TuckER: Tensor Factorization for Knowledge Graph Completion

Ivana Balažević[1] **Carl Allen**[1] **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 structured representations of real world facts. However, they typically contain only a small subset of all possible facts. Link prediction is a task of inferring missing facts based on existing ones. We propose TuckER, a relatively straightforward but powerful linear model based on Tucker decomposition of the binary tensor representation of knowledge graph triples. TuckER outperforms previous state-of-the-art models across standard link prediction datasets, acting as a strong baseline for more elaborate models. We show that TuckER is a fully expressive model, derive sufficient bounds on its embedding dimensionalities and demonstrate that several previously introduced linear models can be viewed as special cases of TuckER.

1 Introduction

Vast amounts of information available in the world can be represented succinctly as *entities* and *relations* between them. *Knowledge graphs* are large, graph-structured databases which store facts in triple form (e_s, r, e_o), with e_s and e_o representing subject and object entities and r a relation. However, far from all available information is currently stored in existing knowledge graphs and manually adding new information is costly, which creates the need for algorithms that are able to automatically infer missing facts.

Knowledge graphs can be represented as a third-order binary tensor, where each element corresponds to a triple, 1 indicating a true fact and 0 indicating the unknown (either a false or a missing fact). The task of *link prediction* is to predict whether two entities are related, based on known facts already present in a knowledge graph, i.e. to infer which of the 0 entries in the tensor are indeed false, and which are missing but actually true.

A large number of approaches to link prediction so far have been linear, based on various methods of factorizing the third-order binary tensor (Nickel et al., 2011; Yang et al., 2015; Trouillon et al., 2016; Kazemi and Poole, 2018). Recently, state-of-the-art results have been achieved using non-linear convolutional models (Dettmers et al., 2018; Balažević et al., 2019). Despite achieving very good performance, the fundamental problem with deep, non-linear models is that they are non-transparent and poorly understood, as opposed to more mathematically principled and widely studied tensor decomposition models.

In this paper, we introduce TuckER (E stands for entities, R for relations), a straightforward linear model for link prediction on knowledge graphs, based on *Tucker decomposition* (Tucker, 1966) of the binary tensor of triples, acting as a strong baseline for more elaborate models. Tucker decomposition, used widely in machine learning (Schein et al., 2016; Ben-Younes et al., 2017; Yang and Hospedales, 2017), factorizes a tensor into a core tensor multiplied by a matrix along each mode. It can be thought of as a form of higher-order SVD in the special case where matrices are orthogonal and the core tensor is "all-orthogonal" (Kroonenberg and De Leeuw, 1980). In our case, rows of the matrices contain entity and relation embeddings, while entries of the core tensor determine the level of interaction between them. Subject and object entity embedding matrices are assumed equivalent, i.e. we make no distinction between the embeddings of an entity depending on whether it appears as a subject or as an object in a particular triple. Due to the low rank of the core tensor, TuckER benefits from *multi-task learning* by parameter sharing across relations.

A link prediction model should have enough expressive power to represent all relation types (e.g. symmetric, asymmetric, transitive). We thus show

that TuckER is *fully expressive*, i.e. given any ground truth over the triples, there exists an assignment of values to the entity and relation embeddings that accurately separates the true triples from false ones. We also derive a dimensionality bound which guarantees full expressiveness.

Finally, we show that several previous state-of-the-art linear models, RESCAL (Nickel et al., 2011), DistMult (Yang et al., 2015), ComplEx (Trouillon et al., 2016) and SimplE (Kazemi and Poole, 2018), are special cases of TuckER.

In summary, key contributions of this paper are:

- proposing TuckER, a new *linear* model for link prediction on knowledge graphs, that is simple, expressive and achieves *state-of-the-art results* across all standard datasets;
- proving that TuckER is *fully expressive* and deriving a bound on the embedding dimensionality for full expressiveness; and
- showing how TuckER subsumes several previously proposed tensor factorization approaches to link prediction.

2 Related Work

Several *linear* models for link prediction have previously been proposed:

RESCAL (Nickel et al., 2011) optimizes a scoring function containing a bilinear product between subject and object entity vectors and a full rank relation matrix. Although a very expressive and powerful model, RESCAL is prone to overfitting due to its large number of parameters, which increases quadratically in the embedding dimension with the number of relations in a knowledge graph.

DistMult (Yang et al., 2015) is a special case of RESCAL with a diagonal matrix per relation, which reduces overfitting. However, the linear transformation performed on entity embedding vectors in DistMult is limited to a stretch. The binary tensor learned by DistMult is symmetric in the subject and object entity mode and thus DistMult cannot model asymmetric relations.

ComplEx (Trouillon et al., 2016) extends DistMult to the complex domain. Subject and object entity embeddings for the same entity are complex conjugates, which introduces asymmetry into the tensor decomposition and thus enables ComplEx to model asymmetric relations.

SimplE (Kazemi and Poole, 2018) is based on Canonical Polyadic (CP) decomposition (Hitchcock, 1927), in which subject and object entity embeddings for the same entity are independent (note that DistMult is a special case of CP). SimplE's scoring function alters CP to make subject and object entity embedding vectors dependent on each other by computing the average of two terms, first of which is a bilinear product of the subject entity head embedding, relation embedding and object entity tail embedding and the second is a bilinear product of the object entity head embedding, inverse relation embedding and subject entity tail embedding.

Recently, state-of-the-art results have been achieved with *non-linear* models:

ConvE (Dettmers et al., 2018) performs a global 2D convolution operation on the subject entity and relation embedding vectors, after they are reshaped to matrices and concatenated. The obtained feature maps are flattened, transformed through a linear layer, and the inner product is taken with all object entity vectors to generate a score for each triple. Whilst results achieved by ConvE are impressive, its reshaping and concatenating of vectors as well as using 2D convolution on word embeddings is unintuitive.

HypER (Balažević et al., 2019) is a simplified convolutional model, that uses a hypernetwork to generate 1D convolutional filters for each relation, extracting relation-specific features from subject entity embeddings. The authors show that convolution is a way of introducing sparsity and parameter tying and that HypER can be understood in terms of tensor factorization up to a non-linearity, thus placing HypER closer to the well established family of factorization models. The drawback of HypER is that it sets most elements of the core weight tensor to 0, which amounts to hard regularization, rather than letting the model learn which parameters to use via soft regularization.

Scoring functions of all models described above and TuckER are summarized in Table 1.

3 Background

Let $\mathcal{E}$ denote the set of all entities and $\mathcal{R}$ the set of all relations present in a knowledge graph. A triple is represented as (e_s, r, e_o), with $e_s, e_o \in \mathcal{E}$ denoting subject and object entities respectively and $r \in \mathcal{R}$ the relation between them.

3.1 Link Prediction

In link prediction, we are given a subset of all true triples and the aim is to learn a *scoring function* ϕ

Model	Scoring Function	Relation Parameters	Space Complexity
RESCAL (Nickel et al., 2011)	$\mathbf{e}_s^\top \mathbf{W}_r \mathbf{e}_o$	$\mathbf{W}_r \in \mathbb{R}^{{d_e}^2}$	$\mathcal{O}(n_e d_e + n_r d_r^2)$
DistMult (Yang et al., 2015)	$\langle \mathbf{e}_s, \mathbf{w}_r, \mathbf{e}_o \rangle$	$\mathbf{w}_r \in \mathbb{R}^{d_e}$	$\mathcal{O}(n_e d_e + n_r d_e)$
ComplEx (Trouillon et al., 2016)	$\text{Re}(\langle \mathbf{e}_s, \mathbf{w}_r, \overline{\mathbf{e}}_o \rangle)$	$\mathbf{w}_r \in \mathbb{C}^{d_e}$	$\mathcal{O}(n_e d_e + n_r d_e)$
ConvE (Dettmers et al., 2018)	$f(\text{vec}(f([\underline{\mathbf{e}}_s; \underline{\mathbf{w}}_r] * w))\mathbf{W})\mathbf{e}_o$	$\mathbf{w}_r \in \mathbb{R}^{d_r}$	$\mathcal{O}(n_e d_e + n_r d_r)$
SimplE (Kazemi and Poole, 2018)	$\frac{1}{2}(\langle \mathbf{h}_{e_s}, \mathbf{w}_r, \mathbf{t}_{e_o} \rangle + \langle \mathbf{h}_{e_o}, \mathbf{w}_{r^{-1}}, \mathbf{t}_{e_s} \rangle)$	$\mathbf{w}_r \in \mathbb{R}^{d_e}$	$\mathcal{O}(n_e d_e + n_r d_e)$
HypER (Balažević et al., 2019)	$f(\text{vec}(\mathbf{e}_s * \text{vec}^{-1}(\mathbf{w}_r \mathbf{H}))\mathbf{W})\mathbf{e}_o$	$\mathbf{w}_r \in \mathbb{R}^{d_r}$	$\mathcal{O}(n_e d_e + n_r d_r)$
TuckER (ours)	$\mathcal{W} \times_1 \mathbf{e}_s \times_2 \mathbf{w}_r \times_3 \mathbf{e}_o$	$\mathbf{w}_r \in \mathbb{R}^{d_r}$	$\mathcal{O}(n_e d_e + n_r d_r)$

Table 1: Scoring functions of state-of-the-art link prediction models, the dimensionality of their relation parameters, and significant terms of their space complexity. d_e and d_r are the dimensionalities of entity and relation embeddings, while n_e and n_r denote the number of entities and relations respectively. $\overline{\mathbf{e}}_o \in \mathbb{C}^{d_e}$ is the complex conjugate of $\mathbf{e}_o$, $\underline{\mathbf{e}}_s, \underline{\mathbf{w}}_r \in \mathbb{R}^{d_w \times d_h}$ denote a 2D reshaping of $\mathbf{e}_s$ and $\mathbf{w}_r$ respectively, $\mathbf{h}_{e_s}, \mathbf{t}_{e_s} \in \mathbb{R}^{d_e}$ are the head and tail entity embedding of entity e_s, and $\mathbf{w}_{r^{-1}} \in \mathbb{R}^{d_r}$ is the embedding of relation r^{-1} (which is the inverse of relation r). $*$ is the convolution operator, $\langle \cdot \rangle$ denotes the dot product and $\times_n$ denotes the tensor product along the n-th mode, f is a non-linear function, and $\mathcal{W} \in \mathbb{R}^{d_e \times d_e \times d_r}$ is the core tensor of a Tucker decomposition.

that assigns a score $s = \phi(e_s, r, e_o) \in \mathbb{R}$ which indicates whether a triple is true, with the ultimate goal of being able to correctly score all missing triples. The scoring function is either a specific form of tensor factorization in the case of linear models or a more complex (deep) neural network architecture for non-linear models. Typically, a positive score for a particular triple indicates a true fact predicted by the model, while a negative score indicates a false one. With most recent models, a non-linearity such as the logistic sigmoid function is typically applied to the score to give a corresponding probability prediction $p = \sigma(s) \in [0, 1]$ as to whether a certain fact is true.

3.2 Tucker Decomposition

Tucker decomposition, named after Ledyard R. Tucker (Tucker, 1964), decomposes a tensor into a set of matrices and a smaller core tensor. In a three-mode case, given the original tensor $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$, Tucker decomposition outputs a tensor $\mathcal{Z} \in \mathbb{R}^{P \times Q \times R}$ and three matrices $\mathbf{A} \in \mathbb{R}^{I \times P}$, $\mathbf{B} \in \mathbb{R}^{J \times Q}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$:

$$\mathcal{X} \approx \mathcal{Z} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C}, \tag{1}$$

with $\times_n$ indicating the tensor product along the n-th mode. *Factor matrices* $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$, when orthogonal, can be thought of as the principal components in each mode. Elements of the *core tensor* $\mathcal{Z}$ show the level of interaction between the different components. Typically, P, Q, R are smaller than I, J, K respectively, so $\mathcal{Z}$ can be thought of as a compressed version of $\mathcal{X}$. Tucker decomposition is not unique, i.e. we can transform $\mathcal{Z}$ without affecting the fit if we apply the inverse transformation to $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ (Kolda and Bader, 2009).

4 Tucker Decomposition for Link Prediction

We propose a model that uses Tucker decomposition for link prediction on the binary tensor representation of a knowledge graph, with entity embedding matrix $\mathbf{E}$ that is equivalent for subject and object entities, i.e. $\mathbf{E} = \mathbf{A} = \mathbf{C} \in \mathbb{R}^{n_e \times d_e}$ and relation embedding matrix $\mathbf{R} = \mathbf{B} \in \mathbb{R}^{n_r \times d_r}$, where n_e and n_r represent the number of entities and relations and d_e and d_r the dimensionality of entity and relation embedding vectors.

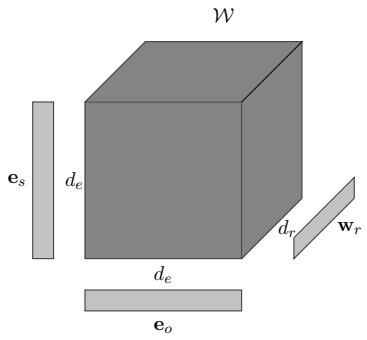


Figure 1: Visualization of the TuckER architecture.

We define the scoring function for TuckER as:

$$\phi(e_s, r, e_o) = \mathcal{W} \times_1 \mathbf{e}_s \times_2 \mathbf{w}_r \times_3 \mathbf{e}_o, \tag{2}$$

where $\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}^{d_e}$ are the rows of $\mathbf{E}$ representing the subject and object entity embedding vectors, $\mathbf{w}_r \in \mathbb{R}^{d_r}$ the rows of $\mathbf{R}$ representing the relation embedding vector and $\mathcal{W} \in \mathbb{R}^{d_e \times d_r \times d_e}$ is the core tensor. We apply logistic sigmoid to each score $\phi(e_s, r, e_o)$ to obtain the predicted probability p of a triple being true. Visualization of the TuckER architecture can be seen in Figure 1. As proven in Section 5.1, TuckER is *fully expressive*. Further, its number of parameters increases *linearly* with

respect to entity and relation embedding dimensionality d_e and d_r, as the number of entities and relations increases, since the number of parameters of $\mathcal{W}$ depends only on the entity and relation embedding dimensionality and not on the number of entities or relations. By having the core tensor $\mathcal{W}$, unlike simpler models such as DistMult, ComplEx and SimplE, TuckER does not encode all the learned knowledge into the embeddings; some is stored in the core tensor and shared between all entities and relations through *multi-task learning*. Rather than learning distinct relation-specific matrices, the core tensor of TuckER can be viewed as containing a shared pool of "prototype" relation matrices, which are linearly combined according to the parameters in each relation embedding.

4.1 Training

Since the logistic sigmoid is applied to the scoring function to approximate the true binary tensor, the implicit underlying tensor is comprised of $-\infty$ and ∞. Given this prevents an explicit analytical factorization, we use numerical methods to train TuckER. We use the standard data augmentation technique, first used by Dettmers et al. (2018) and formally described by Lacroix et al. (2018), of adding reciprocal relations for every triple in the dataset, i.e. we add (e_o, r^{-1}, e_s) for every (e_s, r, e_o). Following the training procedure introduced by Dettmers et al. (2018) to speed up training, we use *1-N scoring*, i.e. we simultaneously score entity-relation pairs (e_s, r) and (e_o, r^{-1}) with all entities $e_o \in \mathcal{E}$ and $e_s \in \mathcal{E}$ respectively, in contrast to *1-1 scoring*, where individual triples (e_s, r, e_o) and (e_o, r^{-1}, e_s) are trained one at a time. The model is trained to minimize the Bernoulli negative log-likelihood loss function. A component of the loss for one entity-relation pair with all others entities is defined as:

$$L = -\frac{1}{n_e}\sum_{i=1}^{n_e}(\mathbf{y}^{(i)}\log(\mathbf{p}^{(i)}) + (1-\mathbf{y}^{(i)})\log(1-\mathbf{p}^{(i)})), \tag{3}$$

where $\mathbf{p} \in \mathbb{R}^{n_e}$ is the vector of predicted probabilities and $\mathbf{y} \in \mathbb{R}^{n_e}$ is the binary label vector.

5 Theoretical Analysis

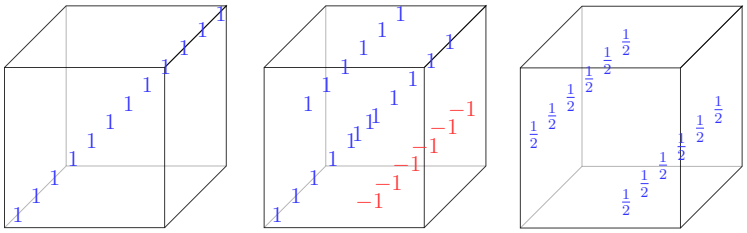
5.1 Full Expressiveness and Embedding Dimensionality

A tensor factorization model is fully expressive if for any ground truth over all entities and relations, there exist entity and relation embeddings that accurately separate true triples from the false. As shown in (Trouillon et al., 2017), ComplEx is fully expressive with the embedding dimensionality bound $d_e = d_r = n_e \cdot n_r$. Similarly to ComplEx, Kazemi and Poole (2018) show that SimplE is fully expressive with entity and relation embeddings of size $d_e = d_r = \textit{min}(n_e \cdot n_r, \gamma + 1)$, where γ represents the number of true facts. They further prove other models are not fully expressive: DistMult, because it cannot model asymmetric relations; and transitive models such as TransE (Bordes et al., 2013) and its variants FTransE (Feng et al., 2016) and STransE (Nguyen et al., 2016), because of certain contradictions that they impose between different relation types. By Theorem 1, we establish the bound on entity and relation embedding dimensionality (i.e. decomposition rank) that guarantees full expressiveness of TuckER.

Theorem 1. *Given any ground truth over a set of entities $\mathcal{E}$ and relations $\mathcal{R}$, there exists a TuckER model with entity embeddings of dimensionality $d_e = n_e$ and relation embeddings of dimensionality $d_r = n_r$, where $n_e = |\mathcal{E}|$ is the number of entities and $n_r = |\mathcal{R}|$ the number of relations, that accurately represents that ground truth.*

Proof. Let $\mathbf{e}_s$ and $\mathbf{e}_o$ be the n_e-dimensional one-hot binary vector representations of subject and object entities e_s and e_o respectively and $\mathbf{w}_r$ the n_r-dimensional one-hot binary vector representation of relation r. For each subject entity $e_s^{(i)}$, relation $r^{(j)}$ and object entity $e_o^{(k)}$, we let the i-th, j-th and k-th element respectively of the corresponding vectors $\mathbf{e}_s$, $\mathbf{w}_r$ and $\mathbf{e}_o$ be 1 and all other elements 0. Further, we set the ijk element of the tensor $\mathcal{W} \in \mathbb{R}^{n_e \times n_r \times n_e}$ to 1 if the fact (e_s, r, e_o) holds and -1 otherwise. Thus the product of the entity embeddings and the relation embedding with the core tensor, after applying the logistic sigmoid, accurately represents the original tensor. □

The purpose of Theorem 1 is to prove that TuckER is capable of potentially capturing all information (and noise) in the data. In practice however, we expect the embedding dimensionalities needed for full reconstruction of the underlying binary tensor to be much smaller than the bound stated above, since the assignment of values to the tensor is not random but follows a certain structure, otherwise nothing unknown could be predicted. Even more so, low decomposition rank is actually a desired property of any bilin-



(a) DistMult (b) ComplEx (c) SimplE

Figure 2: Constraints imposed on the values of core tensor $\mathcal{Z} \in \mathbb{R}^{d_e \times d_e \times d_e}$ for DistMult and $\mathcal{Z} \in \mathbb{R}^{2d_e \times 2d_e \times 2d_e}$ for ComplEx and SimplE. Elements that are set to 0 are represented in white.

ear link prediction model, forcing it to learn that structure and generalize to new data, rather than simply memorizing the input. In general, we expect TuckER to perform better than ComplEx and SimplE with embeddings of lower dimensionality due to parameter sharing in the core tensor (shown empirically in Section 6.4), which could be of importance for efficiency in downstream tasks.

5.2 Relation to Previous Linear Models

Several previous tensor factorization models can be viewed as a special case of TuckER:

RESCAL (Nickel et al., 2011) Following the notation introduced in Section 3.2, the RESCAL scoring function (see Table 1) has the form:

$$\mathcal{X} \approx \mathcal{Z} \times_1 \mathbf{A} \times_3 \mathbf{C}. \tag{4}$$

This corresponds to Equation 1 with $I = K = n_e$, $P = R = d_e$, $Q = J = n_r$ and $\mathbf{B} = \mathbf{I}_J$ the $J \times J$ identity matrix. This is also known as Tucker2 decomposition (Kolda and Bader, 2009). As is the case with TuckER, the entity embedding matrix of RESCAL is shared between subject and object entities, i.e. $\mathbf{E} = \mathbf{A} = \mathbf{C} \in \mathbb{R}^{n_e \times d_e}$ and the relation matrices $\mathbf{W}_r \in \mathbb{R}^{d_e \times d_e}$ are the $d_e \times d_e$ slices of the core tensor $\mathcal{Z}$. As mentioned in Section 2, the drawback of RESCAL compared to TuckER is that its number of parameters grows *quadratically* in the entity embedding dimension d_e as the number of relations increases.

DistMult (Yang et al., 2015) The scoring function of DistMult (see Table 1) can be viewed as equivalent to that of TuckER (see Equation 1) with a core tensor $\mathcal{Z} \in \mathbb{R}^{P \times Q \times R}$, $P = Q = R = d_e$, which is *superdiagonal* with 1s on the superdiagonal, i.e. all elements z_{pqr} with $p = q = r$ are 1 and all the other elements are 0 (as shown in Figure 2a). Rows of $\mathbf{E} = \mathbf{A} = \mathbf{C} \in \mathbb{R}^{n_e \times d_e}$ contain subject and object entity embedding vectors $\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}^{d_e}$ and rows of $\mathbf{R} = \mathbf{B} \in \mathbb{R}^{n_r \times d_e}$ contain relation embedding vectors $\mathbf{w}_r \in \mathbb{R}^{d_e}$. It is interesting to note that the TuckER interpretation of the DistMult scoring function, given that matrices $\mathbf{A}$ and $\mathbf{C}$ are identical, can alternatively be interpreted as a special case of CP decomposition (Hitchcock, 1927), since Tucker decomposition with a superdiagonal core tensor is equivalent to CP decomposition. Due to enforced symmetry in subject and object entity mode, DistMult cannot learn to represent asymmetric relations.

ComplEx (Trouillon et al., 2016) *Bilinear models* represent subject and object entity embeddings as vectors $\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}^{d_e}$, relation as a matrix $\mathbf{W}_r \in \mathbb{R}^{d_e \times d_e}$ and the scoring function as a bilinear product $\phi(e_s, r, e_o) = \mathbf{e}_s \mathbf{W}_r \mathbf{e}_o$. It is trivial to show that both RESCAL and DistMult belong to the family of bilinear models. As explained by Kazemi and Poole (2018), ComplEx can be considered a bilinear model with the real and imaginary part of an embedding for each entity concatenated in a single vector, $[\text{Re}(\mathbf{e}_s); \text{Im}(\mathbf{e}_s)] \in \mathbb{R}^{2d_e}$ for subject, $[\text{Re}(\mathbf{e}_o); \text{Im}(\mathbf{e}_o)] \in \mathbb{R}^{2d_e}$ for object, and a relation matrix $\mathbf{W}_r \in \mathbb{R}^{2d_e \times 2d_e}$, constrained so that its leading diagonal contains duplicated elements of $\text{Re}(\mathbf{w}_r)$, its d_e-diagonal elements of $\text{Im}(\mathbf{w}_r)$ and its -d_e-diagonal elements of -$\text{Im}(\mathbf{w}_r)$, with all other elements set to 0, where d_e and -d_e represent offsets from the leading diagonal.

Similarly to DistMult, we can regard the scoring function of ComplEx (see Table 1) as equivalent to the scoring function of TuckER (see Equation 1), with core tensor $\mathcal{Z} \in \mathbb{R}^{P \times Q \times R}$, $P = Q = R = 2d_e$, where $3d_e$ elements on different tensor diagonals are set to 1, d_e elements on one tensor diagonal are set to -1 and all other elements are set to 0 (see Figure 2b). This shows that the scoring function of ComplEx, which computes a bilinear product with *complex* entity and relation embeddings and disregards the imaginary part of the obtained result, is equivalent to a hard regularization of the core tensor of TuckER in the *real* domain.

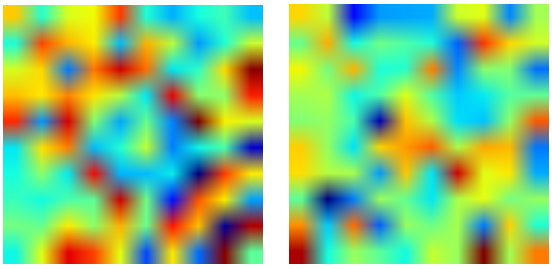
SimplE (Kazemi and Poole, 2018) The authors show that SimplE belongs to the family of bilinear models by concatenating embeddings for head and tail entities for both subject and object into vectors $[\mathbf{h}_{e_s}; \mathbf{t}_{e_s}] \in \mathbb{R}^{2d_e}$ and $[\mathbf{h}_{e_o}; \mathbf{t}_{e_o}] \in \mathbb{R}^{2d_e}$ and constraining the relation matrix $\mathbf{W}_r \in \mathbb{R}^{2d_e \times 2d_e}$ so that it contains the relation embedding vector $\frac{1}{2}\mathbf{w}_r$ on its d_e-diagonal and the inverse relation embedding vector $\frac{1}{2}\mathbf{w}_{r^{-1}}$ on its -d_e-diagonal and 0s elsewhere. The SimplE scoring function (see Table 1) is therefore equivalent to that of TuckER (see Equation 1), with core tensor $\mathcal{Z} \in \mathbb{R}^{P \times Q \times R}$, $P = Q = R = 2d_e$, where $2d_e$ elements on two tensor diagonals are set to $\frac{1}{2}$ and all other elements are set to 0 (see Figure 2c).

5.3 Representing Asymmetric Relations

Each relation in a knowledge graph can be characterized by a certain set of properties, such as symmetry, reflexivity, transitivity. So far, there have been two possible ways in which linear link prediction models introduce *asymmetry* into factorization of the binary tensor of triples:

- distinct (although possibly related) embeddings for subject and object entities and a diagonal matrix (or equivalently a vector) for each relation, as is the case with models such as ComplEx and SimplE; or
- equivalent subject and object entity embeddings and each relation represented by a full rank matrix, which is the case with RESCAL.

The latter approach appears more intuitive, since asymmetry is a property of the relation, rather than the entities. However, the drawback of the latter approach is quadratic growth of parameter number with the number of relations, which often leads to overfitting, especially for relations with a small number of training triples. TuckER overcomes this by representing relations as vectors $\mathbf{w}_r$, which makes the parameter number grow linearly with the number of relations, while still keeping the desirable property of allowing relations to be asymmetric by having an asymmetric *relation-agnostic* core tensor $\mathcal{W}$, rather than encoding the relation-specific information in the entity embeddings. Multiplying $\mathcal{W} \in \mathbb{R}^{d_e \times d_r \times d_e}$ with $\mathbf{w}_r \in \mathbb{R}^{d_r}$ along the second mode, we obtain a full rank relation-specific matrix $\mathbf{W}_r \in \mathbb{R}^{d_e \times d_e}$, which can perform all possible linear transformations on the entity embeddings, i.e. rotation, reflection or stretch, and is thus also capable of modeling asymmetry. Regardless of what kind of transformation is needed for modeling a particular relation, TuckER can learn it from the data. To demonstrate this, we show sample heatmaps of learned relation matrices $\mathbf{W}_r$ for a WordNet symmetric relation "derivationally_related_form" and an asymmetric relation "hypernym" in Figure 3, where one can see that TuckER learns to model the symmetric relation with the relation matrix that is approximately symmetric about the main diagonal, whereas the matrix belonging to the asymmetric relation exhibits no obvious structure.



(a) $\mathbf{W}_{\text{derivationally\_related\_form}}$ (b) $\mathbf{W}_{\text{hypernym}}$

Figure 3: Learned relation matrices for a symmetric (derivationally_related_form) and an asymmetric (hypernym) WN18RR relation. $\mathbf{W}_{\text{derivationally\_related\_form}}$ is approximately symmetric about the leading diagonal.

6 Experiments and Results

6.1 Datasets

We evaluate TuckER using four standard link prediction datasets (see Table 2):

FB15k (Bordes et al., 2013) is a subset of Freebase, a large database of real world facts.

FB15k-237 (Toutanova et al., 2015) was created from FB15k by removing the inverse of many relations that are present in the training set from validation and test sets, making it more difficult for simple models to do well.

WN18 (Bordes et al., 2013) is a subset of WordNet, a hierarchical database containing lexical relations between words.

WN18RR (Dettmers et al., 2018) is a subset of WN18, created by removing the inverse relations from validation and test sets.

6.2 Implementation and Experiments

We implement TuckER in PyTorch (Paszke et al., 2017) and make our code available on GitHub.[1]

We choose all hyper-parameters by random search based on validation set performance. For

[1] https://github.com/ibalazevic/TuckER

FB15k and FB15k-237, we set entity and relation embedding dimensionality to $d_e = d_r = 200$. For WN18 and WN18RR, which both contain a significantly smaller number of relations relative to the number of entities as well as a small number of relations compared to FB15k and FB15k-237, we set $d_e = 200$ and $d_r = 30$. We use batch normalization (Ioffe and Szegedy, 2015) and dropout (Srivastava et al., 2014) to speed up training. We find that lower dropout values $(0.1, 0.2)$ are required for datasets with a higher number of training triples per relation and thus less risk of overfitting (WN18 and WN18RR), whereas higher dropout values $(0.3, 0.4, 0.5)$ are required for FB15k and FB15k-237. We choose the learning rate from $\{0.01, 0.005, 0.003, 0.001, 0.0005\}$ and learning rate decay from $\{1, 0.995, 0.99\}$. We find the following combinations of learning rate and learning rate decay to give the best results: $(0.003, 0.99)$ for FB15k, $(0.0005, 1.0)$ for FB15k-237, $(0.005, 0.995)$ for WN18 and $(0.01, 1.0)$ for WN18RR (see Table 5 in the Appendix A for a complete list of hyper-parameter values on each dataset). We train the model using Adam (Kingma and Ba, 2015) with the batch size 128.

At evaluation time, for each test triple we generate n_e candidate triples by combining the test entity-relation pair with all possible entities $\mathcal{E}$, ranking the scores obtained. We use the *filtered* setting (Bordes et al., 2013), i.e. all known true triples are removed from the candidate set except for the current test triple. We use evaluation metrics standard across the link prediction literature: mean reciprocal rank (MRR) and hits@k, $k \in \{1, 3, 10\}$. Mean reciprocal rank is the average of the inverse of the mean rank assigned to the true triple over all candidate triples. Hits@k measures the percentage of times a true triple is ranked within the top k candidate triples.

Dataset	# Entities (n_e)	# Relations (n_r)
FB15k	14,951	1,345
FB15k-237	14,541	237
WN18	40,943	18
WN18RR	40,943	11

Table 2: Dataset statistics.

6.3 Link Prediction Results

Link prediction results on all datasets are shown in Tables 3 and 4. Overall, TuckER outperforms previous state-of-the-art models on all metrics across all datasets (apart from hits@10 on WN18 where a non-linear model, R-GCN, does better). Results achieved by TuckER are not only better than those of other linear models, such as DistMult, ComplEx and SimplE, but also better than the results of many more complex deep neural network and reinforcement learning architectures, e.g. R-GCN, MINERVA, ConvE and HypER, demonstrating the expressive power of linear models and supporting our claim that simple linear models should serve as a baseline before moving onto more elaborate models.

Even with fewer parameters than ComplEx and SimplE at $d_e = 200$ and $d_r = 30$ on WN18RR ($\sim$9.4 vs $\sim$16.4 million), TuckER consistently obtains better results than any of those models. We believe this is because TuckER exploits knowledge sharing between relations through the core tensor, i.e. multi-task learning. This is supported by the fact that the margin by which TuckER outperforms other linear models is notably increased on datasets with a large number of relations. For example, improvement on FB15k is $+14\%$ over ComplEx and $+8\%$ over SimplE on the toughest hits@1 metric. To our knowledge, ComplEx-N3 (Lacroix et al., 2018) is the only other linear link prediction model that benefits from multi-task learning. There, rank regularization of the embedding matrices is used to encourage a low-rank factorization, thus forcing parameter sharing between relations. We do not include their published results in Tables 3 and 4, since they use the highly non-standard $d_e = d_r = 2000$ and thus a far larger parameter number (18x more parameters than TuckER on WN18RR; 5.5x on FB15k-237), making their results incomparable to those typically reported, including our own. However, running their model with equivalent parameter number to TuckER shows comparable performance, supporting our belief that the two models both attain the benefits of multi-task learning, although by different means.

6.4 Influence of Parameter Sharing

The ability of knowledge sharing through the core tensor suggests that TuckER should need a lower number of parameters for obtaining good results than ComplEx or SimplE. To test this, we re-implement ComplEx and SimplE with reciprocal relations, 1-N scoring, batch normalization and dropout for fair comparison, perform random search to choose best hyper-parameters

		WN18RR				FB15k-237			
	Linear	MRR	Hits@10	Hits@3	Hits@1	MRR	Hits@10	Hits@3	Hits@1
DistMult (Yang et al., 2015)	yes	.430	.490	.440	.390	.241	.419	.263	.155
ComplEx (Trouillon et al., 2016)	yes	.440	.510	.460	.410	.247	.428	.275	.158
Neural LP (Yang et al., 2017)	no	–	–	–	–	.250	.408	–	–
R-GCN (Schlichtkrull et al., 2018)	no	–	–	–	–	.248	.417	.264	.151
MINERVA (Das et al., 2018)	no	–	–	–	–	–	.456	–	–
ConvE (Dettmers et al., 2018)	no	.430	.520	.440	.400	.325	.501	.356	.237
HypER (Balažević et al., 2019)	no	.465	.522	.477	.436	.341	.520	.376	.252
M-Walk (Shen et al., 2018)	no	.437	–	.445	.414	–	–	–	–
RotatE (Sun et al., 2019)	no	–	–	–	–	.297	.480	.328	.205
TuckER (ours)	yes	**.470**	**.526**	**.482**	**.443**	**.358**	**.544**	**.394**	**.266**

Table 3: Link prediction results on WN18RR and FB15k-237. The RotatE (Sun et al., 2019) results are reported without their self-adversarial negative sampling (see Appendix H in the original paper) for fair comparison.

		WN18				FB15k			
	Linear	MRR	Hits@10	Hits@3	Hits@1	MRR	Hits@10	Hits@3	Hits@1
TransE (Bordes et al., 2013)	no	–	.892	–	–	–	.471	–	–
DistMult (Yang et al., 2015)	yes	.822	.936	.914	.728	.654	.824	.733	.546
ComplEx (Trouillon et al., 2016)	yes	.941	.947	.936	.936	.692	.840	.759	.599
ANALOGY (Liu et al., 2017)	yes	.942	.947	.944	.939	.725	.854	.785	.646
Neural LP (Yang et al., 2017)	no	.940	.945	–	–	.760	.837	–	–
R-GCN (Schlichtkrull et al., 2018)	no	.819	**.964**	.929	.697	.696	.842	.760	.601
TorusE (Ebisu and Ichise, 2018)	no	.947	.954	.950	.943	.733	.832	.771	.674
ConvE (Dettmers et al., 2018)	no	.943	.956	.946	.935	.657	.831	.723	.558
HypER (Balažević et al., 2019)	no	.951	958	**.955**	.947	.790	.885	.829	.734
SimplE (Kazemi and Poole, 2018)	yes	.942	.947	.944	.939	.727	.838	.773	.660
TuckER (ours)	yes	**.953**	.958	**.955**	**.949**	**.795**	**.892**	**.833**	**.741**

Table 4: Link prediction results on WN18 and FB15k.

(see Table 6 in the Appendix A for exact hyperparameter values used) and train all three models on FB15k-237 with embedding sizes $d_e = d_r \in \{20, 50, 100, 200\}$. Figure 4 shows the obtained MRR on the test set for each model. It is important to note that at embedding dimensionalities 20, 50 and 100, TuckER has fewer parameters than ComplEx and SimplE (e.g. ComplEx and SimplE have $\sim$3 million and TuckER has $\sim$2.5 million parameters for embedding dimensionality 100).

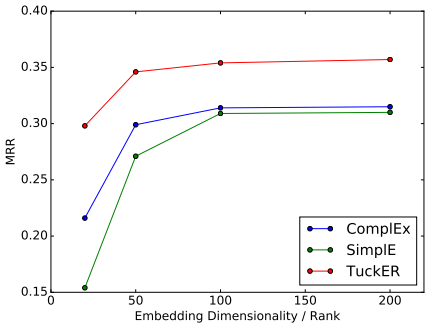


Figure 4: MRR for ComplEx, SimplE and TuckER for different embeddings sizes on FB15k-237.

We can see that the difference between the MRRs of ComplEx, SimplE and TuckER is approximately constant for embedding sizes 100 and 200. However, for lower embedding sizes, the difference between MRRs increases by 0.7% for embedding size 50 and by 4.2% for embedding size 20 for ComplEx and by 3% for embedding size 50 and by 9.9% for embedding size 20 for SimplE. At embedding size 20 ($\sim$300k parameters), the performance of TuckER is almost as good as the performance of ComplEx and SimplE at embedding size 200 ($\sim$6 million parameters), which supports our initial assumption.

7 Conclusion

In this work, we introduce TuckER, a relatively straightforward linear model for link prediction on knowledge graphs, based on the Tucker decomposition of a binary tensor of known facts. TuckER achieves state-of-the-art results on standard link prediction datasets, in part due to its ability to perform multi-task learning across relations. Whilst being fully expressive, TuckER's number of parameters grows linearly with respect to the number of entities or relations in the knowledge graph. We further show that previous linear state-of-the-art models, RESCAL, DistMult, ComplEx and SimplE, can be interpreted as special cases of our model. Future work might include exploring how to incorporate background knowledge on individual relation properties into the existing model.

Acknowledgements

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

Ivana Balažević, Carl Allen, and Timothy M Hospedales. 2019. Hypernetwork Knowledge Graph Embeddings. In *International Conference on Artificial Neural Networks*.

Hedi Ben-Younes, Rémi Cadene, Matthieu Cord, and Nicolas Thome. 2017. MUTAN: Multimodal Tucker Fusion for Visual Question Answering. In *International Conference on Computer Vision*.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems*.

Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. 2018. Go for a Walk and Arrive at the Answer: Reasoning over Paths in Knowledge Bases Using Reinforcement Learning. In *International Conference on Learning Representations*.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2D Knowledge Graph Embeddings. In *Association for the Advancement of Artificial Intelligence*.

Takuma Ebisu and Ryutaro Ichise. 2018. TorusE: Knowledge Graph Embedding on a Lie Group. In *Association for the Advancement of Artificial Intelligence*.

Jun Feng, Minlie Huang, Mingdong Wang, Mantong Zhou, Yu Hao, and Xiaoyan Zhu. 2016. Knowledge Graph Embedding by Flexible Translation. In *Principles of Knowledge Representation and Reasoning*.

Frank L Hitchcock. 1927. The Expression of a Tensor or a Polyadic as a Sum of Products. *Journal of Mathematics and Physics*, 6(1-4):164–189.

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*.

Seyed Mehran Kazemi and David Poole. 2018. SimplE Embedding for Link Prediction in Knowledge Graphs. In *Advances in Neural Information Processing Systems*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

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

Pieter M Kroonenberg and Jan De Leeuw. 1980. Principal Component Analysis of Three-Mode Data by Means of Alternating Least Squares Algorithms. *Psychometrika*, 45(1):69–97.

Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. 2018. Canonical Tensor Decomposition for Knowledge Base Completion. In *International Conference on Machine Learning*.

Hanxiao Liu, Yuexin Wu, and Yiming Yang. 2017. Analogical Inference for Multi-relational Embeddings. In *International Conference on Machine Learning*.

Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. 2016. STransE: a Novel Embedding Model of Entities and Relationships in Knowledge Bases. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A Three-Way Model for Collective Learning on Multi-Relational Data. In *International Conference on Machine Learning*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic Differentiation in PyTorch. In *NIPS-W*.

Aaron Schein, Mingyuan Zhou, David Blei, and Hanna Wallach. 2016. Bayesian Poisson Tucker Decomposition for Learning the Structure of International Relations. In *International Conference on Machine Learning*.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *European Semantic Web Conference*.

Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. 2018. M-Walk: Learning to Walk over Graphs using Monte Carlo Tree Search. In *Advances in Neural Information Processing Systems*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *International Conference on Learning Representations*.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing Text for Joint Embedding of Text and Knowledge Bases. In *Empirical Methods in Natural Language Processing*.

Théo Trouillon, Christopher R Dance, Éric Gaussier, Johannes Welbl, Sebastian Riedel, and Guillaume Bouchard. 2017. Knowledge Graph Completion via Complex Tensor Factorization. *Journal of Machine Learning Research*, 18(1):4735–4772.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In *International Conference on Machine Learning*.

Ledyard R Tucker. 1964. The Extension of Factor Analysis to Three-Dimensional Matrices. *Contributions to Mathematical Psychology*, 110119.

Ledyard R Tucker. 1966. Some Mathematical Notes on Three-Mode Factor Analysis. *Psychometrika*, 31(3):279–311.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *International Conference on Learning Representations*.

Fan Yang, Zhilin Yang, and William W Cohen. 2017. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. In *Advances in Neural Information Processing Systems*.

Yongxin Yang and Timothy Hospedales. 2017. Deep Multi-task Representation Learning: A Tensor Factorisation Approach. In *International Conference on Learning Representations*.

A Hyper-parameters

Table 5 shows best performing hyper-parameter values for TuckER across all datasets, where lr denotes learning rate, dr decay rate, ls label smoothing, and d#k, $k \in \{1, 2, 3\}$ dropout values applied on the subject entity embedding, relation matrix and subject entity embedding after it has been transformed by the relation matrix respectively.

Dataset	lr	dr	d_e	d_r	d#1	d#2	d#3	ls
FB15k	0.003	0.99	200	200	0.2	0.2	0.3	0.
FB15k-237	0.0005	1.0	200	200	0.3	0.4	0.5	0.1
WN18	0.005	0.995	200	30	0.2	0.1	0.2	0.1
WN18RR	0.01	1.0	200	30	0.2	0.2	0.3	0.1

Table 5: Best performing hyper-parameter values for TuckER across all datasets.

Table 6 shows best performing hyper-parameter values for ComplEx and SimplE on FB15k-237, used to produce the result in Figure 4.

Model	lr	dr	d_e	d_r	d#1	d#2	d#3	ls
ComplEx	0.0001	0.99	200	200	0.2	0.	0.	0.1
SimplE	0.0001	0.995	200	200	0.2	0.	0.	0.1

Table 6: Best performing hyper-parameter values for ComplEx and SimplE on FB15k-237.