Adaptive Regularizing Tucker Decomposition for Knowledge Graph Completion

Quanming Yao*, Shimin DI*, Yongqi Zhang*,
*4Paradigm Inc.
#Hong Kong University of Science and Technology
{yaoquanzhang, zhangyongqi}@4paradigm.com, {sdiaa}@cse.ust.hk

Abstract

Tensor factorization approaches have recently become popular in knowledge graph completion (KGC). Among them, TuckER, which introduces Tucker decomposition in KGC, is the state-of-the-art method. However, due to its high model complexity, neither effectiveness nor efficiency of TuckER is satisfied. In this paper, we improve TuckER by automated machine learning (AutoML) techniques. Specifically, we propose to regularize the over-parameterized core tensor in Tucker by the one-shot architecture search algorithm. The resulting new factorization method not only sparsifies but also improves the interpretability of core tensor. Finally, empirical results demonstrate that the proposed method achieves state-of-the-art performance on KGC.

1 Introduction

Knowledge Graph (KG) plays an important role in exploring and organizing knowledge base, which is applicable to many real world scenarios. Generally, real facts in KG are represented in the triplet form (head entity, relation, tail entity) or (h, r, t) for simplicity, e.g. (Beijing, capital of, China). Given a triplet, the crucial task in KGs is to verify whether the triplet is a real fact or not, i.e., knowledge graph completion (KGC). Recently, embedding approaches have been developed as a promising method to tackle this task [Wang et al., 2017]. The entities and relations are firstly mapped into low dimensional vectors h, r, t, then a scoring function (SF), i.e., f(h, r, t), is designed to indicate whether a triplet is a real fact.

In the past decade, various SFs have been proposed to improve the performance of KGs, such as TransE [Bordes et al., 2013], ConvE [Dettmers et al., 2018], DistMult [Wang et al., 2014], ComplEx [Trouillon et al., 2017] and SimplE [Kazemi and Poole, 2018]. Among kinds of methods, tensor factorization models (e.g., DistMult, ComplEx and SimplE) have been demonstrated their superiority due to the expressive guarantee [Wang et al., 2018] and better empirical performance [Lacroix et al., 2018]. More recently, another tensor-based approach, TuckER [Balazevic et al., 2019], adapts Tucker decomposition to serve as the SF and achieves state-of-the-art results. Apart from learning the entity and relation embeddings individually, TuckER introduces an extra core tensor to model the interaction between entity and relation embeddings. The core tensor enables different entities and relations to share the same correlated interaction.

However, the efficiency and effectiveness are still far from desired in TuckER. The core tensor in TuckER has complexity of $O(d_e^2d_r)$, where $d_e$ and $d_r$ are dimensions of entity and relation embedding, respectively. When the dimension of embeddings increases, the size of core tensor increases cubically, which prevents the model to achieve better performance [Lacroix et al., 2018]. Besides, the large amount of parameters in core tensor also makes TuckER’s training difficult since this computation cost increases dramatically and complex models tend to overfit without sufficient data.

In comparison, tensor factorization models such as ComplEx, SimplE achieve relatively good performance without introducing the dense core tensor [Lacroix et al., 2018]. The question comes that is it essential to learn a core tensor with so many trainable parameters to model the interaction between entity and relation embeddings? Based on the view that ComplEx and SimplE can be regarded to have a special core tensor with sparse constraint, we propose a novel way to regularize the core tensor by sparse and diagonal constraints. Inspired by the success of automated machine learning (AutoML) [Hutter et al., 2018], we propose an Adaptive Regularizing Tucker (ART) approach to adaptively search proper regularizer on the core tensor for any given KG. We summarize the contribution as follows:

- Based on TuckER, we propose a novel regularizing method to reduce its model complexity in order to improve Tucker’s performance.
- Inspired by AutoML, we form the regularizing problem as a searching problem. We implement an efficient algorithm to adaptively search the regularized Tucker core tensor for any given KG data.
- We test ART on the link prediction task with four popular benchmark datasets. Experimental results show that ART not only achieves outstanding performance in the above tasks, but also improves efficiency.

Notation. A set of triplets $S = \{(h, r, t)\}$ denotes a KG data with $h, t \in \mathbb{E}$ and $r \in \mathbb{R}$, where $\mathbb{E}$ and $\mathbb{R}$ are sets of entities and relations, respectively. Following [Kolda and Bader, 2009],
we use lowercase boldface for vectors (e.g., $h, t \in R^{d_e}$), uppercase boldface for matrix (e.g., $E \in R^{d_e \times n}$) and Euler script for 3-dimensional (3D) tensor (e.g., $G \in R^{d_e \times d_r \times d_e}$).

A tensor $G$ is diagonal when $g_{i,j,k} \neq 0$ holds if and only if $i = j = k$. We use $I_v$ to denote the diagonal tensor with $v$ on the super-diagonal and zeros elsewhere. Finally, $\times_n$ denotes the tensor product along the $n$-th mode.

## 2 Related Works

### 2.1 Tensor Factorization (TuckER) for KGC

As introduced in Section 1, TuckER encodes all interactions between entity and relation embeddings, which enables different entities and relations to share the same set of knowledge of a given KG. In the tensor factorization models, the KG is represented as a third-order binary tensor $X$, where each entry corresponds to a triplet, 1 indicating a real fact. In order to learn the embeddings, TuckER proposes to decompose $X$ by Tucker decomposition [Kolda and Bader, 2009]:

$$X \approx G \times_1 E^\top \times_2 R^\top \times_3 E^\top,$$

where $G \in R^{d_e \times d_r \times d_e}$ is the Tucker core tensor, $E \in R^{d_e \times n}$ and $R \in R^{d_r \times |R|}$ represent embedding of entities and relations, respectively. Then the SF in TuckER is defined as:

$$f(h, r, t) = G \times_1 h \times_2 r \times_3 t,$$

where $h, t \in R^{d_e}$ and $r \in R^{d_r}$. Although $d_e$ and $d_r$ are much smaller than $|E|$ or $|R|$, the size of $G$ will still be quite large when embedding size increases, which is essential to achieve good performance [Lacroix et al., 2018]. As a result, core tensor with large complexity are difficult to train and easy to overfit since there may not be enough triplets to meet the expressive power of the core tensor. We summarize the comparison of some advanced SFs in terms of model complexity, computation complexity in Table 1.

### 2.2 Automated Machine Learning (AutoML)

Automated Machine Learning (AutoML) [Hutter et al., 2018; Yao and Wang, 2019] has recently shown its power in designing better machine learning models which can adapt to the different tasks. Generally, two important aspects should be considered in AutoML, i.e., 1) search space: it defines what in principle should be searched, such as hyper-parameters or network architectures; 2) search algorithm: it aims to efficiently search in the search space. More recently, one-shot search (OAS) methods [Bender et al., 2018; Liu et al., 2018; Yao et al., 2020] have been proposed to reduce the search cost in classic AutoML techniques. Instead of searching and training candidate models separately, OAS represents the whole search space by a supernet [Bender et al., 2018] and keeps weights for the supernet, thus different architectures are forced to have the same weights (i.e., parameter-sharing).

## 3 The Search Problem

As in Section 2.1, the dense core tensor in TuckER not only makes the model hard to train but also inefficient to generate predictions. Hence, we propose to regularize the core tensor of TuckER to reduce model complexity here.

In TuckER, every element $g_{i,j,k}$ in the core tensor interprets the correlation among the $i$-th element of $h$, $j$-th element of $r$ and $k$-th element of $t$. However, it is quite redundant for TuckER’s core tensor to evaluate the correlations dimension by dimension. For instance, classic methods such as DistMult, SimplE and ComplEx can be represented to have special forms of the core tensors as in Figure 1 which are very sparse. But they can still achieve good performance since high dimensional KG embedding will dilute a large part of information. This motivates us to regularize TuckER’s core tensor by only interpreting the correlation between segmentations of entities and relations.

As shown in Figure 1(d), given a head entity $h \in R^{d_e}$, ART first divides the embedding $h$ into $m$ segmentations as

<table>
<thead>
<tr>
<th>Model</th>
<th>Scoring Function</th>
<th>Model Complexity</th>
<th>Computation Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>DistMult</td>
<td>$\langle h, r, t \rangle$</td>
<td>$O(n_e d + n_r d)$</td>
<td>$O(d)$</td>
</tr>
<tr>
<td>TuckER</td>
<td>$G \times_1 h \times_2 r \times_3 t$</td>
<td>$G$ is dense</td>
<td>$O(d_e^2 d_r + n_e d_e + n_r d_r)$</td>
</tr>
<tr>
<td>ART</td>
<td>$\sum_{i,j,k} G_{ijk} \times_1 h_i \times_2 r_j \times_3 t_k$</td>
<td>$G_{ijk}$ is adaptively sparsiﬁed</td>
<td>$O(m^3 + n_e d + n_r d)$</td>
</tr>
</tbody>
</table>
Recall that ART takes a discrete view of TuckER’s core tensor $G$ as cube segmentations $G = \{G_{ijk}\}$, where every cube $G_{ijk}$ is selected from the operation set $\emptyset = \{-I_3, I_3, I_3\}$. To enable an efficient search method, we design a continuous view as:

$$G_{ijk} = \sum_{o_p \in \emptyset} a_{ijk}^p \cdot o_p,$$  \hspace{1cm} (6)$$

where $o_p \in \emptyset$ and $a_{ijk}^p \in \{0, 1\}$ is the weight for the $p$-th choice in $\emptyset$. Then the SF of ART can be represented into a supernet (a weighted bipartite graph) in Figure 2, where $a_{ijk}^p$ is the weight of the edge between operation $o_p$ and $G_{ijk}$. $A = [a_{ijk}^p] \in R^{m^3 \times 3}$ maintains all operation weights in (6).

As in Section 2.2, parameter-sharing reduces the search time in AutoML. Hence, we propose to keep different SFs share the same embeddings during the search, which allows us to evaluate the performance in each epoch and avoid expensive full model training of candidate $G$. Motivated by the recent progress in optimizing network weights with binary values [Yao et al., 2020], we also maintain two copies of architecture weights, i.e., a continuous $A$ to maintain the continuous weights ($a_{ijk}^p \in [0, 1]$) and a discrete $\hat{A}$ with binary elements ($\hat{a}_{ijk}^p \in \{0, 1\}$), and recover $A$ from $\hat{A}$. In the sequel, we represent the supernet of cube segmentations $G$ selected by the architecture weight $A$ as SP($\hat{A}$) and the embeddings $\{h, r, t\}$ as $X$. Therefore, the loss function in Def. 2 can be represented as $L(\text{SP}(\hat{A})(X), S)$. The above steps are summarized in Algorithm 1.

**Algorithm 1 Adaptive Regularizing Tucker (ART) one-shot search algorithm**

1. Initialize embeddings $X_0$, architectures $A_0$, step-sizes $\eta$ and $\varepsilon$.
2. while not converged do
3. Get discrete architectures $A_{t+1} = [\hat{a}_{ijk}^p]$ from $A_t$, such as $\hat{a}_{ijk}^p = \begin{cases} 1 \text{ if } p = \arg \max_{p} a_{ijk}^p; \\ 0 \text{ otherwise}; \end{cases}$
4. Randomly sample a mini-batch $B_{tra}$ from $S_{tra}$;
5. Update embeddings $X$ with gradients as: $X_{t+1} \leftarrow X_t - \eta \nabla L(\text{SP}(A_{t+1})(X_t), B_{tra})$;
6. Randomly sample a mini-batch $B_{val}$ from $S_{val}$;
7. Update the continuous architecture $A$ as: $A_{t+1} \leftarrow A_t - \varepsilon \nabla A_t \sum_{(h, r, t) \in B_{val}} L(\text{SP}(A_{t+1})(X_{t+1}), B_{val})$;
8. end while
9. Derive $\hat{A}^*$ from the final searched $A^*$;
10. Get embeddings $X^*$ by training $\hat{A}^*$ from scratch to convergence.

**5 Experiments**

Following previous KGC models [Bordes et al., 2013; Trouillon et al., 2017; Kazemi and Poole, 2018; Balazevic et al., 2019], we mainly conduct experiments on four public benchmark data sets: WN18 [Bordes et al., 2013], WN18RR [Dettmers et al., 2018], FB15k [Bordes et al., 2013], FB15k237 [Toutanova and Chen, 2015]. WN18RR and FB15k237 are variants of WN18 and FB15k respectively by removing duplicate and inverse relations.

We test the performance on the link prediction task and adopt the classic metrics [Bordes et al., 2013; Wang et al., 2014]: MRR and Hit@10. We compare the proposed ART (with $m = 4$) with the popular KGC models, i.e. RotatE [Sun et al., 2019], ConvE [Dettmers et al., 2018], HolE/X [Xue et al., 2018], QuatE [Zhang et al., 2019], DistMult [Wang et al., 2014].
Table 2: Comparison of the best SFs identified by ART and the state-of-the-art SFs on the link prediction task.

<table>
<thead>
<tr>
<th>model</th>
<th>WN18</th>
<th>FB15k</th>
<th>FB15k237</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MRR</td>
<td>Hi@10</td>
<td>MRR</td>
</tr>
<tr>
<td>ART (ours)</td>
<td>0.959</td>
<td>56.8</td>
<td>0.840</td>
</tr>
<tr>
<td>SimplE [Kazemi and Poole, 2018]</td>
<td>0.950</td>
<td>55.5</td>
<td>0.830</td>
</tr>
<tr>
<td>TuckER [Balazevic et al., 2019]</td>
<td>0.953</td>
<td>52.6</td>
<td>0.795</td>
</tr>
<tr>
<td>CompEx [Trouillon et al., 2017]</td>
<td>0.951</td>
<td>55.1</td>
<td>0.831</td>
</tr>
<tr>
<td>DistMult [Wang et al., 2014]</td>
<td>0.821</td>
<td>50.7</td>
<td>0.817</td>
</tr>
<tr>
<td>QuaTE [Zhang et al., 2019]</td>
<td>0.950</td>
<td>56.2</td>
<td>0.833</td>
</tr>
<tr>
<td>ConvE [Dettmers et al., 2018]</td>
<td>0.943</td>
<td>50.0</td>
<td>0.797</td>
</tr>
<tr>
<td>HolE [Xue et al., 2018]</td>
<td>0.938</td>
<td>79.1</td>
<td>0.800</td>
</tr>
<tr>
<td>RotAT [Sun et al., 2019]</td>
<td>0.949</td>
<td>57.1</td>
<td>0.806</td>
</tr>
</tbody>
</table>

Table 3: Running time (in hours) analysis of SFs on single GPU.

<table>
<thead>
<tr>
<th>data set</th>
<th>ART (ours)</th>
<th>TuckER</th>
<th>DistMult</th>
</tr>
</thead>
<tbody>
<tr>
<td>WN18</td>
<td>Search</td>
<td>Training</td>
<td>25.42</td>
</tr>
<tr>
<td>WN18RR</td>
<td>3.12</td>
<td>3.94</td>
<td>18.70</td>
</tr>
<tr>
<td>FB15k</td>
<td>13.61</td>
<td>10.79</td>
<td>38.67</td>
</tr>
<tr>
<td>FB15k237</td>
<td>5.66</td>
<td>3.86</td>
<td>21.33</td>
</tr>
</tbody>
</table>

In addition, we only show two searched SFs over two data sets under \( m = 2 \) in Table 4 due to space limits. It indicates that ART can search different \( G \) for various KGs.

Table 4: The example of searched \( G \) on WN18RR with \( m = 2 \).

<table>
<thead>
<tr>
<th>data sets</th>
<th>( q_{111} )</th>
<th>( q_{112} )</th>
<th>( q_{121} )</th>
<th>( q_{122} )</th>
<th>( q_{211} )</th>
<th>( q_{212} )</th>
<th>( q_{221} )</th>
<th>( q_{222} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>WN18RR</td>
<td>( z_1 )</td>
<td>( z_2 )</td>
<td>( z_3 )</td>
<td>( z_1 )</td>
<td>( z_2 )</td>
<td>( z_3 )</td>
<td>( z_0 )</td>
<td>( z_1 )</td>
</tr>
<tr>
<td>FB15k237</td>
<td>( z_1 )</td>
<td>( z_2 )</td>
<td>( z_3 )</td>
<td>( z_0 )</td>
<td>( z_1 )</td>
<td>( z_2 )</td>
<td>( z_3 )</td>
<td>( z_4 )</td>
</tr>
</tbody>
</table>

References


