Abstract

Knowledge graphs are incomplete by nature, with only a limited number of observed facts from the world knowledge being represented as structured relations between entities. To partly address this issue, an important task in statistical relational learning is that of link prediction or knowledge graph completion. Both linear and non-linear models have been proposed to solve the problem. Bilinear models, while expressive, are prone to overfitting and lead to quadratic growth of parameters in number of relations. Simpler models have become more standard, with certain constraints on bilinear map as relation parameters. In this work, we propose a factorized bilinear pooling model, commonly used in multi-modal learning, for better fusion of entities and relations, leading to an efficient and constraint-free model. We prove that our model is fully expressive, providing bounds on the embedding dimensionality and factorization rank. Our model naturally generalizes Tucker decomposition based TuckER model, which has been shown to generalize other models, as efficient low-rank approximation without substantially compromising the performance. Due to low-rank approximation, the model complexity can be controlled by the factorization rank, avoiding the possible cubic growth of TuckER. Empirically, we evaluate on real-world datasets, reaching on par or state-of-the-art performance. At extreme low-ranks, model preserves the performance while staying parameter efficient.

1. Introduction

Knowledge graphs (KGs) are large collections of structured knowledge, organized as subject and object entities and relations, in the form of fact triples \(<\text{sub}, \text{rel}, \text{obj}>\). The usefulness of knowledge graphs, however, is affected primarily by their incompleteness. The task of link prediction or knowledge graph completion (KGC) aims to infer missing facts from existing ones, by essentially scoring a relation and entities triple for use in predicting its validity, and thereby avoiding the cost and time of extending knowledge graphs manually. To accomplish this, several models have been proposed, including linear and non-linear models. Bilinear models have additionally been used in multi-modal learning due to their expressive nature, where the fusion of features from different modalities plays a key role towards the performance of a model, with concatenation or element-wise summation being commonly used fusion techniques. The underlying assumption is that the distributions of features across modalities may vary significantly, and the representation capacity of the fused features may be insufficient, therefore limiting the final prediction performance (Yu et al., 2017). In this work, we apply this assumption to knowledge graphs by considering that the entities and relations come from different multi-modal distributions and good fusion between them can potentially construct a KG.

A major drawback of using bilinear modeling methods is the quadratic growth of parameters, which results in high computational and memory costs and risks overfitting. In multi-modal learning, factorization techniques have therefore been researched to address these challenges (Kim et al., 2016; Fukui et al., 2016; Yu et al., 2017; Ben-Younes et al., 2017; Li et al., 2017; Liu et al., 2018), and constraints based bilinear maps have become a more prevalent standard in link prediction (Yang et al., 2015; Trouillon et al., 2016; Kazemi & Poole, 2018). Applying constraints can be seen as hard regularization since it allows for incorporating background knowledge (Kazemi & Poole, 2018), but restricts the learning potential of the model due to limited parameter sharing (Balažević et al., 2019a). We focus on a constraint-free approach, using the low-rank factorization of bilinear models, as it offers flexibility and generalizes well, naturally leading to other models under certain conditions. Our work extends the multi-modal factorized bilinear pooling (MFB) model, introduced by Yu et al. (2017), and applies it to the link prediction task.

Our contributions are outlined as follows:
LowFER: Low-rank Bilinear Pooling for Link Prediction

- We propose a simple and parameter efficient linear model by extending multi-modal factorized bilinear (MFB) pooling (Yu et al., 2017) for link prediction.
- We prove that our model is fully expressive and provide bounds on entity and relation embedding dimensions, along with the factorization rank.
- We provide relations to the family of bilinear link prediction models and Tucker decomposition (Tucker, 1966) based TuckER model (Balažević et al., 2019a), generalizing them as special cases. We also show the relation to 1D convolution based HypER model (Balažević et al., 2019b), bridging the gap from bilinear to convolutional link prediction models.
- On real-world datasets the model achieves on par or state-of-the-art performance, where at extreme low-ranks, with limited number of parameters, it outperforms most of the prior arts, including deep learning based models.

2. Related Work

Given a set of entities $E$ and relations $R$ in a knowledge graph $KG$, the task of link prediction is to assign a score $s$ to a triple $(e_s, r, e_o)$:

$$s = f(e_s, r, e_o)$$

where $e_s \in E$ is the subject entity, $e_o \in E$ is the object entity and $r \in R$ is the relation between them. The scoring function $f$ estimates the general binary tensor $T \in |E| \times |R| \times |E|$, by assigning a score of 1 to $T_{ijk}$ if relation $r_j$ exists between entities $e_i$ and $e_k$, 0 otherwise. The scoring function can be a linear or non-linear model, trained to predict true triples in a $KG$.

Deep learning based scoring functions such as ConvE (Dettmers et al., 2018) and HypER (Balažević et al., 2019b) use 2D and 1D convolution on subject entity and relation representations respectively. Both perform well in practice and are efficient, but the former lacks direct interpretation, whereas the latter has shown to be related to tensor factorization. Transitional methods (Bordes et al., 2013; Wang et al., 2014; Ji et al., 2015; Lin et al., 2015; Nguyen et al., 2016; Feng et al., 2016) use additive dissimilarity scoring functions, whereby they differ in terms of the constraints applied to the projection matrices. While interpretable, they are theoretically limited as they have shown to be not fully expressive (Wang et al., 2018; Kazemi & Poole, 2018). There are several other related works (Nickel et al., 2016; Das et al., 2017; Yang et al., 2017; Shen et al., 2018; Schlichtkrull et al., 2018; Ebisu & Ichise, 2018; Sun et al., 2019), but we will mainly focus on different types of linear models here, as they are more relevant to our work.

All discussed linear models can be seen as a decomposition of the tensor $T$, using different factorization methods. One way to factorize this tensor is to factorize its slices in the relation dimension with DEDICOMP (Harshman, 1978). RESCAL (Nickel et al., 2011), a relaxed version of DEDICOMP, decomposes using a scoring function that consists of a bilinear product between subject and object entity vectors with a relation specific matrix. RESCAL, however, tends to overfit due to the quadratic growth of parameters in number of relations. Others use Canonical Polyadic decomposition (CPD or simply CP) (Hitchcock, 1927; Harshman & Lundy, 1994) to factorize the binary tensor. In CP, each value in the tensor is obtained as a sum of multiple Hadamard products of three vectors, representing subject, object and relation.

3. Model

Downstream performance for tasks such as visual question answering strongly depends on the multi-modal fusion of features to leverage the heterogeneous data (Liu et al., 2018). Bilinear models are expressive as they allow for pairwise interactions between the feature dimensions but also introduce huge number of parameters that lead to high computational and memory costs and the risk of overfitting (Fukui et al., 2016). Substantial research has therefore focused on efficiently computing the bilinear product. In multi-modal compact bilinear (MCB) pooling (Gao et al., 2016; Fukui et al., 2016), authors employ a sampling based approximation that uses the property that the tensor sketch projection (Charikar et al., 2004; Pham & Pagh, 2013) of the outer product of two vectors can be represented as their sketches convolution. Multi-modal low-rank bilinear (MLB) pooling (Kim et al., 2016) uses two low-rank projection matrices to transform the features from the original space to a common space, followed by the Hadamard product, which was later generalized by the multi-modal factorized bilinear (MFB) pooling (Yu et al., 2017). Our work is based on the MFB model but can also be seen as related to Liu et al. (2018). In contrast to KGC bilinear models, these bilinear models allow for parameter sharing and generally, are constraint-free.
3.1. Multi-modal Factorized Bilinear Pooling (MFB)

Given two feature vectors \( x \in \mathbb{R}^m, y \in \mathbb{R}^n \) and a bilinear map \( W \in \mathbb{R}^{m \times n} \), the bilinear transformation is defined as \( z = x^T W y \). To obtain a vector in \( \mathbb{R}^p \), \( p \) such maps have to be learned (e.g. in RESCAL these would be relation specific matrices), resulting in large number of parameters. However, \( W \) can be factorized into two low-rank matrices:

\[
z = x^T U V^T y = 1^T(U^T x \circ V^T y)
\]

where \( U \in \mathbb{R}^{m \times k}, V \in \mathbb{R}^{n \times k} \), \( k \) is the factorization rank, \( \circ \) is the element-wise product of two vectors and \( 1 \in \mathbb{R}^k \) is vector of all ones. Therefore, to obtain a output feature vector \( z \in \mathbb{R}^o \), two 3D tensors are required, \( W_x = [U_1, U_2, \ldots, U_o] \) reshaped \( W_x \) and \( W_y = [V_1, V_2, \ldots, V_o] \) reshaped \( W_y \), where \( W_x \in \mathbb{R}^{m \times k \times o} \) and \( W_y \in \mathbb{R}^{n \times k \times o} \) are their reshaped 2D matrices respectively. The final (fused) vector \( z \) is then obtained by summing non-overlapping windows of size \( k \) over the Hadamard product of projected vectors using \( W_x \) and \( W_y \):

\[
z = \text{SumPool}(W_x^T x \circ W_y^T y, k)
\]

At \( k = 1 \), MFB reduces to MLB, which converges slowly, and MCB requires very high-dimensional vectors to perform well (Yu et al., 2017). Further, MFB significantly lowers the number of parameters with low-rank factorized matrices and leads to better performance.

3.2. Low-rank Bilinear Pooling for Link Prediction

Consider that entities and relations are not intrinsically bound and come from two different modalities, such that good fusion between them can potentially result in a knowledge graph of fact triples. Entities and relations can be shown to possess certain properties that allow them to function similarly to others within the same modality. For example, the relation place-of-birth shares inherent properties with the relation residence. As such, similar entity pairs can yield similar relations, given appropriate shared properties. Like in multi-modal auditory-visual fusion, where the sound of a roar may better predict a resulting image within the distribution of animals that roar, a relation such as place-of-birth, can better predict an entity pair within a distribution of person, place entity pairs. In link prediction, we assume that the latent decomposition with MFB can help the model capture different aspects of interactions between an entity and a relation, which can lead to better scoring with the missing entity. We therefore, apply the Low-rank Factorization trick of bilinear maps with \( k \)-sized non-overlapping summation pooling (section 3.1) to Entities and Relations (LowFER).

![Figure 1. Overview of the LowFER model.](image)

More formally, for an entity \( e \in \mathcal{E} \), we represent its embedding vector \( e_\mathcal{E} \) of dimension as a look-up from entity embedding matrix \( E \in \mathbb{R}^{n_e \times d_e} \) and relation vector \( r \in \mathbb{R}^{n_r} \) from relation embedding matrix \( R \in \mathbb{R}^{n_r \times d_r} \), where \( n_e \) and \( n_r \) are number of entities and relations in KG. LowFER projects \( e_\mathcal{E} \) and \( r \) into a common space \( \mathbb{R}^{kd_e} \) followed by Hadamard product and \( k \)-summation pooling, where \( k \) is the factorization rank. The output vector \( z \) is then matched against target entity \( e_\mathcal{E} \) to give final score.

Using this compact notation in Eq. 2, the final scoring function of LowFER is obtained as:

\[
f(e_s, r, e_o) = (S^k \text{diag}(U^T e_s) V^T r) \cdot e_o
\]

\[\text{diag}(U^T e_s) \in \mathbb{R}^{kd_e \times d_e} \text{ and } S^k \in \mathbb{R}^{d_e \times kd_e} \text{ is a constant matrix}\]

\[^1\text{Note that at } k = 1, S^1 = \mathbb{I}_{d_e}\]
Table 1. Bounds for fully expressive linear models, where \(n\) is the number of true facts and the trivial bound is given by \(n_s^2n_r\).

<table>
<thead>
<tr>
<th>Model</th>
<th>Full Expressibility Bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESCAL (Nickel et al., 2011)</td>
<td>((d_e, d_r) = (n_s^2, n_r^2))</td>
</tr>
<tr>
<td>HolE (Nickel et al., 2016)</td>
<td>(d_e = d_r = 2n_s n_r + 1)</td>
</tr>
<tr>
<td>ComplEx (Trouillon et al., 2016)</td>
<td>(d_e = d_r = n_s n_r)</td>
</tr>
<tr>
<td>SimplE (Kazemi &amp; Poole, 2018)</td>
<td>(d_e = d_r = \min(n_s, n_r + 1))</td>
</tr>
<tr>
<td>TuckER (Baláževič et al., 2019a)</td>
<td>((d_e, d_r) = (n_s, n_r))</td>
</tr>
<tr>
<td>LowFER</td>
<td>((d_e, d_r) = (n_s, n_r)) for (k = \min(n_s, n_r))</td>
</tr>
</tbody>
</table>

3.3. Training LowFER

To train the LowFER model, we follow the setup of Baláževič et al. (2019a). First, we apply sigmoid non-linearity after Eq. 5 to get the probability \(p(y(e, r, e_o)) = \sigma(f(e, r, e_o))\) of a triple belonging to a \(KG\). Then, for every triple \((e_s, r, e_o)\) in the dataset, a reciprocal relation is added by generating a synthetic example \((e_o, r^{-1}, e_s)\) (Dettmers et al., 2018; Lacroix et al., 2018) to create the training set \(D\). For faster training, Dettmers et al. (2018) introduced 1-N scoring, where each tuple \((e_s, r)\) and \((e_o, r^{-1})\) is simultaneously scored against all entities \(e \in E\) to predict 1 if \(e = e_o\) or \(e_s\) respectively and 0 elsewhere (see Trouillon et al. (2017) and Sun et al. (2019) for other methods to collect negative samples). The model is trained with binary cross-entropy instead of margin based ranking loss (Bordes et al., 2019), which is prone to overfitting for link prediction (Trouillon & Nickel, 2017; Kazemi & Poole, 2018). For a mini-batch \(B\) of size \(m\) drawn from \(D\), we minimize:

\[
\min_{\Theta} \frac{1}{m} \sum_{(e, r) \in B} \frac{1}{n_e} \sum_{i=1}^{n_e} \left( y_i \log(p(y(e, r, e_i))) \\
+ (1 - y_i) \log(1 - p(y(e, r, e_i))) \right)
\]

where \(y_i\) is a target label for a given entity-relation pair \((e, r)\) for entity \(e_i\), \(y_{e, r, e_i}\) is the model prediction and \(\Theta\) represents model parameters. Following Yu et al. (2017), we also apply the power normalization \(x \leftarrow \text{sign}(x)|x|^{0.5}\) and \(l_2\)-normalization \(x \leftarrow x/||x||\) before summation pooling to stabilize the training from large output values as a result of Hadamard product in Eq. 3.

4. Theoretical Analysis

4.1. Full Expressibility

A key theoretical property of link prediction models is their ability to be fully expressive, which we define formally as:

**Definition 1.** Given a set of entities \(E\), relations \(R\), correct triples \(T \subseteq E \times R \times E\) and incorrect triples \(T' = E \times R \times E \setminus T\), then a model \(M\) with scoring function \(f(e_s, r, e_o)\) is said to be fully expressive if it can accurately separate \(T\) from \(T'\) for all \(e_s, e_o \in E\) and \(r \in R\).

A fully expressive model can represent relations of any type, including symmetric, asymmetric, reflexive, and transitive among others. Models such as RESCAL, HolE, ComplEx, SimplE and TuckER have been shown to be fully expressive (Trouillon & Nickel, 2017; Wang et al., 2018; Kazemi & Poole, 2018; Baláževič et al., 2019a). On the other hand, DistMult is not fully expressive as it enforces symmetric relations only. Further, Wang et al. (2018) showed that TransE is not fully expressive, which was later expanded by Kazemi & Poole (2018), showing that other translational variants, including, FTransE, STTransE, FSTransE, TransR and TransH are likewise not fully expressive. By the virtue of universal approximation theorem (Cybenko, 1989; Hornik, 1991), neural networks can be considered fully expressive (Kazemi & Poole, 2018). Table 1 summarizes the bounds of linear models that are fully expressive. With Proposition 1 (proof in Appendix A.1), we establish that LowFER is fully expressive and provide bounds on entity and relation embedding dimensions and the factorization rank \(k\).

**Proposition 1.** For a set of entities \(E\) and a set of relations \(R\), given any ground truth \(T\), there exists an assignment of values in the LowFER model with entity embeddings of dimension \(d_e = |E|\), relation embeddings of dimension \(d_r = |R|\) and the factorization rank \(k = \min(d_e, d_r)\) that makes it fully expressive.

As a given example, consider a set of entities \(E = \{e_1, e_2, e_3, e_4\}\) and relations \(R = \{r_1, r_2, r_3, r_4\}\) such that \(r_1\) is reflexive, \(r_2\) is symmetric, \(r_3\) is asymmetric, and \(r_4\) is transitive, then for ground truth \(T = \{(e_1, r_1, e_1), (e_1, r_2, e_2), (e_2, r_3, e_1), (e_3, r_3, e_2), (e_4, r_4, e_4), (e_4, r_4, e_3)\}\) and following the settings in Proposition 1, Figure 2 shows the model parameters \(U\) and \(V\) for this toy example. Now, consider the case \(k = d_e = n_e\), then \(U\) copies each entity vector in \(k\)-sized slices and \(V\) buckets target entities per relation such that each source entity is distributed into disjoint sets. Note that reshaping \(V\) as 3D tensor of size \(n_e \times n_e \times n_e\) and transposing first two dimensions results in binary tensor \(T\).
4.2. Relation with TuckER

Initially, it was shown by Kazemi & Poole (2018) that RESCAL, DistMult, ComplEx and SimplE belong to a family of bilinear models with different set of constraints. Later, Balazević et al. (2019a) established that TuckER generalizes all of these models as special cases. In this section, we will formulate relation between our model and TuckER (Balazević et al., 2019a), followed by relations with the family of bilinear models in the next section. This provides a unifying view and shows LowFER’s ability to generalize.

TuckER’s scoring function is defined as follows (Balazević et al., 2019a):

\[ \phi_t(e_s, r, e_o) = W \times_1 e_s \times_2 r \times_3 e_o \]  

(6)

where \( W \in \mathbb{R}^{d_e \times d_r \times d_e} \) is the core tensor, \( e_s, e_o \in \mathbb{R}^{d_e} \) and \( r \in \mathbb{R}^{d_r} \) are subject entity, object entity and the relation vectors respectively. \( \times_n \) denotes the tensor product along the n-th mode. First, note that Eq. 4 can be expanded as:

\[
S^k(U^T e_s \circ V^T r) = \begin{bmatrix}
    e_s^T (\sum_{i=1}^k u_i \odot v_i) r \\
    \vdots \\
    e_s^T (\sum_{i=(j-1)k+1}^{jk} u_i \odot v_i) r \\
    \vdots \\
    e_s^T (\sum_{i=kd_e}^{kd_e} u_i \odot v_i) r
\end{bmatrix}
\]

where \( u_i \in \mathbb{R}^{d_e} \) and \( v_i \in \mathbb{R}^{d_r} \) are column vectors of \( U \) and \( V \) respectively and \( \odot \) represents the outer product of two vectors. To take the vectors \( e_s \) and \( r \) out, we realize the above matrix operations in a different way. We first create \( k \) distance apart matrices sliced from \( U \) (and \( V \), i.e., for the \( l \)-th slice, we have \( W_{U}^{(l)} = [u_l, u_{l+k}, ..., u_{l(kd_e-1)+l}] \in \mathbb{R}^{d_e \times d_e} \) and \( W_{V}^{(l)} = [v_l, v_{l+k}, ..., v_{l(kd_e-1)+l}] \in \mathbb{R}^{d_r \times d_e} \). Taking the column-wise outer product of these sliced matrices forms a 3D tensor in \( \mathbb{R}^{d_e \times d_r \times d_e} \). With slight abuse of notation, we also use \( \otimes \) to represent this tensor operation. It can be viewed as transforming the matrix obtained by mode-2 Khatri-Rao product into a 3D tensor (Cichocki et al., 2016). Now consider a 3D tensor \( W \in \mathbb{R}^{d_e \times d_r \times d_e} \) as the sum of these \( k \) products:

\[ W = \sum_{i=1}^k W_U^{(i)} \otimes W_V^{(i)} \]  

(7)

Figure 3 shows these operations. With this tensor, the scoring function \( f \) in Eq. 5 can be re-written as TuckER’s scoring function as follows:

\[ \tilde{\phi}_t(e_s, r, e_o) = W \times_1 e_s \times_2 r \times_3 e_o \]  

(8)

It should be noted that \( W \) in Eq. 8 is obtained as a summation of \( k \) low-rank 3D tensors, each of which is obtained by stacking rank-1 matrices in contrast to TuckER’s core tensor \( W \) in Eq. 6, which can be a full rank 3D tensor. Our model can therefore approximate TuckER and can be viewed as a generalization of TuckER (Balazević et al., 2019a). We further show that we can accurately obtain \( W \) with appropriate \( W_U^{(i)} \)’s and \( W_V^{(i)} \)’s in Eq. 7 (proof in Appendix A.2).

**Proposition 2.** Given a TuckER model with entity embedding dimension \( d_e \), relation embedding dimension \( d_r \) and core tensor \( W \), there exists a LowFER model with \( k <= \min(d_e, d_r) \), entity embedding dimension \( d_e \) and relation embedding dimension \( d_r \) that accurately represents the former.

LowFER and TuckER parameters grow linearly in the number of entities and relations as \( O(n_e d_e + n_r d_r) \). However, LowFER’s shared parameters complexity can be controlled by decoupled low-rank matrices through the factorization rank, making it more flexible, e.g., consider \( d = d_e = d_r \), the core tensor \( W \) of TuckER grows as \( O(d^3) \), whereas LowFER grows only as \( O(kd^2) \). As an example, in Lacroix...
et al. (2018) authors used \( d_c = d_r = 2000 \) which would require more than 8 billion parameters to model with TuckER compared to only 4k million for LowFER, with \( k \) controlling the growth. More generally, at \( k = d_c / 2 \), LowFER has equal number of parameters as TuckER therefore, we expect similar performance at such rank values. In practice, \( k = \{1, 10, 30\} \) performs extremely well (section 5.1).

### 4.3. Relations with the Family of Bilinear Models

In this section, we will establish relations between LowFER and other bilinear models. For simplicity, we consider the relation embedding to be a constant matrix \( R = I_n \), in all the cases and use \( V \) to model relation parameters. However, the conditions presented here can be extended otherwise, with the remark that they are not unique.

**RESCAL** (Nickel et al., 2011): scoring function is defined as:

\[
\phi_r(e_s, r_l, e_o) = e_o^T W_r e_o
\]

where \( W_r \in \mathbb{R}^{d_r \times d_r} \) is \( l \)-th relation matrix. For LowFER to encode RESCAL with Eq. 5, we set \( k = d_c, d_r = n_r \) and \( U = [I_{d_c} | I_{d_c} | \ldots | I_{d_c}] \in \mathbb{R}^{d_c \times d_r^2} \) (block matrix partitioned as \( d_c \) identity matrices of size \( d_c \times d_r \)). This is effectively taking a row \( l \) from \( V \in \mathbb{R}^{n_r \times d_r^2} \), reshaping it to \( d_c \times d_c \) matrix and then taking the transpose to get the equivalent \( W_r \) in RESCAL’s scoring function.

**DistMult** (Yang et al., 2015): scoring function is defined as:

\[
\phi_d(e_s, r_l, e_o) = e_o^T \text{diag}(w_l) e_o
\]

where \( w_l \in \mathbb{R}^{d_r} \) is the vector for \( l \)-th relation. For LowFER to encode DistMult with Eq. 5, we set \( k = 1, d_r = n_r \) and \( U = I_{d_c} \). This is effectively taking a row \( l \) from \( V \in \mathbb{R}^{n_r \times d_r} \) and creating a diagonal matrix of it to get the equivalent \( \text{diag}(w_l) \) in DistMult’s scoring function.

**Simple** (Kazemi & Poole, 2018): scoring function is defined as:

\[
\phi_s(e_s, r_l, e_o) = \frac{1}{2}(h_{e_s}^T \text{diag}(r_l) t_{e_o} + h_{e_o}^T \text{diag}(r_l^{-1}) t_{e_s})
\]

where \( h_{e_s}, h_{e_o} \in \mathbb{R}^d \) are subject, object entities head vectors, \( t_{e_s}, t_{e_o} \in \mathbb{R}^d \) are subject, object entities tail vectors and \( r_l, r_l^{-1} \in \mathbb{R}^d \) are relation and inverse relation vectors. Let \( e_s = \{e_s, h_{e_s}\} \in \mathbb{R}^{2d}, e_o = \{h_{e_o}, t_{e_o}\} \in \mathbb{R}^{2d} \) and \( r_l = [r_l^{-1}; r_l] \in \mathbb{R}^{2d} \) then Simple scoring is equivalent to \( \frac{1}{2}e_s^T \text{diag}(r_l) e_o \), where \( e_s \) and \( r_l \) are obtained by swapping the head, tail vectors in \( e_s = [h_{e_s}; t_{e_s}] \) and relation, inverse relation vectors in \( r_l = [r_l^{-1}; r_l^{-1}] \) respectively. For LowFER to encode Simple, \( U \) becomes a permutation matrix (ignoring the \( \frac{1}{2} \) scaling factor), swapping the first \( d \)-half with the second \( d \)-half of a given vector in \( \mathbb{R}^{2d} \) and \( l \)-th row in \( V \) is \( r_l \), more specifically, with Eq. 5, we set \( k = 1, d_c = 2d, d_r = n_r \) and \( U = \mathbb{R}^{2d \times 2d} \) is a block matrix with four partitions such that, \( U_{12} = U_{21} = \frac{1}{2}I_d \) and 0s elsewhere.

**COMPLEX** (Trouillon et al., 2016) scoring function is defined as:

\[
\phi_c(e_s, r_l, e_o) = \text{Re}(e_s)^T \text{diag}(\text{Re}(r_l)) \text{Re}(e_o) + \text{Im}(e_s)^T \text{diag}(\text{Im}(r_l)) \text{Im}(e_o)
\]

\[
+ \text{Re}(e_s)^T \text{diag}(\text{Im}(r_l)) \text{Im}(e_o) - \text{Im}(e_s)^T \text{diag}(\text{Re}(r_l)) \text{Re}(e_o)
\]

where \( \text{Re}(\cdot) \) and \( \text{Im}(\cdot) \) represents the real and imaginary parts of a complex vector. Consider \( e_s = [\text{Re}(e_s); \text{Im}(e_s)] \in \mathbb{R}^{2d} \) and \( e_o = [\text{Re}(e_o); \text{Im}(e_o)] \in \mathbb{R}^{2d} \) then the ComplEx scoring function can be obtained as \( e_s^T W_r e_o \), where \( W_r \in \mathbb{R}^{2d \times 2d} \) represents the \( l \)-th relation matrix such that its diagonal is \( [\text{Re}(r_l); \text{Re}(r_l)] \), the \( d \) offset diagonal is \( \text{Im}(r_l) \) and \( -d \) offset diagonal is \( -\text{Im}(r_l) \). For LowFER to encode ComplEx, similar to Simple, we will use two permutation matrices to obtain the above four terms. That is, in Eq. 8, we have \( k = 2, d_c = 2d, d_r = n_r \) and \( U \in \mathbb{R}^{2d \times 4d} \) is such that \( W_r^{(1)} \) is a block matrix with \( W_r^{(1)}_{U_{11}} = W_r^{(1)}_{U_{12}} = I_d \) and 0 elsewhere. Further, \( W_r^{(2)} \) is also a block matrix with \( W_r^{(2)}_{U_{21}} = -I_d, W_r^{(2)}_{U_{22}} = I_d \) and 0 elsewhere. Lastly, \( V \in \mathbb{R}^{n_r \times 4d} \) is such that \( W_r^{(1)} \) row \( l \) has \( [\text{Re}(r_l); \text{Im}(r_l)] \) and \( W_r^{(2)} \) row \( l \) has \( [\text{Im}(r_l); \text{Re}(r_l)] \), i.e.,
where we show that LowFER encodes HypER, bringing it closer.

\( \mathbf{W}^{(2)} = \mathbf{W}^{(1)} \mathbf{P} \), where \( \mathbf{P} \in \mathbb{R}^{2d \times 2d} \) is the \( d \)-half swapping permutation matrix. Figure 4 demonstrates how LowFER parameters for the family of bilinear models under the conditions discussed in this section.

4.4. Relation to HypER

HypER (Balažević et al., 2019b) is a convolutional model based on hypernetworks (Ha et al., 2017), where the relation specific 1D filters are generated by the hypernetwork and convolved with the subject entity vector. Balažević et al. (2019b) showed that it can be understood in terms of tensor factorization up to a non-linearity. With a similar argument, we show that LowFER encodes HypER, bringing it closer to the convolutional approaches as well.

HypER scoring function is defined as (Balažević et al., 2019b):

\[
\phi_h(e_s, r, e_o) = h(\text{vec}(e_s \ast F_r) \mathbf{W})e_o \tag{9}
\]

where \( F_r = \text{vec}^{-1}(H_r) \in \mathbb{R}^{n_f \times 1}, H \in \mathbb{R}^{n_f \times d_e} \) (hypernetwork), \( \mathbf{W} \in \mathbb{R}^{n_f \times d_e \times d_e} \), vec(·) transforms \( n \times m \) matrix to \( nm \)-sized vector, vec\(^{-1}(·) \) does the reverse operation, \( \ast \) is the convolution operator, \( h(·) \) is a ReLU non-linearity and \( n_f, l_f \) and \( l_m = d_e - l_f + 1 \) are number of filters, filter length and output length of convolution. The convolution between a filter and the subject entity embedding can be seen as a matrix multiplication, where the filter is converted to a Toeplitz matrix of size \( l_m \times d_e \). With \( n_f \) filters, we can realize a 3D tensor of size \( n_f \times l_m \times d_e \). Since the filters are generated by the hypernetwork, we have \( d_e \) such 3D tensors, resulting in a 4D tensor of size \( n_f \times l_m \times d_e \times d_r \) (Balažević et al., 2019b).

Without loss of generality, we can view this 4D tensor as a 3D tensor \( \mathbf{F} \in \mathbb{R}^{n_f \times l_m \times d_e \times d_r} \). Taking mode-1 product as \( \mathbf{F} \times \mathbf{W}^T \) returns a final tensor \( \mathbf{G} \in \mathbb{R}^{d_e \times d_e \times d_e} \). Thus, HypER operations vec(\( e_s \ast F_r \)) \mathbf{W}

simplify to \( \mathbf{G} \times_3 \mathbf{r} \times_2 \mathbf{e}_s \). At \( k = d_e \), with \( \mathbf{U} \in \mathbb{R}^{d_e \times d_e} \) as block identity matrices (same as in LowFER’s relation to RESCAL) and \( \mathbf{V} \in \mathbb{R}^{d_e \times d_e} \) set to \( \mathbf{G}^T \) (\( \mathbf{G} \) viewed as a matrix of size \( d_e^2 \times d_e \) and transposed), LowFER’s score in Eq. 5 represents HypER, up to the non-linearity.

5. Experiments and Results

We conducted the experiments on four benchmark datasets: WN18 (Bordes et al., 2013), WN18RR (Dettmers et al., 2018), FB15k (Bordes et al., 2013) and FB15k-237 (Toutanova et al., 2015) (see Appendix B for the details, including best hyperparameters and additional experiments).

5.1. Link Prediction

Table 2 shows our main results, where LowFER-1, LowFER-10 and LowFER-\( k^* \) represent our model for \( k = 1, k = 10 \) and \( k = \text{best} \). We choose LowFER-1 and LowFER-10 as baselines. Overall, LowFER reaches competitive performance on all the datasets with state-of-the-art results on FB15k and FB15k-237. On WN18 and WN18RR, TuckER is marginally better than LowFER.

LowFER performs well at low-ranks with significantly less number of parameters compared to other linear models (Table 3). At \( k = 1 \), it performs better than or on par with both non-linear and linear models (including ComplEx and Simple) except HypER and TuckER. For FB15k-237, LowFER-1 (\( \sim 3M \) parameters) outperforms R-GCN, RotatE, DistMult and ComplEx by an average of 5.9% on MRR, and it additionally outperforms convolutional models (ConvE, HypER) at \( k = 10 \) with only +0.8M parameters. On FB15k, the best reported TuckER model is improved upon, with absolute +1.9% increase on toughest Hits@1 metric. This already achieves state-of-the-art with almost half the parameters, \( \sim 5.5M \) in contrast to TuckER’s \( \sim 11.3M \). On WN18RR and WN18, LowFER-1 outperforms all the models excluding TuckER and HypER. With LowFER-\( k^* \), we marginally reach state-of-the-art performance on WN18RR and FB15k-237. On FB15k, we reach new state-of-the-art for \( \sim 9.5M \) parameters with +2.9% and +4.1% improvement on MRR and Hits@1.

The empirical gains can be attributed to LowFER’s ability to perform good fusion between entities and relations while avoiding overfitting through low-rank matrices remaining parameter efficient, with strong performance even at ex-
Table 3. Comparison between the number of parameters in millions (M) of strong linear models. For LowFER-\(k\)-*, the \(k\) values are 10, 100, 30 and 50 for WN18, FB15k-237, WN18RR and FB15k respectively.

<table>
<thead>
<tr>
<th>Model</th>
<th>WN18</th>
<th>FB15k-237</th>
<th>WN18RR</th>
<th>FB15k</th>
</tr>
</thead>
<tbody>
<tr>
<td>ComplEx</td>
<td>16.4</td>
<td>6.0</td>
<td>16.4</td>
<td>6.5</td>
</tr>
<tr>
<td>SimplE</td>
<td>16.4</td>
<td>-</td>
<td>16.4</td>
<td>6.5</td>
</tr>
<tr>
<td>TuckER</td>
<td>9.4</td>
<td>11.0</td>
<td>9.4</td>
<td>11.3</td>
</tr>
<tr>
<td>LowFER-1</td>
<td>8.2</td>
<td>3.0</td>
<td>8.2</td>
<td>4.6</td>
</tr>
<tr>
<td>LowFER-10</td>
<td>8.6</td>
<td>3.8</td>
<td>8.6</td>
<td>5.5</td>
</tr>
<tr>
<td>LowFER-(k)*</td>
<td>8.6</td>
<td>11.3</td>
<td>9.6</td>
<td>9.5</td>
</tr>
</tbody>
</table>

Figure 5. Influence of increasing the factorization rank on MRR and Hits@1 scores for FB15k.

Empirically, we found when \(d_e = d_r\), taking \(k = d_e/2\) performs nearly the same as TuckER (Balažević et al., 2019a). This can be observed in LowFER-\(k\)* for FB15k-237 \((d_e = d_r = 200, k = 100)\), where our results are almost indistinguishable from TuckER’s. This can be expected as the number of parameters in both models are almost the same (~11M). It should be noted that in practice when we train LowFER, we initialize with two i.i.d matrices, which are not shared, compared to TuckER’s core tensor (Eq. 6), allowing us to reach almost the same performance despite less parameter sharing.

5.2. Effect of Factorization Rank

From link prediction results, we observe that rank plays an important role depending on the entities-to-relations ratio in the dataset. For \(d_e = 200\) and \(d_r = 30\), we vary \(k\) from \(\{1, 5, 10, 30, 50, 100, 150, 200\}\) on FB15k and plot the MRR and Hits@1 scores (Figure 5). From \(k = 1\) to \(k = 5\), the MRR score increases from 0.62 to 0.72 and Hits@1 increases from 0.53 to 0.64. For higher ranks (after 50), the change is minimal. Empirically, the effect of \(k\) diminishes as the number of the entities per relation becomes larger, e.g., it is \(\sim 3722\) for WN18RR in contrast to \(\sim 11\) for FB15k. We suspect that this could be due to the fact that as \(n_e \gg d_e\), most of the knowledge is learned through embedding matrices rather than the model parameters \(U\) and \(V\). To test this, we took a trained LowFER model, on WN18 dataset, and added zero mean Gaussian noise with variance in \(\{1.0, 1.25, 1.5, 1.75, 2.0\}\) to \(U\) and \(V\) and evaluated on the test set. The MRR score changed from 0.95 to \(\{0.92, 0.84, 0.65, 0.42, 0.24\}\) for each level of noise. This shows that in cases as such, the embeddings have potential to capture more knowledge than the shared parameters.

5.3. Effect of Embedding Dimension

The size of entity embedding dimension \(d_e\) accounts for the significant number of parameters in LowFER, growing linearly with number of entities \(n_e\). To study the effect, we trained our models on FB15k, with \(d_r = 30, k = 50\) constant, and varying \(d_e\) in \(\{30, 50, 100, 150, 200, 250, 300, 350, 400\}\). As can be seen in Figure 6, increasing the entity embedding dimension significantly increases the Hits@1 metric, for almost linear growth in number of parameters. However, it only improves till 300 and starts overfitting afterwards.

In Balažević et al. (2019a), authors reported \(d_e = d_r = 200\) as best choice of dimensions for TuckER on FB15k, however, we found using \(d_e = 300\) and \(d_r = 30\) better with lesser number of parameters for LowFER. For fair compari-
son, we also provide the results for $d_e = d_r = 200$ for $k$ in \{1, 5, 10, 30, 50, 100\} in Table 4. As $k$ is increased, we see an improvement over all the metrics. At $k = 100$, where we expected LowFER to match TuckER’s performance (MRR=0.795, Hits@1=0.741, $\sim$11 million parameters), it was lower ($-1.6\%$ on MRR and $-2.4\%$ on Hits@1). In comparison, our model with $d_e = 300$, $d_r = 30$ and $k = 10$ with $\sim$5.6 million parameters only, gives better results than this setting and TuckER. Therefore, at $d_e = d_r = 200$, our model is most likely overfitting.

As noted above that it could be that LowFER is overfitting therefore, we did coarse grid search over relation embedding dimension in \{30, 50, 100, 150, 200\} and $k$ in \{1, 5, 10, 30, 50, 100, 150, 200\} while keeping $d_e = 200$ fixed. We found $d_e = 50$ at $k = 150$ reaches almost the same performance as TuckER with $\sim$10.6M parameters compared to TuckER’s $\sim$11.3M parameters. We also experimented with $l_2$-regularization (Reg) and noted minor improvements, with regularization strength 0.0005. Table 5 summarizes these results. Note that all the experiments reported in main results (Table 2) were without any regularization. In general, we only noticed slight improvements in FB15k with $l_2$-regularization.

5.4. Analysis of Relation Results

Link prediction models that can discover relation types automatically without prior knowledge indicate better generalization. As shown, and discussed in section 4, LowFER, among other models (Table 1), can learn to capture all relation types without additional constraints. However, in practice, these bounds are loose and require very large dimensions, raising an inspection into their performance on different relation types. In Kazemi & Poole (2018), it was identified that WN18 contains redundant relations, i.e., $\forall e_i, e_j \in E : (e_i, r_1, e_j) \in T \iff (e_j, r_2, e_i) \in T$, such as $<$hypernym, hyponym$>$, $<$meronym, holonym$>$ etc. To alleviate this, Dettmers et al. (2018) proposed WN18RR with such relations removed, since knowledge about one can help infer the knowledge about the other. Table 6 shows the per relation results of LowFER and TuckER on WN18 and WN18RR. We see that performance drops for 7 relations, with an average performance decrease of $-70.6\%$ and $-69.3\%$ for LowFER and TuckER respectively (with highest decrease on member_of_domain_usage for both). For symmetric relations (such as derivationally_related_form), the performance is approximately the same where we observe severe limitation to model asymmetry. We believe this is because LowFER (also TuckER) is constraint-free and adding certain constraints based on background knowledge is necessary to improve the model’s accuracy. SimplE is the only fully expressive model that has formally shown to address these limitations (cf. Proposition 3, 4 and 5 in Kazemi & Poole (2018)). Since LowFER subsumes SimplE therefore, such rules can be studied for extending LowFER to incorporate the background knowledge.

6. Conclusion

This work proposes a simple and parameter efficient fully expressive linear model that is theoretically well sound and performs on par or state-of-the-art in practice. We showed that LowFER generalizes to other linear models in KGC, providing a unified theoretical view. It offers a strong baseline to the deep learning based models and raises further interest into the study of linear models. We also highlighted some limitations with respect to gains on harder relations, which still pose a challenge. We conclude that the constraint-free and parameter efficient linear models, which allow for parameter sharing, are better from a modeling perspective, but are still similarly limited in learning difficult relations. Therefore, studying the trade-off between parameters sharing and constraints becomes an important future work.

Acknowledgements

The authors would like to thank the anonymous reviewers for helpful feedback and gratefully acknowledge the use of code released by Balažević et al. (2019a). The work was partially funded by the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 777107 through the project Precise4Q and by the German Federal Ministry of Education and Research (BMBF) through the project DEEPLIE (01IW17001).

Table 5. Link prediction results on FB15k with $d_e = 200$, $d_r = 50$, $k = 150$ and $l_2$-regularization 0.0005.

<table>
<thead>
<tr>
<th>Model</th>
<th>Params (M)</th>
<th>MRR</th>
<th>Hits@1</th>
<th>Hits@3</th>
<th>Hits@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>TuckER</td>
<td>11.3</td>
<td>0.795</td>
<td>0.741</td>
<td>0.833</td>
<td>0.892</td>
</tr>
<tr>
<td>LowFER-$k^*$</td>
<td>10.6</td>
<td>0.795</td>
<td>0.739</td>
<td>0.831</td>
<td>0.891</td>
</tr>
<tr>
<td>LowFER-$k^*$ + Reg</td>
<td>10.6</td>
<td>0.802</td>
<td>0.749</td>
<td>0.837</td>
<td>0.892</td>
</tr>
</tbody>
</table>

Table 6. Relation specific test set results on WN18 and WN18RR with LowFER-$k^*$ and best reported TuckER model (Balažević et al., 2019a).

<table>
<thead>
<tr>
<th>Relation</th>
<th>LowFER</th>
<th>TuckER</th>
<th>LowFER</th>
<th>TuckER</th>
</tr>
</thead>
<tbody>
<tr>
<td>also_see</td>
<td>0.638</td>
<td>0.630</td>
<td>0.627</td>
<td>0.614</td>
</tr>
<tr>
<td>derivationally_related_form</td>
<td>0.954</td>
<td>0.956</td>
<td>0.957</td>
<td>0.957</td>
</tr>
<tr>
<td>has_part</td>
<td>0.944</td>
<td>0.945</td>
<td>0.938</td>
<td>0.939</td>
</tr>
<tr>
<td>hypernym</td>
<td>0.961</td>
<td>0.962</td>
<td>0.969</td>
<td>0.969</td>
</tr>
<tr>
<td>instance_hypernym</td>
<td>0.986</td>
<td>0.982</td>
<td>0.976</td>
<td>0.969</td>
</tr>
<tr>
<td>member_meronym</td>
<td>0.930</td>
<td>0.927</td>
<td>0.955</td>
<td>0.955</td>
</tr>
<tr>
<td>member_of_domain_region</td>
<td>0.885</td>
<td>0.885</td>
<td>0.900</td>
<td>0.900</td>
</tr>
<tr>
<td>member_of_domain_usage</td>
<td>0.917</td>
<td>0.917</td>
<td>0.925</td>
<td>0.925</td>
</tr>
<tr>
<td>similar_to</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>synset_domain_topic_of</td>
<td>0.956</td>
<td>0.952</td>
<td>0.494</td>
<td>0.494</td>
</tr>
<tr>
<td>verb_group</td>
<td>0.974</td>
<td>0.974</td>
<td>0.974</td>
<td>0.974</td>
</tr>
</tbody>
</table>
LowFER: Low-rank Bilinear Pooling for Link Prediction

References


LowFER: Low-rank Bilinear Pooling for Link Prediction


