Adaptive Regularizing Tucker Decomposition for Knowledge Graph Completion

Quanming Yao[#], Shimin DI[*], Yongqi Zhang[#],

[#]4Paradigm Inc.
[*]Hong Kong University of Science and Technology
{yaoquanming, 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 $\mathbf{h}, \mathbf{r}, \mathbf{t}$, then a scoring function (SF), i.e., $f(\mathbf{h}, \mathbf{r}, \mathbf{t})$, 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 KGC, 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 SimpIE) 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 $O(d_e^2 d_r)$, 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.

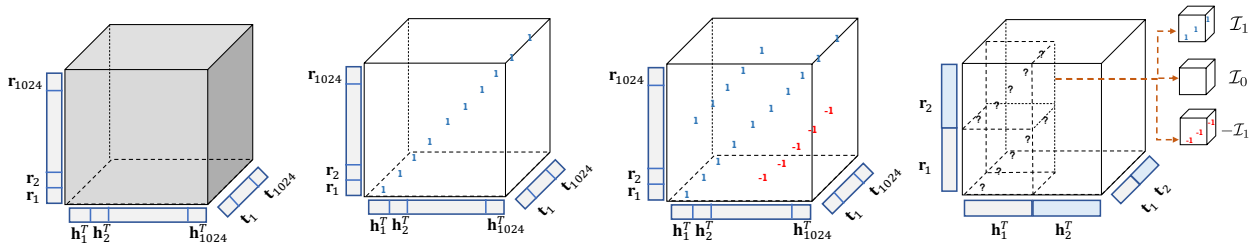
In comparison, tensor factorization models such as ComplEx, SimplE achieve relatively good performance without introducing the dense core tensor [Lacroix *et al.*, 2018]. The question comes that is it essential to learn a core tensor with so many trainable parameters to model the interaction between 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 $\mathbb{S} = \{(h, r, t)\}$ 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],

Table 1: Summary of scoring functions. n_e and n_r are the number of entities and relations, respectively. $\langle\cdot\rangle$ represents the dot product. Note that DistMult, ComplEx and ART utilize the same dimensionality d for entities and relations, while Tucker sets d_e and d_r respectively. The computation complexity is the cost of calculating the score of any given $(\mathbf{h}, \mathbf{r}, \mathbf{t})$.

Model	Scoring Function		Model Complexity	Computation Complexity
DistMult	$\langle \mathbf{h}, \mathbf{r}, \mathbf{t} \rangle$		$O(n_e d + n_r d)$	$O(d)$
TuckER	$\mathcal{G} \times_1 \mathbf{h} \times_2 \mathbf{r} \times_3 \mathbf{t}$	$\mathcal{G}$ is dense	$O(d_e^2 d_r + n_e d_e + n_r d_r)$	$O(d_e^2 d_r)$
ART	$\sum_{ijk} \mathcal{G}_{ijk} \times_1 \mathbf{h}_i \times_2 \mathbf{r}_j \times_3 \mathbf{t}_k$	$\mathcal{G}_{ijk}$ is adaptively sparsified	$O(m^3 + n_e d + n_r d)$	$O(m^2 d)$



(a) TuckER. (b) DistMult. (c) ComplEx. (d) ART.

Figure 1: (a) Each element in TuckER core tensor interprets the correlation between entities and relations of every embedding dimension, thus $\mathcal{O}(d^3 + n_e d + n_r d)$; (b) and (c) illustrate DistMult and ComplEx under representations of TuckER core tensor, respectively; (d) Each element in ART core tensor only interprets the correlation between entities and relations of embedding segmentation, hence $O(m^3 + n_e d + n_r d)$. Note that elements that are set to 0 are represented in white while gray elements are unknown.

we use lowercase boldface for vectors (e.g., $\mathbf{h}, \mathbf{t} \in R^{d_e}$), uppercase boldface for matrix (e.g., $\mathbf{E} \in R^{d_e \times n}$) and Euler script for 3-dimensional (3D) tensor (e.g., $\mathcal{G} \in R^{d_e \times d_r \times d_e}$). A tensor $\mathcal{G}$ is *diagonal* when $g_{i,j,k} \neq 0$ holds if and only if $i = j = k$. We use $\mathcal{I}_v$ 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 $\mathcal{X}$, where each entry corresponds to a triplet, 1 indicating a real fact. In order to learn the embeddings, TuckER proposes to decompose $\mathcal{X}$ by Tucker decomposition [Kolda and Bader, 2009]:

$$\mathcal{X} \approx \mathcal{G} \times_1 \mathbf{E}^\top \times_2 \mathbf{R}^\top \times_3 \mathbf{E}^\top, \tag{1}$$

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

$$f(\mathbf{h}, \mathbf{r}, \mathbf{t}) = \mathcal{G} \times_1 \mathbf{h} \times_2 \mathbf{r} \times_3 \mathbf{t}, \tag{2}$$

where $\mathbf{h}, \mathbf{t} \in R^{d_e}$ and $\mathbf{r} \in R^{d_r}$. Although d_e and d_r are much smaller than $|\mathbb{E}|$ or $|\mathbb{R}|$, the size of $\mathcal{G}$ 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 rencently, 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 $\mathbf{h}$, j-th element of $\mathbf{r}$ and k-th element of $\mathbf{t}$. 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 $\mathbf{h} \in R^d$, ART first divides the embedding $\mathbf{h}$ into m segmentations as

$\mathbf{h} = [\mathbf{h}_1; \ldots; \mathbf{h}_m]$ where $\mathbf{h}_i \in R^{\frac{d}{m}}$, and same for relation $\mathbf{r}$ and tail $\mathbf{t}$. Then, after delving deep into the designs of DistMult and ComplEx in Figure 1 (b) and (c), we observe that simple values (i.e., -1, 0 and 1) on the diagonal form of the core tensor are fully expressive for capturing interactions. Therefore, we propose that for each core tensor cube, we choose the appropriate diagonal tensor from $\{-\mathcal{I}_1, \mathcal{I}_0, \mathcal{I}_1\}$ as Figure 1 (d). Note that any positive or negative value v can be used for $\mathcal{I}_v$ here. We utilize 1 and -1 for simplicity. Then, SFs considered by ART is in Definition 1.

Definition 1. *Given a tensor $\mathcal{G} \in R^{d \times d \times d}$, let $\delta(\mathcal{G})$ divide $\mathcal{G}$ into m^3 cube segmentations $\mathbb{G} = \{\mathcal{G}_{ijk}\}$ where $\mathcal{G}_{ijk} \in \mathbb{O}$. The SF here is:*

$$f_{\mathbb{G}}(\mathbf{h}, \mathbf{r}, \mathbf{t}) = \sum\nolimits_{ijk} \mathcal{G}_{ijk} \times_1 \mathbf{h}_i \times_2 \mathbf{r}_j \times_3 \mathbf{t}_k. \quad (3)$$

Note that each $\mathcal{G}_{ijk}$ can be arbitrarily and independently choosen from $\mathbb{O}$. Here we denote all possible $\mathbb{G}$s returned by $\delta(\mathcal{G})$ as a space $\mathfrak{G}$. As introduced in Section 1, SFs design often suffer from KG diversity. Hence, in this paper, given any KG data $\mathbb{S}$, we follow AutoML and formulate the problem to find a better $\mathbb{G}$ from $\mathfrak{G}$ for $\mathbb{S}$ as follows:

Definition 2 (Search Problem)**.** *Given the training and validation triplet sets $\mathbb{S}_{tra}$ and $\mathbb{S}_{val}$ respectively, the adaptive regularizing Tucker search problem is defined as follows:*

$$\bar{\mathbb{G}} = \arg\min_{\mathbb{G} \in \mathfrak{G}} \sum_{(h,r,t) \in \mathbb{S}_{val}} \mathcal{L}\left(f_{\mathbb{G}}\left(\bar{\mathbf{h}}, \bar{\mathbf{r}}, \bar{\mathbf{t}}\right), \mathbb{S}_{val}\right), \quad (4)$$

$$s.t.\ \{\bar{\mathbf{h}}, \bar{\mathbf{r}}, \bar{\boldsymbol{t}}\} = \arg\min_{\{\mathbf{h},\mathbf{r},\mathbf{t}\}} \sum_{(h,r,t) \in \mathbb{S}_{tra}} \mathcal{L}\left(f_{\mathbb{G}}\left(\mathbf{h}, \mathbf{r}, \mathbf{t}\right), \mathbb{S}_{tra}\right), \quad (5)$$

where $\mathcal{L}(\cdot, \mathbb{S}_{val})$ and $\mathcal{L}(\cdot, \mathbb{S}_{tra})$ measures the loss of given embeddings on the corresponding data.

Note that we adopt multi-log loss as $\mathcal{L}$ because it currently achieves the best performance [Lacroix *et al.*, 2018]. For a given KG data, the SF design problem has been converted to adaptively searching a proper regularizer $\mathbb{G}$ with learned embeddings $\{\mathbf{h}, \mathbf{r}, \mathbf{t}\}$. However, it is hard to search a proper $\mathbb{G}$ from a large amount of candidates. For instance, there are 3^{64} candidates when $m = 4$. Hence, we propose to represent the search space with a supernet, which enables an efficient algorithm on it.

4 One-Shot Search Algorithm

Recall that ART takes a discrete view of TuckER's core tensor $\mathcal{G}$ as cube segmentations $\mathbb{G} = \{\mathcal{G}_{ijk}\}$, where every cube $\mathcal{G}_{ijk}$ is selected from the operation set $\mathbb{O} = \{-\mathcal{I}_1, \mathcal{I}_0, \mathcal{I}_1\}$. To enable an efficient search method, we design a continuous view as:

$$\mathcal{G}_{ijk} = \sum\nolimits_{o_p \in \mathbb{O}} a^p_{ijk} \cdot o_p, \quad (6)$$

where $o_p \in \mathbb{O}$ and $a^p_{ijk} \in \{0, 1\}$ is the weight for the p-th choice in $\mathbb{O}$. Then the SF of ART can be represented into a supernet (a weighted bipartite graph) in Figure 2, where a^p_{ijk} is the weight of the edge between operation o_p and $\mathcal{G}_{ijk}$. $\mathbf{A} = [a^p_{ijk}] \in R^{m^3 \times 3}$ 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

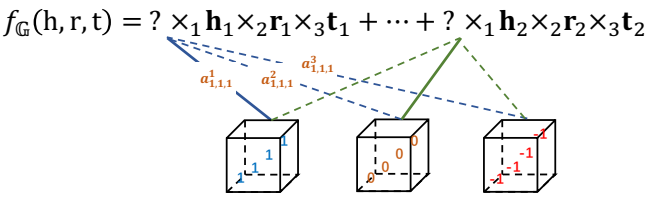


Figure 2: A supernet representation of ART's search space based on Eq.6 (with $m = 2$).

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 $\mathbb{G}$. 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 $\mathbf{A}$ to maintain the continuous weights ($a^p_{ijk} \in [0, 1]$) and a discrete $\bar{\mathbf{A}}$ with binary elements ($\bar{a}^p_{ijk} \in \{0, 1\}$), and recover $\bar{\mathbf{A}}$ from $\mathbf{A}$. In the sequel, we represent the supernet of cube segmentations $\mathbb{G}$ selected by the architecture weight $\bar{\mathbf{A}}$ as SP($\bar{\mathbf{A}}$) and the embeddings $\{\mathbf{h}, \mathbf{r}, \mathbf{t}\}$ as $\mathbf{X}$. Therefore, the loss function in Def. 2 can be represented as $\mathcal{L}\left(f_{\text{SP}(\bar{\mathbf{A}})}(\mathbf{X}), \mathbb{S}\right)$. The above steps are summarized in Algorithm 1.

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

1: Initialize embeddings $\mathbf{X}_0$, architectures $\mathbf{A}_0$, step-sizes η and ε.
2: **while** not converged **do**
3: Get *discrete* architectures $\bar{\mathbf{A}}_{t+1} = [\bar{a}^p_{ijk}]$ from $\mathbf{A}_t$, such as $\bar{a}^p_{ijk} = \begin{cases} 1 \text{ if } p = \arg\max_{\bar{p}} a^{\bar{p}}_{ijk} \\ 0 \text{ otherwise.} \end{cases}$;
4: Randomly sample a mini-batch $\mathbb{B}_{tra}$ from $\mathbb{S}_{tra}$;
5: Update embeddings $\mathbf{X}$ with gradients as: $\mathbf{X}_{t+1} \leftarrow \mathbf{X}_t - \eta \nabla_{\mathbf{X}_t} \sum_{(h,r,t) \in \mathbb{B}_{\text{tra}}} \mathcal{L}(f_{\text{SP}(\bar{\mathbf{A}}_{t+1})}(\mathbf{X}_t), \mathbb{B}_{tra})$;
6: Randomly sample a mini-batch $\mathbb{B}_{val}$ from $\mathbb{S}_{val}$;
7: Update the continuous architecture $\mathbf{A}$ as: $\mathbf{A}_{t+1} \leftarrow \mathbf{A}_t - \varepsilon \nabla_{\mathbf{A}_t} \sum_{(h,r,t) \in \mathbb{B}_{\text{val}}} \mathcal{L}\left(f_{\text{SP}(\mathbf{A}_t)}(\mathbf{X}_{t+1}), \mathbb{B}_{val}\right)$;
8: **end while**
9: Derive $\bar{\mathbf{A}}^*$ from the final searched $\mathbf{A}^*$;
10: Get embeddings $\mathbf{X}^*$ by training $\bar{\mathbf{A}}^*$ 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], HolEX [Xue *et al.*, 2018], QuatE [Zhang *et al.*, 2019], DistMult [Wang *et al.*,

Table 2: Comparison of the best SFs identified by ART and the state-of-the-art SFs on the link prediction task.

model	WN18		WN18RR		FB15k		FB15k237	
	MRR	Hit@10	MRR	Hit@10	MRR	Hit@10	MRR	Hit@10
RotatE [Sun *et al.*, 2019]	0.949	**95.9**	0.476	57.1	0.797	88.4	0.297	48.0
ConvE [Dettmers *et al.*, 2018]	0.943	95.6	0.430	52.0	0.657	83.1	0.325	50.1
HolEX [Xue *et al.*, 2018]	0.938	94.9	-	-	0.800	88.6	-	-
QuatE [Zhang *et al.*, 2019]	0.950	**95.9**	0.488	**58.2**	0.833	90.0	0.357	55.3
DistMult [Wang *et al.*, 2014]	0.821	95.2	0.443	50.7	0.817	89.5	0.349	53.7
ComplEx [Trouillon *et al.*, 2017]	0.951	95.7	0.471	55.1	0.831	90.5	0.347	54.1
SimplE [Kazemi and Poole, 2018]	0.950	**95.9**	0.48	55.5	0.830	90.3	0.350	54.4
TuckER [Balazevic *et al.*, 2019]	**0.953**	95.8	0.470	52.6	0.795	89.2	0.358	54.4
ART (ours)	0.950	**95.9**	**0.489**	56.8	**0.840**	**90.8**	**0.360**	**55.0**

2014], ComplEx [Trouillon *et al.*, 2017], SimplE [Kazemi and Poole, 2018] and TuckER [Balazevic *et al.*, 2019].

The effectiveness comparison of ART with the other methods are in Table 2. Firstly, it is clear that TuckER achieves good performance among classic SFs. Furthermore, ART achieves the state-of-the-art performance compared with TuckER. This is because that ART regularizes the dense core tensor to less complexity so that ART's core tensor is easy to train and unlikely overfit.

We summarize the running time of two steps of ART and other SFs on 4 data sets in Table 3. Note that ART sets embedding dimensions to 512 in search procedure for all data sets. As for trained-from-scratch, we set embeddings dimensions for ART, TuckER and DistMult at 1024. It is obvious that ART can extremely reduce the time cost compared with TuckER. The searched SF training time of ART is a little longer than the simplest tensor-based method, DistMult. Moreover, ART search takes a bit more time compared with stand-alone training because it needs to forward and backward loss for updating architectures in search. Gennerally, the time cost of ART is cheap.

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

data set	ART		TuckER	DistMult
	Search	Training		
WN18	5.89	4.73	25.42	1.9
WN18RR	3.12	3.04	18.70	0.42
FB15k	13.61	10.79	38.67	8.36
FB15k237	5.66	3.86	21.33	2.6

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 $\mathbb{G}$ for various KGs.

Table 4: The example of searched $\mathbb{G}$ on WN18RR with $m = 2$.

data sets	$\mathcal{G}_{111}$	$\mathcal{G}_{112}$	$\mathcal{G}_{121}$	$\mathcal{G}_{122}$	$\mathcal{G}_{211}$	$\mathcal{G}_{212}$	$\mathcal{G}_{221}$	$\mathcal{G}_{222}$
WN18RR	$\mathcal{I}_1$	$\mathcal{I}_1$	$-\mathcal{I}_1$	$\mathcal{I}_0$	$\mathcal{I}_1$	$-\mathcal{I}_1$	$\mathcal{I}_1$	$-\mathcal{I}_1$
FB15k237	$\mathcal{I}_1$	$-\mathcal{I}_1$	$\mathcal{I}_1$	$\mathcal{I}_1$	$\mathcal{I}_0$	$-\mathcal{I}_1$	$-\mathcal{I}_1$	$\mathcal{I}_1$

References

[Balazevic *et al.*, 2019] I. Balazevic, C. Allen, and T. Hospedales. Tucker: Tensor factorization for knowledge graph completion. In *(EMNLP-IJCNLP)*, pages 5188–5197, 2019.

[Bender *et al.*, 2018] G. Bender, P.-J. Kinderm, B. Zoph, V. Vasudevan, and Q. Le. Understanding and simplifying one-shot architecture search. In *ICML*, pages 549–558, 2018.

[Bordes *et al.*, 2013] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, pages 2787–2795, 2013.

[Dettmers *et al.*, 2018] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, 2018.

[Hutter *et al.*, 2018] F. Hutter, L. Kotthoff, and J. Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018.

[Kazemi and Poole, 2018] S. Kazemi and D. Poole. Simple embedding for link prediction in knowledge graphs. In *NeurIPS*, pages 4284–4295, 2018.

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

[Lacroix *et al.*, 2018] Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. *ICML*, page 2863–2872, 2018.

[Liu *et al.*, 2018] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. In *ICLR*, 2018.

[Sun *et al.*, 2019] Z. Sun, Z. Deng, J. Nie, and J. Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *ICLR*, 2019.

[Toutanova and Chen, 2015] K. Toutanova and D. Chen. Observed versus latent features for knowledge base and text inference. In *Workshop on CVSMC*, pages 57–66, 2015.

[Trouillon *et al.*, 2017] T. Trouillon, Christopher R., É. Gaussier, J. Welbl, S. Riedel, and G. Bouchard. Knowledge graph completion via complex tensor factorization. *JMLR*, 18(1):4735–4772, 2017.

[Wang *et al.*, 2014] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, 2014.

[Wang *et al.*, 2017] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *TKDE*, 29(12):2724–2743, 2017.

[Wang *et al.*, 2018] Y. Wang, R. Gemulla, and H. Li. On multi-relational link prediction with bilinear models. In *AAAI*, 2018.

[Xue *et al.*, 2018] Y. Xue, Y. Yuan, Z. Xu, and A. Sabharwal. Expanding holographic embeddings for knowledge completion. In *NeurIPS*, pages 4491–4501, 2018.

[Yao and Wang, 2019] Q. Yao and M. Wang. Taking human out of learning applications: A survey on automated machine learning. Technical report, arXiv preprint, 2019.

[Yao *et al.*, 2020] Q. Yao, J. Xu, W. Tu, and Z. Zhu. Differentiable neural architecture search via proximal iterations. In *AAAI*, 2020.

[Zhang *et al.*, 2019] S. Zhang, Y. Tay, L. Yao, and Q. Liu. Quaternion knowledge graph embeddings. In *NeurIPS*, pages 2731–2741, 2019.