by penalizing confident output distributions. arXiv preprint arXiv:1701.06548 (2017)
15. Schlichtkrull, M., Kipf, T.N., Bloem, P., van den Berg, R., Titov, I., Welling, M.: Modeling Relational Data with Graph Convolutional Networks. In: European Semantic Web Conference (2018)
16. Shen, Y., Chen, J., Huang, P.S., Guo, Y., Gao, J.: M-Walk: Learning to Walk over Graphs using Monte Carlo Tree Search. In: Advances in Neural Information Processing Systems (2018)
17. Socher, R., Chen, D., Manning, C.D., Ng, A.: Reasoning with Neural Tensor Networks for Knowledge Base Completion. In: Advances in Neural Information Processing Systems (2013)
18. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research **15**(1), 1929–1958 (2014)
19. Sun, Z., Deng, Z.H., Nie, J.Y., Tang, J.: RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In: International Conference on Learning Representations (2019)
20. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the Inception Architecture for Computer Vision. In: Computer Vision and Pattern Recognition (2016)

21. Toutanova, K., Chen, D., Pantel, P., Poon, H., Choudhury, P., Gamon, M.: Representing Text for Joint Embedding of Text and Knowledge Bases. In: Empirical Methods in Natural Language Processing (2015)
22. Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., Bouchard, G.: Complex Embeddings for Simple Link Prediction. In: International Conference on Machine Learning (2016)
23. Yang, B., Yih, W.t., He, X., Gao, J., Deng, L.: Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In: International Conference on Learning Representations (2015)
24. Yang, F., Yang, Z., Cohen, W.W.: Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In: Advances in Neural Information Processing Systems (2017)