# Typed Tensor Decomposition of Knowledge Bases for Relation Extraction

**Kai-Wei Chang**[†][*]   **Wen-tau Yih**[♮]    **Bishan Yang**[‡][*]   **Christopher Meek**[♮]

[†]University of Illinois, Urbana, IL 61801, USA
[‡]Cornell University, Ithaca, NY 14850, USA
[♮]Microsoft Research, Redmond, WA 98052, USA

## Abstract

While relation extraction has traditionally been viewed as a task relying solely on textual data, recent work has shown that by taking as input existing facts in the form of entity-relation triples from both knowledge bases and textual data, the performance of relation extraction can be improved significantly. Following this new paradigm, we propose a tensor decomposition approach for knowledge base embedding that is highly scalable, and is especially suitable for relation extraction. By leveraging relational domain knowledge about entity type information, our learning algorithm is significantly faster than previous approaches and is better able to discover new relations missing from the database. In addition, when applied to a relation extraction task, our approach alone is comparable to several existing systems, and improves the weighted mean average precision of a state-of-the-art method by 10 points when used as a subcomponent.

## 1 Introduction

Identifying the relationship between entities from free text, *relation extraction* is a key task for acquiring new facts to increase the coverage of a structured knowledge base. Given a pre-defined database schema, traditional relation extraction approaches focus on learning a classifier using textual data alone, such as patterns between the occurrences of two entities in documents, to determine whether the entities have a particular relation. Other than using the existing known facts to label the text corpora in a distant supervision setting (Bunescu and Mooney, 2007; Mintz et al.,

---

2009; Riedel et al., 2010; Ritter et al., 2013), an existing knowledge base is typically not involved in the process of relation extraction.

However, this paradigm has started to shift recently, as researchers showed that by taking existing facts of a knowledge base as an integral part of relation extraction, the model can leverage richer information and thus yields better performance. For instance, Riedel et al. (2013) borrowed the idea of *collective filtering* and constructed a matrix where each row is a pair of entities and each column is a particular relation. For a true entity-relation triple $(e_1, r, e_2)$, either from the text corpus or from the knowledge base, the corresponding entry in the matrix is 1. A previously unknown fact (i.e., triple) can be discovered through matrix decomposition. This approach can be viewed as creating vector representations of each relation and candidate pair of entities. Because each entity does not have its own representation, relationships of any unpaired entities cannot be discovered. Alternatively, Weston et al. (2013) created two types of embedding – one based on textual similarity and the other based on knowledge base, where the latter maps each entity and relation to the same $d$-dimensional vector space using a model proposed by Bordes et al. (2013a). They also showed that combining these two models results in a significant improvement over the model trained using only textual data.

To make such an integrated strategy work, it is important to capture all existing entities and relations, as well as the known facts, from both textual data and large databases. In this paper, we propose a new knowledge base embedding model, TRESCAL, that is highly efficient and scalable, with relation extraction as our target application. Our work is built on top of RESCAL (Nickel et al., 2011), which is a tensor decomposition method that has proven its scalability by factoring YAGO (Biega et al., 2013) with 3 million entities

and 41 million triples (Nickel et al., 2012). We improve the tensor decomposition model with two technical innovations. First, we exclude the triples that do not satisfy the relational constraints (e.g., both arguments of the relation *spouse-of* need to be *person* entities) from the loss, which is done by selecting sub-matrices of each slice of the tensor during training. Second, we introduce a mathematical technique that significantly reduces the computational complexity in both time and space when the loss function contains a regularization term. As a consequence, our method is more than four times faster than RESCAL, and is also more accurate in discovering unseen triples.

Our contributions are twofold. First, compared to other knowledge base embedding methods developed more recently, it is much more efficient to train our model. As will be seen in Sec. 5, when applied to a large knowledge base created using NELL (Carlson et al., 2010) that has 1.8M entity-relation triples, our method finishes training in 4 to 5 hours, while an alternative method (Bordes et al., 2013a) needs almost 3 days. Moreover, the prediction accuracy of our model is competitive to others, if not higher. Second, to validate its value to relation extraction, we apply TRESCAL to extracting relations from a free text corpus along with a knowledge base, using the data provided in (Riedel et al., 2013). We show that TRESCAL is complementary to existing systems and significantly improves their performance when using it as a subcomponent. For instance, this strategy improves the weighted mean average precision of the best approach in (Riedel et al., 2013) by 10 points (47% to 57%).

The remainder of this paper is organized as follows. We survey most related work in Sec. 2 and provide the technical background of our approach in Sec. 3. Our approach is detailed in Sec. 4, followed by the experimental validation in Sec. 5. Finally, Sec. 6 concludes the paper.

## 2 Related Work

Our approach of creating knowledge base embedding is based on tensor decomposition, which is a well-developed mathematical tool for data analysis. Existing tensor decomposition models can be categorized into two main families: the CP and Tucker decompositions. The CP (CAN-DECOMP/PARAFAC) decomposition (Kruskal, 1977; Kiers, 2000) approximates a tensor by a sum of rank-one tensors, while the Tucker decomposition (Tucker, 1966), also known as high-order SVD (De Lathauwer et al., 2000), factorizes a tensor into a core tensor multiplied by a matrix along each dimension. A highly scalable distributional algorithm using the Map-Reduce architecture has been proposed recently for computing CP (Kang et al., 2012), but not for the Tucker decomposition, probably due to its inherently more complicated model form.

Matrix and tensor decomposition methods have been applied to modeling multi-relational data. For instance, Speer et al. (2008) aimed to create vectors of latent components for representing concepts in a common sense knowledge base using SVD. Franz et al. (2009) proposed TripleRank to model the `subject-predicate-object` RDF triples in a tensor, and then applied the CP decomposition to identify hidden triples. Following the same tensor encoding, Nickel et al. (2011) proposed RESCAL, a restricted form of Tucker decomposition for discovering previously unknown triples in a knowledge base, and later demonstrated its scalability by applying it to YAGO, which was encoded in a 3M $\times$ 3M $\times$ 38 tensor with 41M triples (Nickel et al., 2012).

Methods that revise the objective function based on additional domain information have been proposed, such as MrWTD, a multi-relational weighted tensor decomposition method (London et al., 2013), coupled matrix and tensor factorization (Papalexakis et al., 2014), and collective matrix factorization (Singh and Gordon, 2008). Alternatively, instead of optimizing for the least-squares reconduction loss, a non-parametric Bayesian approach for 3-way tensor decomposition for modeling relational data has also been proposed (Sutskever et al., 2009). Despite the existence of a wide variety of tensor decomposition models, most methods do not scale well and have only been tested on datasets that are much smaller than the size of real-world knowledge bases.

Multi-relational data can be modeled by neural-network methods as well. For instance, Bordes et al. (2013b) proposed the Semantic Matching Energy model (SME), which aims to have the same $d$-dimensional vector representations for both entities and relations. Given the vectors of entities $e_1$, $e_2$ and relation $r$. They first learn the latent representations of $(e_1, r)$ and $(e_2, r)$. The score of $(e_1, r, e_2)$ is determined by the inner product

of the vectors of $(e_1, r)$ and $(e_2, r)$. Later, they proposed a more scalable method called translating embeddings (TransE) (Bordes et al., 2013a). While both entities and relations are still represented by vectors, the score of $(e_1, r, e_2)$ becomes the negative dissimilarity measure of the corresponding vectors $-\|\mathbf{e_i} + \mathbf{r_k} - \mathbf{e_j}\|$, motivated by the work in (Mikolov et al., 2013b; Mikolov et al., 2013a). Alternatively, Socher et al. (2013) proposed a Neural Tensor Network (NTN) that represents entities in $d$-dimensional vectors created separately by averaging pre-trained word vectors, and then learns a $d \times d \times m$ tensor describing the interactions between these latent components in each of the $m$ relations. All these methods optimize for loss functions that are more directly related to the true objective – the prediction accuracy of correct entity-relation triples, compared to the mean-squared reconstruction error in our method. Nevertheless, they typically require much longer training time.

# 3 Background

In this section, we first describe how entity-relation triples are encoded in a tensor. We then introduce the recently proposed tensor decomposition method, RESCAL (Nickel et al., 2011) and explain how it adopts an alternating least-squares method, ASALSAN (Bader et al., 2007), to compute the factorization.

## 3.1 Encoding Binary Relations in a Tensor

Suppose we are given a knowledge base with $n$ entities and $m$ relation types, and the facts in the knowledge base are denoted as a set of entity-relation triples $\mathcal{T} = \{(e_i, r_k, e_j)\}$, where $i, j \in \{1, 2, \cdots n\}$ and $k \in \{1, 2, \cdots m\}$. A triple $(e_i, r_k, e_j)$ simply means that the $i$-th entity and the $j$-th entity have the $k$-th relation. Following (Franz et al., 2009), these triples can naturally be encoded in a 3-way tensor $\mathcal{X} \in \{0, 1\}^{n \times n \times m}$, such that $\mathcal{X}_{i,j,k} = 1$ if and only if the triple $(e_i, r_k, e_j) \in \mathcal{T}^1$. The tensor can be viewed as consisting of $m$ slices, where each slice is an $n \times n$ square matrix, denoting the interactions of the entities of a particular relation type. In the remainder of this paper, we will use $\mathcal{X}_k$ to refer to the $k$-th slice of the tensor $\mathcal{X}$. Fig. 1 illustrates this representation.

---
[1] This representation can easily be extended for a probabilistic knowledge base by allowing nonnegative real values.



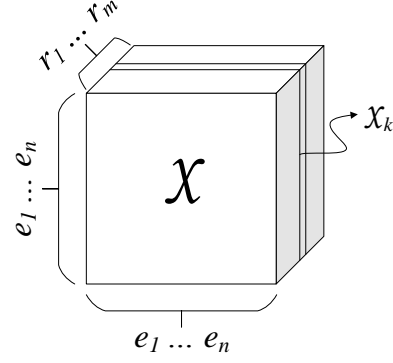Figure 1: A tensor encoding of $m$ binary relation types and $n$ entities. A slice $\mathcal{X}_k$ denotes the entities having the $k$-th relation.

## 3.2 RESCAL

In order to identify latent components in a tensor for collective learning, Nickel et al. (2011) proposed RESCAL, which is a tensor decomposition approach specifically designed for the multi-relational data described in Sec. 3.1. Given a tensor $\mathcal{X}_{n \times n \times m}$, RESCAL aims to have a rank-$r$ approximation, where each slice $\mathcal{X}_k$ is factorized as

$$\mathcal{X}_k \approx \mathbf{A} \mathcal{R}_k \mathbf{A}^T. \tag{1}$$

$\mathbf{A}$ is an $n \times r$ matrix, where the $i$-th row denotes the $r$ latent components of the $i$-th entity. $\mathcal{R}_k$ is an asymmetric $r \times r$ matrix that describes the interactions of the latent components according to the $k$-th relation. Notice that while $\mathcal{R}_k$ differs in each slice, $\mathbf{A}$ remains the same.

$\mathbf{A}$ and $\mathcal{R}_k$ are derived by minimizing the loss function below.

$$\min_{\mathbf{A}, \mathcal{R}_k} f(\mathbf{A}, \mathcal{R}_k) + \lambda \cdot g(\mathbf{A}, \mathcal{R}_k), \tag{2}$$

where $f(\mathbf{A}, \mathcal{R}_k) = \frac{1}{2} \left( \sum_k \|\mathcal{X}_k - \mathbf{A} \mathcal{R}_k \mathbf{A}^T\|_F^2 \right)$ is the mean-squared reconstruction error and $g(\mathbf{A}, \mathcal{R}_k) = \frac{1}{2} \left( \|\mathbf{A}\|_F^2 + \sum_k \|\mathcal{R}_k\|_F^2 \right)$ is the regularization term.

RESCAL is a special form of Tucker decomposition (Tucker, 1966) operating on a 3-way tensor. Its model form (Eq. (1)) can also be regarded as a relaxed form of DEDICOM (Bader et al., 2007), which derives the low-rank approximation as: $\mathcal{X}_k \approx \mathbf{A} \mathcal{D}_k \mathbf{R} \mathcal{D}_k \mathbf{A}^T$. To compare RESCAL to other tensor decomposition methods, interested readers can refer to (Kolda and Bader, 2009).

The optimization problem in Eq. (2) can be solved using the efficient alternating least-squares (ALS) method. This approach alternatively fixes $\mathcal{R}_k$ to solve for $\mathbf{A}$ and then fixes $\mathbf{A}$ to solve $\mathcal{R}_k$. The whole procedure stops until $\frac{f(\mathbf{A}, \mathcal{R}_k)}{\|\mathcal{X}\|_F^2}$ converges to some small threshold $\epsilon$ or the maximum number of iterations has been reached.

By finding the solutions where the gradients are 0, we can derive the update rules of $\mathbf{A}$ and $\mathcal{R}_k$ as below.

$$\mathbf{A} \leftarrow \left[ \sum_k \mathcal{X}_k \mathbf{A} \mathcal{R}_k^T + \mathcal{X}_k^T \mathbf{A} \mathcal{R}_k \right] \left[ \sum_k \mathcal{B}_k + \mathcal{C}_k + \lambda \mathbf{I} \right]^{-1},$$

where $\mathcal{B}_k = \mathcal{R}_k \mathbf{A}^T \mathbf{A} \mathcal{R}_k^T$ and $\mathcal{C}_k = \mathcal{R}_k^T \mathbf{A}^T \mathbf{A} \mathcal{R}_k$.

$$\mathbf{vec}(\mathcal{R}_k) \leftarrow \left( \mathbf{Z}^T \mathbf{Z} + \lambda \mathbf{I} \right)^{-1} \mathbf{Z}^T \mathbf{vec}(\mathcal{X}_k), \quad (3)$$

where $\mathbf{vec}(\mathcal{R}_k)$ is the vectorization of $\mathcal{R}_k$, $\mathbf{Z} = \mathbf{A} \otimes \mathbf{A}$ and the operator $\otimes$ is the Kronecker product.

**Complexity Analysis** Following the analysis in (Nickel et al., 2012), we assume that each $\mathcal{X}_k$ is a sparse matrix, and let $p$ be the number of non-zero entries[2]. The complexity of computing $\mathcal{X}_k \mathbf{A} \mathcal{R}_k^T$ and $\mathcal{X}_k^T \mathbf{A} \mathcal{R}_k$ is $O(pr + nr^2)$. Evaluating $\mathcal{B}_k$ and $\mathcal{C}_k$ requires $O(nr^2)$ and the matrix inversion requires $O(r^3)$. Therefore, the complexity of updating $\mathbf{A}$ is $O(pr + nr^2)$ assuming $n \gg r$. The updating rule of $R_k$ involves inverting an $r^2 \times r^2$ matrix. Therefore, directly computing the inversion requires time complexity $O(r^6)$ and space complexity $O(r^4)$. Although Nickel et al. (2012) considered using QR decomposition to simplify the updates, it is still time consuming with the time complexity $O(r^6 + pr^2)$. Therefore, the total time complexity is $O(r^6 + pr^2)$ and the step of updating $\mathcal{R}_k$ is the bottleneck in the optimization process. We will describe how to reduce the time complexity of this step to $O(nr^2 + pr)$ in Section 4.2.

## 4 Approach

We describe how we leverage the relational domain knowledge in this section. By removing the incompatible entity-relation triples from the loss

---

[2]Notice that we use a slightly different definition of $p$ from the one in (Nickel et al., 2012). The time complexity of multiplying an $n \times n$ sparse matrix $\mathcal{X}_k$ with $p$ non-zero entries by an $n \times r$ dense matrix is $O(pr)$ assuming $n \gg r$.

function, training can be done much more efficiently and results in a model with higher prediction accuracy. In addition, we also introduce a mathematical technique to reduce the computational complexity of the tensor decomposition methods when taking into account the regularization term.

### 4.1 Applying Relational Domain Knowledge

In the domain of knowledge bases, the notion of entity *types* is the side information that commonly exists and dictates whether some entities can be legitimate arguments of a given predicate. For instance, suppose the relation of interest is *born-in*, which denotes the birth *location* of a *person*. When asked whether an incompatible pair of entities, such as two *person* entities like Abraham Lincoln and John Henry, having this relation, we can immediately reject the possibility. Although the type information and the constraints are readily available, it is overlooked in the previous work on matrix and tensor decomposition models for knowledge bases (Riedel et al., 2013; Nickel et al., 2012). Ignoring the type information has two implications. Incompatible entity-relation triples still participate in the loss function of the optimization problem, which incurs unnecessary computation. Moreover, by choosing values for these incompatible entries we introduce errors in training the model that can reduce the quality of the model.

Based on this observation, we propose *Typed-RESCAL*, or TRESCAL, which leverages the entity type information to improve both the efficiency of model training and the quality of the model in term of prediction accuracy. We employ a direct and simple approach by excluding the triples of the incompatible entity types from the loss in Eq. (2). For each relation, let $\mathbb{L}_k$ and $\mathbb{R}_k$ be the set of entities with a compatible type to the $k$-th relation. That is, $(e_i, r_k, e_j)$ is a feasible triple if and only if $e_i \in \mathbb{L}_k$ and $e_j \in \mathbb{R}_k$. For notational convenience, we use $\mathbf{A}_{k_l}, \mathbf{A}_{k_r}$ to denote the sub-matrices of $\mathbf{A}$ that consists of rows associated with $\mathbb{L}_k$ and $\mathbb{R}_k$, respectively. Analogously, let $\mathcal{X}_{k_{lr}}$ be the sub-matrix of $\mathcal{X}_k$ that consists of only the entity pairs compatible to the $k$-th relation. The rows and columns of $\mathcal{X}_{k_{lr}}$ map to the entities in $\mathbf{A}_{k_l}$ and $\mathbf{A}_{k_r}$, respectively. In other words, entries of $\mathcal{X}_k$ but not in $\mathcal{X}_{k_{lr}}$ do not satisfy the type constraint and are ignored from the computation.
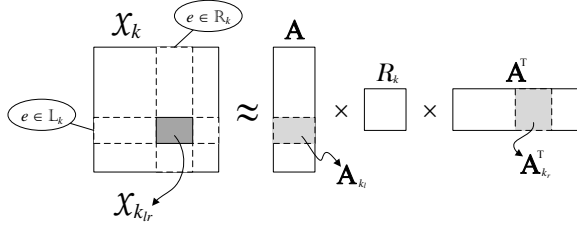
Figure 2: The construction of TRESCAL. Suppose the $k$-th relation is *born-in*. $\mathbb{L}_k$ is then a set of *person* entities and $\mathbb{R}_k$ is a set of *location* entities. Only the sub-matrix corresponds to the compatible entity pairs (i.e., $\mathcal{X}_{k_{lr}}$) and the sub-matrices of the associated entities (i.e., $\mathbf{A}_{k_l}$ and $\mathbf{A}_{k_r}^T$) will be included in the loss.

Fig. 2 illustrates this construction.

TRESCAL solves the following optimization problem:

$$\min_{\mathbf{A}, \mathcal{R}_k} f'(\mathbf{A}, \mathcal{R}_k) + \lambda \cdot g(\mathbf{A}, \mathcal{R}_k), \qquad (4)$$

where $f'(\mathbf{A}, \mathcal{R}_k) = \frac{1}{2} \sum_k \|\mathcal{X}_{k_{lr}} - \mathbf{A}_{k_l} \mathcal{R}_k \mathbf{A}_{k_r}^T\|_F^2$ and $g(\mathbf{A}, \mathcal{R}_k) = \frac{1}{2} \left( \|\mathbf{A}\|_F^2 + \sum_k \|\mathcal{R}_k\|_F^2 \right)$.

Similarly, $\mathbf{A}$ and $\mathcal{R}_k$ can be solved using the alternating least-squares method. The update rule of $\mathbf{A}$ is

$$\mathbf{A} \leftarrow \left[ \sum_k \left( \mathcal{X}_{k_{lr}} \mathbf{A}_{k_r} \mathcal{R}_k^T + \mathcal{X}_{k_{lr}}^T \mathbf{A}_{k_l} \mathcal{R}_k \right) \right] \times$$
$$\left[ \sum_k \mathcal{B}_{k_r} + \mathcal{C}_{k_l} + \lambda \mathbf{I} \right]^{-1},$$

where $\mathcal{B}_{k_r} = \mathcal{R}_k \mathbf{A}_{k_r}^T \mathbf{A}_{k_r} \mathcal{R}_k^T$ and $\mathcal{C}_{k_l} = \mathcal{R}_k^T \mathbf{A}_{k_l}^T \mathbf{A}_{k_l} \mathcal{R}_k$.

The update of $\mathcal{R}_k$ becomes:

$$\mathbf{vec}(\mathcal{R}_k) \leftarrow \left( \mathbf{A}_{k_r}^T \mathbf{A}_{k_r} \otimes \mathbf{A}_{k_l}^T \mathbf{A}_{k_l} + \lambda \mathbf{I} \right)^{-1} \times$$
$$\mathbf{vec}(\mathbf{A}_{k_l}^T \mathcal{X}_{k_{lr}} \mathbf{A}_{k_r}),$$
$$(5)$$

**Complexity Analysis** Let $\bar{n}$ be the average number of entities with a compatible type to a relation. Follow a similar derivation in Sec. 3.2, the time complexity of updating $\mathbf{A}$ is $O(pr + \bar{n}r^2)$ and the time complexity of updating $\mathcal{R}_k$ remains to be $O(r^6 + pr^2)$.

## 4.2 Handling Regularization Efficiently

Examining the update rules of both RESCAL and TRESCAL, we can see that the most time-consuming part is the matrix inversions. For RESCAL, this is the term $(\mathbf{Z}^T \mathbf{Z} + \lambda \mathbf{I})^{-1}$ in Eq. (3), where $\mathbf{Z} = \mathbf{A} \otimes \mathbf{A}$. Nickel et al. (2011) made the observation that if $\lambda = 0$, the matrix inversion can be calculated by

$$(\mathbf{Z}^T \mathbf{Z})^{-1} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A} \otimes (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}.$$

Then, it only involves an inversion of an $r \times r$ matrix, namely $\mathbf{A}^T \mathbf{A}$. However, if $\lambda > 0$, directly calculating Eq. (3) requires to invert an $r^2 \times r^2$ matrix and thus becomes a bottleneck in solving Eq. (2).

To reduce the computational complexity of the update rules of $\mathcal{R}_k$, we compute the inversion $\left( \mathbf{Z}^T \mathbf{Z} + \lambda \mathbf{I} \right)^{-1}$ by applying singular value decomposition (SVD) to $\mathbf{A}$, such that $\mathbf{A} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T$, where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices and $\mathbf{\Sigma}$ is a diagonal matrix. Then by using properties of the Kronecker product we have:

$$\left( \mathbf{Z}^T \mathbf{Z} + \lambda \mathbf{I} \right)^{-1}$$
$$= \left( \lambda \mathbf{I} + \mathbf{V} \mathbf{\Sigma}^2 \mathbf{V}^T \otimes \mathbf{V} \mathbf{\Sigma}^2 \mathbf{V}^T \right)^{-1}$$
$$= \left( \lambda \mathbf{I} + (\mathbf{V} \otimes \mathbf{V})(\mathbf{\Sigma}^2 \otimes \mathbf{\Sigma}^2)(\mathbf{V} \otimes \mathbf{V})^T \right)^{-1}$$
$$= (\mathbf{V} \otimes \mathbf{V}) \left( \lambda \mathbf{I} + \mathbf{\Sigma}^2 \otimes \mathbf{\Sigma}^2 \right)^{-1} (\mathbf{V} \otimes \mathbf{V})^T.$$

The last equality holds because $\mathbf{V} \otimes \mathbf{V}$ is also an orthogonal matrix. We leave the detailed derivations in Appendix A. Notice that $\left( \lambda \mathbf{I} + \mathbf{\Sigma}^2 \otimes \mathbf{\Sigma}^2 \right)^{-1}$ is a diagonal matrix. Therefore, the inversion calculation is trivial.

This technique can be applied to TRESCAL as well. By applying SVD to both $\mathbf{A}_{k_l}$ and $\mathbf{A}_{k_r}$, we have $\mathbf{A}_{k_l} = \mathbf{U}_{k_l} \mathbf{\Sigma}_{k_l} \mathbf{V}_{k_l}^T$ and $\mathbf{A}_{k_r} = \mathbf{U}_{k_r} \mathbf{\Sigma}_{k_r} \mathbf{V}_{k_r}^T$, respectively. The computation of $\left( \mathbf{A}_{k_r}^T \mathbf{A}_{k_r} \otimes \mathbf{A}_{k_l}^T \mathbf{A}_{k_l} + \lambda \mathbf{I} \right)^{-1}$ of Eq. (5) thus becomes:

$$(\mathbf{V}_{k_l} \otimes \mathbf{V}_{k_r}) \left( \lambda \mathbf{I} + \mathbf{\Sigma}_{k_l}^2 \otimes \mathbf{\Sigma}_{k_r}^2 \right)^{-1} (\mathbf{V}_{k_l} \otimes \mathbf{V}_{k_r})^T.$$

The procedure of updating $\mathcal{R}$ is depicted in Algorithm 1.

**Complexity Analysis** For RESCAL, $\mathbf{V}$ and $\mathbf{\Sigma}$ can be computed by finding eigenvectors of $\mathbf{A}^T \mathbf{A}$. Therefore, computing SVD of $\mathbf{A}$ costs $O(nr^2 + r^3) = O(nr^2)$. Computing Step 4 in Algorithm 1 takes $O(nr^2 + pr)$. Step 5 and Step 6 require

**Algorithm 1** Updating $\mathcal{R}$ in TRESCAL

**Require:** $\mathcal{X}$, $\mathbf{A}$, and entity sets $\mathbb{R}_k, \mathbb{L}_k, \forall k$
**Ensure:** $\mathcal{R}_k, \forall k.$

1: **for** $k = 1 \dots m$ **do**
2: $\quad [\mathbf{U}_{k_l}, \boldsymbol{\Sigma}_{k_l}^2, \mathbf{V}_{k_l}] \leftarrow \text{SVD}(\mathbf{A}_{k_l}^T \mathbf{A}_{k_l}).$
3: $\quad [\mathbf{U}_{k_r}, \boldsymbol{\Sigma}_{k_r}^2, \mathbf{V}_{k_r}] \leftarrow \text{SVD}(\mathbf{A}_{k_r}^T \mathbf{A}_{k_r}).$
4: $\quad \mathbf{M}_1 \leftarrow \mathbf{V}_{k_l}^T \mathbf{A}_{k_l}^T \mathcal{X}_{k_{lr}} \mathbf{A}_{k_r} \mathbf{V}_{k_r}.$
5: $\quad \mathbf{M}_2 \leftarrow \text{diag}(\boldsymbol{\Sigma}_{k_l}^2)\,\text{diag}(\boldsymbol{\Sigma}_{k_r}^2)^T + \lambda\mathbf{1}.$
    ($\mathbf{1}$ is a matrix of all ones. Function $\text{diag}$ converts the diagonal entries of a matrix to a vector. )
6: $\quad \mathcal{R}_k \leftarrow \mathbf{V}_{k_l}(M_1./M_2)\mathbf{V}_{k_r}^T.$
    (The operator "./" is element-wise division.)
7: **end for**

---

$O(r^2)$ and $O(r^3)$, respectively. The overall time complexity of updating $\mathcal{R}_k$ becomes $O(nr^2 + pr)$.

Using a similar derivation, the time complexity of updating $\mathcal{R}_k$ in TRESCAL is $O(\bar{n}r^2 + pr)$. Therefore, the total complexity of each iteration is $O(\bar{n}r^2 + pr)$.

## 5 Experiments

We conduct two sets of experiments. The first evaluates the proposed TRESCAL algorithm on inferring unknown facts using existing relation–entity triples, while the second demonstrates its application to relation extraction when a text corpus is available.

### 5.1 Knowledge Base Completion

We evaluate our approach on a knowledge base generated by the CMU Never Ending Language Learning (NELL) project (Carlson et al., 2010). NELL collects human knowledge from the web and has generated millions of entity-relation triples. We use the data generated from version 165 for training[3], and collect the new triples generated between NELL versions 166 and 533 as the development set and those generated between version 534 and 745 as the test set[4]. The data statistics of the training set are summarized in Table 1. The numbers of triples in the development and test sets are 19,665 and 117,889, respectively. Notice that this dataset is substantially larger than the datasets used in recent work. For example, the Freebase data used in (Socher et al., 2013) and (Bordes et

[3] http://www.cs.cmu.edu/~nlao/
[4] http://bit.ly/trescal

| NELL | |
|---|---|
| # entities | 753k |
| # relation types | 229 |
| # entity types | 300 |
| # entity-relation triples | 1.8M |

Table 1: Data statistics of the training set from NELL in our experiments.

al., 2013a) have 316k and 483k[5] triples, respectively, compared to 1.8M in this dataset.

In the NELL dataset, the entity type information is encoded in a specific relation, called *Generalization*. Each entity in the knowledge base is assigned to at least one category presented by the Generalization relationship. Based on this information, the compatible entity type constraint of each relation can be easily identified. Specifically, we examined the entities and relations that occur in the triples of the training data, and counted all the types appearing in these instances of a given relation legitimate.

We implement RESCAL and TRESCAL in MATLAB with the Matlab tensor Toolbox (Bader et al., 2012). With the efficient implementation described in Section 4.2, all experiments can be conducted on a commodity PC with 16 GB memory. We set the maximal number of iterations of both RESCAL and TRESCAL to be 10, which we found empirically to be enough to generate a stable model. Note that Eq. (4) is non-convex, and the optimization process does not guarantee to converge to a global minimum. Therefore, initializing the model properly might be important for the performance. Following the implementation of RESCAL, we initialize $\mathbf{A}$ by performing singular value decomposition over $\bar{\mathcal{X}} = \sum_k (\mathcal{X}_k + \mathcal{X}_k^T)$, such that $\bar{\mathcal{X}} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$ and set $\mathbf{A} = \mathbf{U}$. Then, we apply the update rule of $\mathcal{R}_k$ to initialize $\{\mathcal{R}_k\}$. RESCAL and TRESCAL have two types of parameters: (1) the rank $r$ of the decomposed tensor and (2) the regularization parameter $\lambda$. We tune the rank parameter on development set in a range of $\{100, 200, 300, 400\}$ and the regularization parameter in a range of $\{0.01, 0.05, 0.1, 0.5, 1\}$.

For comparison, we also use the code released by Bordes et al. (2013a), which is implemented using Python and the Theano library (Bergstra et al., 2010), to train a TransE model using the

[5] In (Bordes et al., 2013a), there is a much larger dataset, FB1M, that has 17.5M triples used for evaluation. However, this dataset has not been released.

| | Entity Retrieval | | | Relation Retrieval | | |
|---|---|---|---|---|---|---|
| | TransE | RESCAL | TRESCAL | TransE | RESCAL | TRESCAL |
| w/o type checking | 51.41%‡ | 51.59% | 54.79% | 75.88% | 73.15%† | 76.12% |
| w/ type checking | 67.56% | 62.91%‡ | 69.26% | 70.71%‡ | 73.08%† | 75.70% |

Table 2: Model performance in mean average precision (MAP) on *entity retrieval* and *relation retrieval*. † and ‡ indicate the comparison to TRESCAL in the same setting is statistically significant using a paired-t test on average precision of each query, with $p < 0.01$ and $p < 0.05$, respectively. Enforcing type constraints during test time improves entity retrieval substantially, but does not help in relation retrieval.

same NELL dataset. We reserved randomly 1% of the training triples for the code to evaluate the model performance in each iteration. As suggested in their paper, we experiment with several hyper-parameters, including learning rate of {0.01, 0.001}, the latent dimension of {50, 100} and the similarity measure of {L1, L2}. In addition, we also adjust the number of batches of {50, 100, 1000}. Of all the configurations, we keep the models picked by the method, as well as the final model after 500 training iterations. The final model is chosen by the performance on our development set.

### 5.1.1 Training Time Reduction

We first present experimental results demonstrating that TRESCAL indeed reduces the time required to factorize a knowledge database, compared to RESCAL. The experiment is conducted on NELL with $r = 300$ and $\lambda = 0.1$. When $\lambda \neq 0$, the original RESCAL algorithm described in (Nickel et al., 2011; Nickel et al., 2012) cannot handle a large $r$, because updating matrices $\{\mathcal{R}_k\}$ requires $O(r^4)$ memory. Later in this section, we will show that in some situation a large rank $r$ is necessary for achieving good testing performance.

Comparing TRESCAL with RESCAL, each iteration of TRESCAL takes 1,608 seconds, while that of RESCAL takes 7,415 seconds. In other words, by inducing the entity type information and constraints, TRESCAL enjoys around 4.6 times speed-up, compared to an improved regularized version of RESCAL. When updating $\mathbf{A}$ and $\{\mathcal{R}_k\}$ TRESCAL only requires operating on sub-matrices of $\mathbf{A}$, $\{\mathcal{R}_k\}$ and $\{\mathcal{X}_k\}$, which reduces the computation substantially. In average, TRESCAL filters 96% of entity triples that have incompatible types.

In contrast, it takes TransE at least 2 days and 19 hours to finish training the model (the default 500 iterations)[6], while TRESCAL finishes the training

in roughly 4 to 5 hours[7].

### 5.1.2 Test Performance Improvement

We consider two different types of tasks to evaluate the prediction accuracy of different models – *entity retrieval* and *relation retrieval*.

**Entity Retrieval** In the first task, we collect a set of entity-relation pairs $\{(e_i, r_k)\}$ and aim at predicting $e_j$ such that the tuple $(e_i, r_k, e_j)$ is a recorded triple in the NELL knowledge base. For each pair $(e_i, r_k)$, we collect triples $\{(e_i, r_k, e_j^*)\}$ from the NELL test corpus as positive samples and randomly pick 100 entries $e_j'$ to form negative samples $\{e_i, r_k, e_j'\}$. Given $\mathbf{A}$ and $\mathcal{R}_k$ from the factorization generated by RESCAL or TRESCAL, the score assigned to a triple $\{e_i, r_k, e_j'\}$ is computed by $a_i^T R_k a_j$ where $a_i$ and $a_j$ are the $i$-th and $j$-th rows of $\mathbf{A}$. In TransE, the score is determined by the negative dissimilarity measures of the learned embeddings: $-d(e_i, r_k, e_j') = -\|\mathbf{e_i} + \mathbf{r_k} - \mathbf{e_j'}\|_2^2$.

We evaluate the performance using mean average precision (MAP), which is a robust and stable metric (Manning et al., 2008). As can be observed in Table 2 (left), TRESCAL achieves 54.79%, which outperforms 51.59% of RESCAL and 51.41% of TransE. Adding constraints during test time by assigning the lowest score to the entity triples with incompatible types improves results of all models – TRESCAL still performs the best (69.26%), compared to TransE (67.56%) and RESCAL (62.91%).

**Relation Retrieval** In the second task, given a relation type $r_k$, we are looking for the entity pairs $(e_i, e_j)$ that have this specific relationship. To generate test data, for each relation type, we collect

_____

[6]It took almost 4 days to train the best TransE model that

is included in Table 2.

[7]We also tested the released code from (Socher et al., 2013) for training a neural tensor network model. However, we are not able to finish the experiments as each iteration of this method takes almost 5 hours.

gold entity pairs from the NELL knowledge base as positive samples and randomly pick a set of entity pairs as negative samples such that the number of positive samples are the same as negative ones.

Results presented in Table 2 (right) show that TRESCAL achieves 76.12%, while RESCAL and TransE are 73.15% and 75.88%, respectively. Therefore, incorporating the type information in training seems to help in this task as well. Enforcing the type constraints during test time does not help as in *entity retrieval*. By removing incompatible entity pairs, the performance of TRESCAL, RESCAL and TransE drop slightly to 75.70%, 73.08% and 70.71% respectively. One possible explanation is that the task of *relation retrieval* is easier than *entity retrieval*. The incorrect type information of some entities ends up filtering out a small number of entity pairs that were retrieved correctly by the model.

Notice that TRESCAL achieves different levels of performance on various relations. For example, it performs well on predicting *AthletePlaysSport* (81%) and *CoachesInLeague* (88%), but achieves suboptimal performance on predicting *Works-For* (49%) and *BuildingLocatedInCity* (35%). We hypothesize that it is easier to generalize entity-relation triples when the relation has several related relations. For examples, *AthletePlaysForTeam* and *TeamPlaysSport* may help discover entity-relation triples of AthletePlaysSport.

### 5.1.3 Sensitivity to Parameters

We also study if TRESCAL is sensitive to the rank parameter $r$ and the regularization parameter $\lambda$, where the detailed results can be found in Appendix B. In short, we found that increasing the rank $r$ generally leads to better models. Also, while the model is not very sensitive to the value of the regularization parameter $\lambda$, tuning $\lambda$ is still necessary for achieving the best performance.

### 5.2 Relation Extraction

Next, we apply TRESCAL to the task of extracting relations between entities, jointly from a text corpus and a structured knowledge base. We use a corpus from (Riedel et al., 2013) that is created by aligning the entities in NYTimes and Freebase. The corpus consists of a training set and a test set. In the training set, a list of entity pairs are provided, along with surface patterns extracted from NYTimes and known relations obtained from Freebase. In the test set, only the surface patterns are given. By jointly factoring a matrix consisting of the surface patterns and relations, Riedel et al. (2013) show that their model is able to capture the mapping between the surface patterns and the structured relations and hence is able to extract the entity relations from free text. In the following, we show that TRESCAL can be applied to this task.

We focus on the 19 relations listed in Table 1 of (Riedel et al., 2013) and only consider the surface patterns that co-occur with these 19 relations. We prune the surface patterns that occur less than 5 times and remove the entities that are not involved in any relation and surface pattern. Based on the training and test sets, we build a 80,698×80,698×1,652 tensor, where each slice captures a particular structured relation or a surface pattern between two entities. There are 72 fine types extracted from Freebase assigned to 53,836 entities that are recorded in Freebase. In addition, special types, PER, LOC, ORG and MISC, are assigned to the remaining 26,862 entities based on the predicted NER tags provided by the corpus. A type is considered incompatible to a relation or a surface pattern if in the training data, none of the argument entities of the relation belongs to the type. We use $r = 400$ and $\lambda = 0.1$ in TRESCAL to factorize the tensor.

We compare the proposed TRESCAL model to RI13 (Riedel et al., 2013), YA11 (Yao et al., 2011), MI09 (Mintz et al., 2009) and SU12 (Surdeanu et al., 2012)[8]. We follow the protocol used in (Riedel et al., 2013) to evaluate the results. Given a relation as query, the top 1,000 entity pairs output by each system are collected and the top 100 ones are judged manually. Besides comparing individual models, we also report the results of combined models. To combine the scores from two models, we simply normalize the scores of entity-relation tuples to zero mean and unit variance and take the average. The results are summarized in Table 3.

As can been seen in the table, using TRESCAL alone is not very effective and its performance is only compatible to MI09 and YA11, and is significantly inferior to RI13. This is understandable because the problem setting favors RI13 as only entity pairs that have occurred in the text or the database will be considered in RI13, both during model training and testing. In contrast, TRESCAL

---

[8] The corpus and the system outputs are from http://www.riedelcastro.org/uschema

| Relation | # | MI09 | YA11 | SU12 | RI13 | TR | TR+SU12 | TR+RI13 |
|---|---|---|---|---|---|---|---|---|
| person/company | 171 | 0.41 | 0.40 | 0.43 | 0.49 | 0.43 | 0.53 | **0.64** |
| location/containedby | 90 | 0.39 | 0.43 | 0.44 | 0.56 | 0.23 | 0.46 | **0.58** |
| parent/child | 47 | 0.05 | 0.10 | 0.25 | 0.31 | 0.19 | 0.24 | **0.35** |
| person/place_of_birth | 43 | 0.32 | 0.31 | 0.34 | 0.37 | 0.50 | 0.61 | **0.66** |
| person/nationality | 38 | 0.10 | **0.30** | 0.09 | 0.16 | 0.13 | 0.16 | 0.22 |
| author/works_written | 28 | 0.52 | 0.53 | 0.54 | **0.71** | 0.00 | 0.39 | 0.62 |
| person/place_of_death | 26 | 0.58 | 0.58 | 0.63 | 0.63 | 0.54 | 0.72 | **0.89** |
| neighborhood/neighborhood_of | 13 | 0.00 | 0.00 | 0.08 | 0.67 | 0.08 | 0.13 | **0.73** |
| person/parents | 8 | 0.21 | 0.24 | **0.51** | 0.34 | 0.01 | 0.16 | 0.38 |
| company/founders | 7 | 0.14 | 0.14 | 0.30 | 0.39 | 0.06 | 0.17 | **0.44** |
| film/directed_by | 4 | 0.06 | 0.15 | 0.25 | 0.30 | 0.03 | 0.13 | **0.35** |
| sports_team/league | 4 | 0.00 | 0.43 | 0.18 | 0.63 | 0.50 | 0.29 | **0.63** |
| team/arena_stadium | 3 | 0.00 | 0.06 | 0.06 | 0.08 | 0.00 | 0.04 | **0.09** |
| team_owner/teams_owned | 2 | 0.00 | 0.50 | 0.70 | *0.75* | 0.00 | 0.00 | *0.75* |
| roadcast/area_served | 2 | *1.00* | 0.50 | *1.00* | *1.00* | 0.50 | 0.83 | *1.00* |
| structure/architect | 2 | 0.00 | 0.00 | *1.00* | *1.00* | 0.00 | 0.02 | *1.00* |
| composer/compositions | 2 | 0.00 | 0.00 | 0.00 | **0.12** | 0.00 | 0.00 | 0.12 |
| person/religion | 1 | 0.00 | *1.00* | *1.00* | *1.00* | 0.00 | *1.00* | *1.00* |
| film/produced_by | 1 | *1.00* | *1.00* | *1.00* | 0.33 | 0.00 | *1.00* | 0.25 |
| Weighted MAP | | 0.33 | 0.36 | 0.39 | 0.47 | 0.30 | 0.44 | **0.57** |

Table 3: Weighted Mean Average Precisions. The # column shows the number of true facts in the pool. Bold faced are winners per relation, italics indicate ties based on a sign test.

predicts all the possible combinations between entities and relations, which makes the model less fit to the task. However, when combining TRESCAL with a pure text-based method, such as SU12, we can clearly see TRESCAL is complementary to SU12 (0.39 to 0.44 in weighted MAP score), which makes the results competitive to RI13.

Interestingly, although both TRESCAL and RI13 leverage information from the knowledge base, we find that by combining them, the performance is improved quite substantially (0.47 to 0.57). We suspect that the reason is that in our construction, each entity has its own vector representation, which is lacked in RI13. As a result, the new triples that TRESCAL finds are very different from those found by RI13. Nevertheless, combining more methods do not always yield an improvement. For example, combining TR, RI13 and SU12 together (not included in Table 3) achieves almost the same performance as TR+RI13.

## 6 Conclusions

In this paper we developed TRESCAL, a tensor decomposition method that leverages relational domain knowledge. We use relational domain knowledge to capture which triples are potentially valid and found that, by excluding the triples that are incompatible when performing tensor decomposition, we can significantly reduce the training time and improve the prediction performance as compared with RESCAL and TransE. More-over, we demonstrated its effectiveness in the application of relation extraction. Evaluated on the dataset provided in (Riedel et al., 2013), the performance of TRESCAL alone is comparable to several existing systems that leverage the idea of distant supervision. When combined with the state-of-the-art systems, we found that the results can be further improved. For instance, the weighted mean average precision of the previous best approach in (Riedel et al., 2013) has been increased by 10 points (47% to 57%).

There are a number of interesting potential extensions of our work. First, while the experiments in this paper are on traditional knowledge bases and textual data, the idea of leveraging relational domain knowledge is likely to be of value to other linguistic databases as well. For instance, part-of-speech tags can be viewed as the "types" of words. Incorporating such information in other tensor decomposition methods (e.g., (Chang et al., 2013)) may help lexical semantic representations. Second, relational domain knowledge goes beyond entity types and their compatibility with specific relations. For instance, the entity-relation triple $(e_1, \text{child-of}, e_2)$ can be *valid* only if $e_1.\text{type} = \text{person} \land e_2.\text{type} = \text{person} \land e_1.\text{age} < e_2.\text{age}$. It would be interesting to explore the possibility of developing efficient methods to leverage other types of relational domain knowledge. Finally, we would like to create more sophisticated models of knowledge base embedding, targeting complex in-

ference tasks to better support semantic parsing and question answering.

## Acknowledgments

## A Detailed Derivation

We first introduce some lemmas that will be useful for our derivation. Lemmas 2, 3 and 4 are the basic properties of the Kronecker product. Their proofs can be found at (Laub, 2005).

**Lemma 1.** *Let* $\mathbf{V}$ *be an orthogonal matrix and* $\mathbf{\Sigma}$ *a diagonal matrix. Then* $(\mathbf{I} + \mathbf{V}\mathbf{\Sigma}\mathbf{V}^T)^{-1} = \mathbf{V}(\mathbf{I} + \mathbf{\Sigma})^{-1}\mathbf{V}^T$.

**Proof.**

$$(\mathbf{I} + \mathbf{V}\mathbf{\Sigma}\mathbf{V}^T)^{-1} = (\mathbf{V}\mathbf{I}\mathbf{V}^T + \mathbf{V}\mathbf{\Sigma}\mathbf{V}^T)^{-1}$$
$$= \mathbf{V}(\mathbf{I} + \mathbf{\Sigma})^{-1}\mathbf{V}^T \quad \square$$

**Lemma 2.** $(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = \mathbf{A}\mathbf{C} \otimes \mathbf{B}\mathbf{D}$.

**Lemma 3.** $(\mathbf{A} \otimes \mathbf{B})^T = \mathbf{A}^T \otimes \mathbf{B}^T$.

**Lemma 4.** *If* $\mathbf{A}$ *and* $\mathbf{B}$ *are orthogonal matrices, then* $\mathbf{A} \otimes \mathbf{B}$ *will also be an orthogonal matrix.*

Let $\mathbf{Z} = \mathbf{A} \otimes \mathbf{A}$ and apply singular value decomposition to $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$. The term $(\mathbf{Z}^T\mathbf{Z} + \lambda\mathbf{I})^{-1}$ can be rewritten as:

$$(\mathbf{Z}^T\mathbf{Z} + \lambda\mathbf{I})^{-1}$$
$$= (\lambda\mathbf{I} + (\mathbf{A}^T \otimes \mathbf{A}^T)(\mathbf{A} \otimes \mathbf{A}))^{-1} \quad (6)$$
$$= (\lambda\mathbf{I} + \mathbf{A}^T\mathbf{A} \otimes \mathbf{A}^T\mathbf{A})^{-1} \quad (7)$$
$$= (\lambda\mathbf{I} + \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T \otimes \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T)^{-1} \quad (8)$$
$$= (\lambda\mathbf{I} + (\mathbf{V} \otimes \mathbf{V})(\mathbf{\Sigma}^2 \otimes \mathbf{\Sigma}^2)(\mathbf{V} \otimes \mathbf{V})^T)^{-1} \quad (9)$$
$$= (\mathbf{V} \otimes \mathbf{V})(\lambda\mathbf{I} + \mathbf{\Sigma}^2 \otimes \mathbf{\Sigma}^2)^{-1}(\mathbf{V} \otimes \mathbf{V})^T \quad (10)$$

Eq. (6) is from replacing $\mathbf{Z}$ with $\mathbf{A} \otimes \mathbf{A}$ and Lemma 3. Eq. (7) is from Lemma 2. Eq. (8) is from the properties of SVD, where $\mathbf{U}$ and $\mathbf{V}$ are orthonormal matrices. Eq. (9) is from Lemma 2 and Lemma 3. Finally, Eq. (10) comes from Lemma 1.

## B Sensitivity to Hyper-parameters in TRESCAL

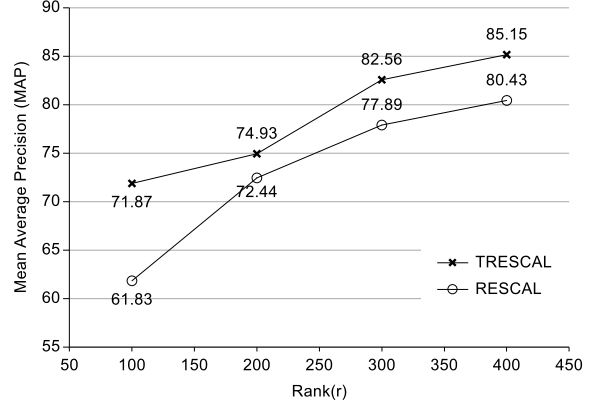We study if TRESCAL is sensitive to the rank parameter $r$ and the regularization parameter $\lambda$.



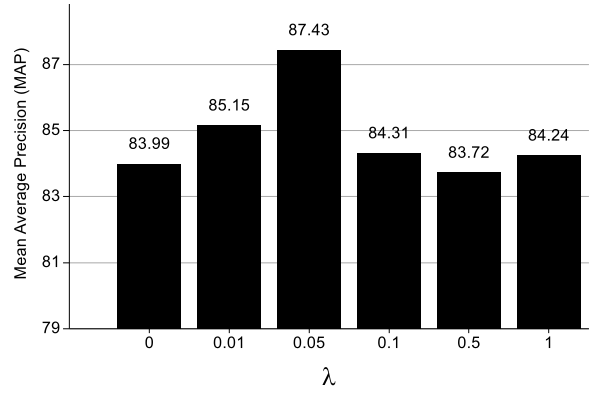Figure 3: Prediction performance of TRESCAL and RESCAL with different rank ($r$).



Figure 4: Prediction performance of TRESCAL with different regularization parameter ($\lambda$).

We use the task of *relation retrieval* and present the model performance on the development set. Fig. 3 shows the performance of TRESCAL and RESCAL with different rank ($r$) values while fixing $\lambda = 0.01$. Results show that both TRESCAL and RESCAL achieve better performance when $r$ is reasonably large. TRESCAL obtains a better model with smaller $r$ than RESCAL, because TRESCAL only needs to fit the triples of the compatible entity types. Therefore, it allows to use smaller number of latent variables to fit the training data.

Fixing $r = 400$, Fig. 4 shows the performance of TRESCAL at different values of the regularization parameter $\lambda$, including no regularization at all ($\lambda = 0$). While the results suggest that the method is not very sensitive to $\lambda$, tuning $\lambda$ is still necessary for achieving the best performance.

# References

Brett W Bader, Richard A Harshman, and Tamara G Kolda. 2007. Temporal analysis of semantic graphs using ASALSAN. In *ICDM*, pages 33–42. IEEE.

Brett W. Bader, Tamara G. Kolda, et al. 2012. Matlab tensor toolbox version 2.5. Available online, January.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June. Oral Presentation.

Joanna Biega, Erdal Kuzey, and Fabian M Suchanek. 2013. Inside YOGO2s: a transparent information extraction architecture. In *WWW*, pages 325–328.

A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. 2013a. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems 26*.

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. 2013b. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, pages 1–27.

Razvan Bunescu and Raymond Mooney. 2007. Learning to extract relations from the web using minimal supervision. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 576–583, Prague, Czech Republic, June. Association for Computational Linguistics.

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

Kai-Wei Chang, Wen-tau Yih, and Christopher Meek. 2013. Multi-relational latent semantic analysis. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1602–1612, Seattle, Washington, USA, October. Association for Computational Linguistics.

Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. 2000. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278.

Thomas Franz, Antje Schultz, Sergej Sizov, and Steffen Staab. 2009. Triplerank: Ranking semantic web data by tensor decomposition. In *The Semantic Web-ISWC 2009*, pages 213–228. Springer.

U Kang, Evangelos Papalexakis, Abhay Harpale, and Christos Faloutsos. 2012. Gigatensor: scaling tensor analysis up by 100 times-algorithms and discoveries. In *KDD*, pages 316–324. ACM.

Henk AL Kiers. 2000. Towards a standardized notation and terminology in multiway analysis. *Journal of chemometrics*, 14(3):105–122.

Tamara G. Kolda and Brett W. Bader. 2009. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, September.

Joseph B Kruskal. 1977. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear algebra and its applications*, 18(2):95–138.

Alan J Laub, 2005. *Matrix analysis for scientists and engineers*, chapter 13, pages 139–150. SIAM.

Ben London, Theodoros Rekatsinas, Bert Huang, and Lise Getoor. 2013. Multi-relational learning using weighted tensor decomposition with modular loss. Technical report, University of Maryland College Park. http://arxiv.org/abs/1303.1733.

C. Manning, P. Raghavan, and H. Schutze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. 2013a. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013b. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, June. Association for Computational Linguistics.

Mike Mintz, Steven Bills, Rion Snow, and Daniel Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1003–1011, Suntec, Singapore, August. Association for Computational Linguistics.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *ICML*, pages 809–816.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2012. Factorizing YAGO: scalable machine learning for linked data. In *WWW*, pages 271–280.

Evangelos E Papalexakis, Tom M Mitchell, Nicholas D Sidiropoulos, Christos Faloutsos, Partha Pratim Talukdar, and Brian Murphy. 2014. Turbo-smt: Accelerating coupled sparse matrix-tensor factorizations by 200x. In *SDM*.

Sebastian Riedel, Limin Yao, and Andrew McCallum. 2010. Modeling relations and their mentions without labeled text. In *Proceedings of ECML/PKDD 2010*. Springer.

Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. 2013. Relation extraction with matrix factorization and universal schemas. In *NAACL*, pages 74–84.

Alan Ritter, Luke Zettlemoyer, Mausam, and Oren Etzioni. 2013. Modeling missing data in distant supervision for information extraction. *Transactions of the Association for Computational Linguistics*, 1:367–378, October.

Ajit P Singh and Geoffrey J Gordon. 2008. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 650–658. ACM.

Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. 2013. Reasoning With Neural Tensor Networks For Knowledge Base Completion. In *Advances in Neural Information Processing Systems 26*.

Robert Speer, Catherine Havasi, and Henry Lieberman. 2008. Analogyspace: Reducing the dimensionality of common sense knowledge. In *AAAI*, pages 548–553.

Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D. Manning. 2012. Multi-instance multi-label learning for relation extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

Ilya Sutskever, Joshua B Tenenbaum, and Ruslan Salakhutdinov. 2009. Modelling relational data using Bayesian clustered tensor factorization. In *NIPS*, pages 1821–1828.

Ledyard R Tucker. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311.

Jason Weston, Antoine Bordes, Oksana Yakhnenko, and Nicolas Usunier. 2013. Connecting language and knowledge bases with embedding models for relation extraction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1366–1371, Seattle, Washington, USA, October. Association for Computational Linguistics.

Limin Yao, Aria Haghighi, Sebastian Riedel, and Andrew McCallum. 2011. Structured relation discovery using generative models. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1456–1466, Edinburgh, Scotland, UK., July. Association for Computational Linguistics.