Kronecker Decomposition for Knowledge Graph Embeddings

CAGLAR DEMIR, Data Science Group, Paderborn University, Germany
JULIAN LIENEN, Heinz Nixdorf Institute, Paderborn University, Germany
AXEL-CYRILLE NGONGA NGOMO, Data Science Group, Paderborn University, Germany

Knowledge graph embedding research has mainly focused on learning continuous representations of entities and relations tailored towards the link prediction problem. Recent results indicate an ever increasing predictive ability of current approaches on benchmark datasets. However, this effectiveness often comes with the cost of over-parameterization and increased computationally complexity. The former induces extensive hyperparameter optimization to mitigate malicious overfitting. The latter magnifies the importance of winning the hardware lottery. Here, we investigate a remedy for the first problem. We propose a technique based on Kronecker decomposition to reduce the number of parameters in a knowledge graph embedding model, while retaining its expressiveness. Through Kronecker decomposition, large embedding matrices are split into smaller embedding matrices during the training process. Hence, embeddings of knowledge graphs are not plainly retrieved but reconstructed on the fly. The decomposition ensures that elementwise interactions between three embedding vectors are extended with interactions within each embedding vector. This implicitly reduces redundancy in embedding vectors and encourages feature reuse. To quantify the impact of applying Kronecker decomposition on embedding matrices, we conduct a series of experiments on benchmark datasets. Our experiments suggest that applying Kronecker decomposition on embedding matrices leads to an improved parameter efficiency on all benchmark datasets. Moreover, empirical evidence suggests that reconstructed embeddings entail robustness against noise in the input knowledge graph. To foster reproducible research, we provide an open-source implementation of our approach, including training and evaluation scripts as well as pre-trained models.[1]

CCS Concepts: • **Computing methodologies** → **Machine learning algorithms**; **Artificial intelligence**.

Additional Key Words and Phrases: Knowledge Graph Embedding, Kronecker Decomposition, Link Prediction

ACM Reference Format:
Caglar Demir, Julian Lienen, and Axel-Cyrille Ngonga Ngomo. 2022. Kronecker Decomposition for Knowledge Graph Embeddings. *ACM Trans. Graph.* 37, 4, Article 111 (August 2022), 17 pages. https://doi.org/10.1145/3511095.3531276

1 INTRODUCTION

Knowledge Graph Embedding (KGE) models learn continuous vector representations of entities and relations [Demir and Ngomo 2021a; Dettmers et al. 2018; Nickel et al. 2015; Sun et al. 2019; Trouillon et al. 2016; Wang et al. 2017]. These representations have been successfully applied in a wide range of applications including question answering, link prediction, and recommender systems [Cai et al. 2018; Eder 2012; Ji et al. 2020; Nickel et al. 2015].

[1]https://github.com/dice-group/dice-embeddings

Authors' addresses: Caglar Demir, caglar.demir@upb.de, Data Science Group, Paderborn University, Paderborn, Germany; Julian Lienen, julian.lienen@upb.de, Heinz Nixdorf Institute, Paderborn University, Paderborn, Germany; Axel-Cyrille Ngonga Ngomo, axel.ngonga@upb.de, Data Science Group, Paderborn University, Paderborn, Germany.

Manuscript submitted to ACM

Most KGE models are designed to retain a linear complexity in the number of trainable parameters and in the number of triples to scale to large Knowledge Graphs (KGs) [Bordes et al. 2013; Demir and Ngomo 2021a; Sun et al. 2019; Trouillon et al. 2016; Yang et al. 2015]. Although the number of parameters of most KGE models grows linearly in the number of unique entities and relations, training on large KGs become a computationally challenging task. For example, when embedding Freebase with an embedding size of 200, DistMult requires 33 GB of main memory solely to store its parameters [Dettmers et al. 2018]. Such considerable memory requirement is still a barrier for efficient training and inference of embedding models [Edalati et al. 2021]. Although there have been numerous attempts to reduce high memory usage by means of decomposing pre-trained overparameterized language models (see DistilBERT [Sanh et al. 2019], TinyBERT [Jiao et al. 2019], MobileBERT [Sun et al. 2020], ALP-KD[Passban et al. 2020], MATE-KD [Rashid et al. 2021], and KnGPT2 [Edalati et al. 2021]), the decomposition of KGE models have not yet been studied.

In this paper, we investigate the problem of learning *compressed* KGEs. Our aim is to reduce the number of explicitly stored parameters, while retaining the expressiveness of KGE models. Trouillon et al. [2016] showed that finding the best ratio between expressiveness and the number of parameters is a keystone in successful applications of KGE models. To increase the expressiveness of most KGE models, increasing the embedding vector size is often the only option [Dettmers et al. 2018]. Yet, solely increasing the parameters of models (a) makes models more prone to overfitting and (b) increases the hardware demand, i.e., the importance of winning the hardware lottery is magnified [Hooker 2021]. KGE literature shows that considerable amount of effort is often invested into hyperparameter optimization to find a good ratio between expressiveness and the number of parameters to mitigate overfitting [Ruffinelli et al. 2019; Sun et al. 2019; Zhang et al. 2019]. With this consideration, we propose a technique based on Kronecker Decomposition (KD). Applying KD on embedding matrices implicitly reduces redundancy in embedding vectors and encourages feature reuse (see Section 4). Through the decomposition, embeddings are not merely retrieved but reconstructed on the fly. This inherently reduces the number of explicitly stored parameters, and hence decreases the memory footprint of the decomposed models. Importantly, our technique can be readily used in combination with many KGE models.

To evaluate our technique, we conducted a series of experiments on benchmark datasets. On each benchmark dataset, we applied a fine-grained parameter sweep in the size of embedding vectors. All competing KGE models are trained via a standard training technique as in the literature (see Section 3.2). We developed a hardware-agnostic embedding framework to conduct our experiments with an ease (see Section 5.4). To ensure the reproducibility of our experiments, we fixed the seed for the pseudo-random generator to 1. Overall, our results suggest that learning compressed embeddings via KD reduces the number of parameters on each dataset with each hyperparameter configuration, while yielding competitive performance in the link prediction task (see Section 6). First series of our experiments indicated that **59% parameter reduction** can be achieved via applying KD on relation embeddings, while retraining the training and testing performance. Moreover, applying KD on relation embeddings and entity embeddings results in on average **16.4 times fewer parameters**. We also evaluate link prediction performances of models with model calibration and under noisy input triples, since most predictions in the link prediction task are often uncalibrated and publicly available datasets involve noisy triples (see Section 3.3). Results of the former experiments suggest that MRR performance of all models are improved up to absolute 10% provided that they suffer from overfitting. As the degree of overfitting decreases, model calibration techniques (see Label Smoothing and Label Relaxation in Section 3) do not improve generalization performances. Results of the latter experiments suggest that learning compressed knowledge graph embeddings makes models more robust against noise in KGs.

2 RELATED WORK

In the last decade, a plethora of KGE approaches have been successfully applied to tackle various tasks [Cai et al. 2018; Gligorijević et al. 2021; Ji et al. 2020; Nickel et al. 2015]. Here, we give a brief overview of selected KGE approaches.

Nickel et al. [2011] proposed a three-way factorization of a third-order binary tensor representing the input KG. The proposed approach (RESCAL) is limited in its scalability as it has a quadratic complexity in the factorization rank [Nickel et al. 2016]. Yang et al. [2015] proposed DistMult that can be seen as an extension of RESCAL with a diagonal matrix per relation, where the dense core tensor is replaced with diagonal matrices. DistMult does not perform well on triples with antisymmetric relations (e.g. (`Barack`, `HasChild`, `Malia`)), whereas it performs well on symmetric relations (e.g. (`Barack`, `Married`, `Michelle`)). To avoid this shortcoming, Trouillon et al. [2016] extended DistMult by learning representations in a complex vector space. Their approach (ComplEx) is able to infer both symmetric and antisymmetric relations via a Hermitian inner product of embeddings which involves the conjugate-transpose of one of the two input vectors. Motivated by learning complex-valued embeddings, Sun et al. [2019] designed RotatE which employs a rotational model taking predicates as rotations from subjects to objects in complex space via the element-wise Hadamard product. QuatE extends ComplEx into quaternions through applying the quaternion multiplication followed by an inner product to compute scores of triples [Zhang et al. 2019].

All aforementioned approaches learn embeddings of entities and relations through capturing multiplicative based interactions. Although such approaches perform well in terms of predictive accuracy and computational complexity, to increase their expressiveness the embedding vector size is the only option. Dettmers et al. [2018], Nguyen et al. [2017], and Balažević et al. [2019a] have shown that convolution operation can be applied to increase the expresiveness of KGE models without significantly increasing the number of parameters. ConvE applies a 2D convolution operation to model the interactions between entities and relations [Dettmers et al. 2018]. ConvKB extends ConvE by omitting the reshaping operation in the encoding of representations in the convolution operation [Nguyen et al. 2017]. Similarly, HypER extends ConvE by applying relation-specific 1D convolutions as opposed to applying filters from concatenated subject and relation vectors [Balažević et al. 2019a]. Demir et al. [2021]; Demir and Ngomo [2021a] extended ConvE through combining 2D convolution operation with Hermitian inner product, Quaternion and Octonion multiplications.

3 BACKGROUND

3.1 Knowledge Graphs & Link Prediction

Let $\mathcal{E}$ and $\mathcal{R}$ denote a set of entities and relations. A knowledge graph (KG) can be defined as a set of triples $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, where each triple $(\mathtt{h}, \mathtt{r}, \mathtt{t}) \in \mathcal{G}$ contains two entities $\mathtt{h}, \mathtt{t} \in \mathcal{E}$ and a relation $\mathtt{r} \in \mathcal{R}$ [Demir and Ngomo 2021a]. Hence, knowledge graphs represent structured collections of facts in the form of typed relationships between entities [Hogan et al. 2020]. These collections of facts have been used in a wide range of applications, including web search, question answering, and recommender systems [Nickel et al. 2015]. Yet, most knowledge graphs on the web are far from complete. The link prediction task on KGs refers to predicting whether a triple is likely to be true [Demir and Ngomo 2021a; Dettmers et al. 2018]. This task is often formulated as the problem of learning a parametrized scoring function $\phi_\Theta : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightarrow \mathbb{R}$ such that $\phi_\Theta(\mathtt{h}, \mathtt{r}, \mathtt{t})$ ideally signals the likelihood of $(\mathtt{h}, \mathtt{r}, \mathtt{t})$ is true [Ji et al. 2020]. Here, Θ contains embeddings of entities and relations along with other trainable parameters. For instance, given the triples (`Barack`, `Married`, `Michelle`) and $(\mathtt{Michelle}, \mathtt{HasChild}, \mathtt{Malia}) \in \mathcal{G}$, a good scoring function is expected to return high scores for (`Barack`, `HasChild`, `Malia`) and (`Michelle`, `Married`, `Barack`), while returning a considerably lower score for (`Malia`, `HasChild`, `Barack`).

3.2 Knowledge Graph Embeddings and Training Strategies

Most knowledge graph embedding (KGE) models are designed to learn continuous vector representations of entities and relations tailored towards predicting missing triples. In our notation, the embedding vector of the entity $e \in \mathcal{E}$ is denoted by $\mathbf{e} \in \mathbb{R}^{d_e}$ and the embedding vector for the relation $r \in \mathcal{R}$ is denoted by $\mathbf{r} \in \mathbb{R}^{d_r}$. Embedding vectors are learned through minimizing a loss function (see Equation 2) with a first-order optimizer. The resulting models are then evaluated w.r.t. their ability of predicting missing entity rankings [Ruffinelli et al. 2019].

Three training strategies are commonly used for KGE models. Bordes et al. [2013] designed a negative sampling technique via perturbing an entity in a randomly sampled triple. In this setting, a triple $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$ is considered as a positive example, whilst $\{(\mathsf{h}, \mathsf{r}, \mathsf{x}) \mid \forall x \in \mathcal{E}\} \cup \{(\mathsf{x}, \mathsf{r}, \mathsf{t}) \mid \forall x \in \mathcal{E}\}$ is considered as a set of possible candidate negative examples. For each positive triple $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$, a negative triple is sampled from the set of corresponding candidate negative triples. Kotnis and Nastase [2017] analysed the impact of variations of the negative sampling technique designed by Bordes et al. [2013] on the link prediction task. Lacroix et al. [2018] discarded the idea of randomly sampling negative triples and proposed 1vsAll/1vsN the training strategy. For each positive triple $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$, all possible tail perturbed set of triples are considered as negative triples regardless of whether a perturbed triple exists in the input knowledge graph KG ($\{(\mathsf{h}, \mathsf{r}, \mathsf{x}) | \forall x \in \mathcal{E} : x \neq t\}$) as similarly done by Bordes et al. [2013]. Given that this setting does not involve negative triples via head perturbed entities, a data augmentation technique is applied to add inverse triples (also known as reciprocal triples [Balažević et al. 2019b]) $(\mathsf{t}, \mathsf{r}^{-1}, \mathsf{h})$ for each $(\mathsf{h}, \mathsf{r}, \mathsf{t})$. Their results showed that even simpler models reached state-of-the-art performance in link prediction task via 1vsAll. This partially stems from the fact that the ability of modelling antisymmetric relations is not required due to learning inverse relation representations. In the 1vsAll training strategy, a training data point consists of (h, r) and a binary vector containing a single "1" for the t and "0"s for other entities. Dettmers et al. [2018] extended 1vsAll into KvsAll[2] via constructing multi-label binary vectors for each (h, r). More specifically, a training data point consists of a pair (h, r) and a binary vector containing "1" for $\{x | x \in \mathcal{E} \wedge (\mathsf{h}, \mathsf{r}, \mathsf{x}) \in \mathcal{G}\}$ and "0"s for other entities. During training, for a given pair (h, r), predicted scores (logits) for all entities are computed, i.e., $\forall x \in \mathcal{E} : \phi((\mathsf{h}, \mathsf{r}, \mathsf{x}))) =: \mathbf{z} \in \mathbb{R}^{|\mathcal{E}|}$. Through the logistic sigmoid function $\sigma(\mathbf{z}) = \frac{1}{1+\exp(-\mathbf{z})}$, scores are normalized to obtain predicted probabilities of entities denoted by $\hat{\mathbf{y}}$. A loss incurred on a training data point is then computed as

$$l(\hat{\mathbf{y}}, \mathbf{y}) = -\frac{1}{|\mathcal{E}|} \sum_{i=1}^{|\mathcal{E}|} \mathbf{y}^{(i)} \log(\hat{\mathbf{y}}^{(i)} + \big(1 - \mathbf{y}^{(i)}\big) \log\big(1 - \hat{\mathbf{y}}^{(i)}\big), \tag{1}$$

where $\mathbf{y} \in [0, 1]^{|\mathcal{E}|}$ is the binary sparse label vector. If $(\mathsf{h}, \mathsf{r}, \mathsf{e}_i) \in \mathcal{G}$, then $\mathbf{y}^{(i)} = 1$, otherwise $\mathbf{y}^{(i)} = 0$. Recent works show that learning Θ by means of minimizing Equation 1 often leads to state-of-the-art link prediction performance [Balažević et al. 2019a,b; Demir and Ngomo 2021a; Dettmers et al. 2018]. Expectedly, 1vsAll and Kvsall are computationally more expensive than the negative sampling. As $|\mathcal{E}|$ increases, 1vsAll and KvsAll training strategies become less applicable. Yet, recent KGE models are commonly trained with 1vsAll or KvsAll [Ruffinelli et al. 2019].

3.3 Model Calibration and Robustness

Section 3.2 elucidated that most KGE models are trained to predict missing entities in a fashion akin to multi-class (1vsAll) and multi-label (KvsAll) classification problems. Tabacof and Costabello [2019] showed that previous state-of-the-art KGE models generate uncalibrated predictions, i.e., probability estimates assigned to triples are unreliable. In

[2]We use the terminology introduced by Ruffinelli et al. [2019].

many recent works, a model calibration technique has been applied to obtain calibrated predictions [Balažević et al. 2019a,b; Demir et al. 2021; Demir and Ngomo 2021a; Dettmers et al. 2018].

A KGE model trained via 1vsAll or Kvsall can be interpreted as a probabilistic mapping $\hat{p} : \mathcal{E} \times \mathcal{R} \mapsto \mathbb{P}(\mathcal{Y})^{|\mathcal{E}|}$ with $\mathcal{Y} = \{0, 1\}$ representing the events that the respective entities apply (1) or not (0). Hence, the training information is in the simplest form rendered as probabilities $p(\mathtt{t} \mid \mathtt{h}, \mathtt{r}) := p(\mathbb{1}(\mathtt{t} \in \mathtt{r}(\mathtt{h})) \in \mathcal{Y} \mid \mathtt{h}, \mathtt{r}) = 1 \text{ s.t. } \forall(\mathtt{h}, \mathtt{r}, \mathtt{t}) \in \mathcal{G}$, e.g., the probability of the triple $(\mathtt{Michelle}, \mathtt{HasChild}, \mathtt{Malia}) \in \mathcal{G}$ would be set to 1. Then, Equation 1 can be rewritten as

$$L = -\frac{1}{|\mathcal{E}|} \sum_{\mathtt{t} \in \mathcal{E}} p(\mathtt{t} \mid \mathtt{h}, \mathtt{r}) \log \hat{p}(\mathtt{t} \mid \mathtt{h}, \mathtt{r}) + (1 - p(\mathtt{t} \mid \mathtt{h}, \mathtt{r})) \log(1 - \hat{p}(\mathtt{t} \mid \mathtt{h}, \mathtt{r})). \tag{2}$$

The calibration of $\hat{p}$ characterizes the convergence between the model confidence and actual accuracy, ideally matching each other [Guo et al. 2017]. Not only can overconfident but incorrect predictions result into harmful consequences, also too modest predictions can leave much potential unused. Hence, well-calibration can be seen as a second optimization goal in knowledge graph embedding learning. Training models with nondegenerate probability distributions —hard target values 1/0 — provokes overconfidence in predictions as the learner is urged to reproduce extreme probabilities [Tabacof and Costabello 2019]. To weaken this issue, a plethora of calibration means has been proposed, commonly by leveraging explicit calibration data, such as isotonic regression [Zadrozny and Elkan 2002], Bayesian binning [Naeini et al. 2015] or temperature scaling [Guo et al. 2017]. Besides calibration, model *robustness* against label noise further plays an essential role in making a embedding model applicable to real-world data. Large knowledge graphs observed in practice typically comprise erroneous relations, misleading the learner when used as training instance [Heindorf et al. 2017, 2016, 2019]. For instance, Albert Einstein has the type of https://dbpedia.org/ontology/Eukaryote that is defined a subclass of Fungus or https://dbpedia.org/page/Boo_(dog) has the type of Scientist and Dog.

3.3.1 Label Smoothing. Szegedy et al. [2016] designed the Label Smoothing technique (LS) that combines favorable properties in both quality respects discussed above. Not only can it be considered as an implicit calibration method [Müller et al. 2019], it has also shown improved robustness capabilities over the cross-entropy loss [Lukasik et al. 2020]. LS transforms the probability distributions p as defined above to less extreme distributions p^s by distributing a certain amount of probability mass among all other events, in its simplest formulation uniformly. To this end, let $\alpha \in (0, 1)$ be the smoothing parameter. Then, the smoothed training distributions are formally given by $p^s(\mathtt{t} \mid \mathtt{h}, \mathtt{r}) = 1 - \alpha/2 \; \forall(\mathtt{h}, \mathtt{r}, \mathtt{t}) \in \mathcal{G}$. These distributions p^s replace p in the cross-entropy loss formulation. Referring to the running example, label smoothing would assign $p^s(\mathtt{Malia} \mid \mathtt{Michelle}, \mathtt{HasChild}) = 1 - \alpha/2$ and $\alpha/2$ to the complementary event. By this, the model is no longer urged to reproduce probabilities approaching 1, resulting in less over-confidence.

3.3.2 Label Relaxation. Albeit LS undeniably improves generalization performances, it can be questioned from a data modelling perspective. The smoothed distribution p^s, by default determined by a uniform smoothing policy, is unlikely to match the exact underlying ground-truth probability, introducing a modeling bias that may distort the learning in a less extreme yet still unrealistic way. Coming back to our previous example, the smoothed target probability $p^s(\mathbb{1}(\mathtt{Malia} \notin \mathtt{HasChild}(\mathtt{Michelle})) \mid \mathtt{Michelle}, \mathtt{HasChild}) = \alpha/2$ seems to be implausible in case there is clear evidence for the truthfulness of the triple, and the learner would be incited to assign less probability on purpose.

As a generalization to mitigate such issues, so-called *label relaxation* (LR) [Lienen and Hüllermeier 2021] models the target as (credal) *set* $\mathcal{Q}_\alpha \subseteq \mathbb{P}(\mathcal{Y})$ of candidate probability distributions "sufficiently close" to the original distribution p, whereby the exact ground-truth distribution is assumed to be in $\mathcal{Q}_\alpha$. To construct such target sets, LR assigns a certain amount of plausibility $\alpha \in (0, 1]$ to other outcomes than the one observed in the training data. Transferred to KGE

learning, the parameter α can be interpreted as an upper probability for the event that the respective entity does not apply to a pair (h, r). Hence, the credal set $\mathcal{Q}_\alpha^{(\mathsf{h},\mathsf{r},\mathsf{t})}$ of the triple $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$ is given by

$$\mathcal{Q}_\alpha^{(\mathsf{h},\mathsf{r},\mathsf{t})} := \{p'(\cdot \mid \mathsf{h}, \mathsf{r}) \in \mathbb{P}(\mathcal{Y}) \mid p'(\mathsf{t} \mid \mathsf{h}, \mathsf{r}) \geq 1 - \alpha\}.$$

As a result, instead of committing to a particular distribution, the resulting target set $\mathcal{Q}_\alpha$ is less precisely but more reliably expressing the belief about the ground-truth probability, reducing the risk of entailing an undesirable bias in the target modelling. Relating to the ongoing example relation, the target probability p is expected to be $p(\texttt{Malia} \mid \texttt{Michelle}, \texttt{HasChild}) \in [1-\alpha, 1]$, i.e., it must not necessarily be the smoothed nor the degenerate distribution. In the following, we will drop the dependence of the target set on $(\mathsf{h}, \mathsf{r}, \mathsf{t})$, i.e., $\mathcal{Q}_\alpha := \mathcal{Q}_\alpha^{(\mathsf{h},\mathsf{r},\mathsf{t})}$, for the sake of brevity.

To optimize a probabilistic model provided such (weak) supervision, the learner is then trained by comparing $\hat{p}$ to the most plausible candidate $p' \in \mathcal{Q}_\alpha$ among the distributions in the target set in terms of the Kullback-Leibler divergence $D_{KL}(p||\hat{p}) = \sum_{y \in \mathcal{Y}} p(y) \log \hat{p}(y)$ based on the current model belief $\hat{p}$. Leveraging the framework of *optimistic superset learning* [Hüllermeier and Cheng 2015], the LR loss is formally defined as

$$L^*(\mathcal{Q}_\alpha, \hat{p}) = \min_{p' \in \mathcal{Q}_\alpha} D_{KL}\big(p' \mid\mid \hat{p}\big),$$

which simplifies to the closed-form

$$L^*(Q_\alpha, \hat{p}) = \begin{cases} 0 & \text{if } \hat{p} \in Q_\alpha \\ D_{KL}(p'||\hat{p}) & \text{otherwise,} \end{cases}$$

where $p'(\mathbb{1}(\mathsf{t} \in \mathsf{r}(\mathsf{h})) \mid \mathsf{h}, \mathsf{r}) = 1-\alpha$ and α otherwise. Roughly speaking, the consideration of p' as closest, and thus most plausible, distribution to $\hat{p}$ in the target set constitutes a form of optimism in the validity of $\hat{p}$. This loss instantiation has been proven to be convex and can be efficiently optimized. Results signal improved model calibration while retaining the generalization performance as achieved by LS [Lienen and Hüllermeier 2021; Lienen and Hüllermeier 2021].

3.4 Kronecker Product and Decomposition

For any matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $\mathbf{Y} \in \mathbb{R}^{m \times n}$, the Hadamard product $\mathbf{X} \circ \mathbf{Y}$ is defined as

$$\mathbf{X} \circ \mathbf{Y} = \begin{bmatrix} \mathbf{X}_{11}\mathbf{Y}_{11} & \dots & \mathbf{X}_{1n}\mathbf{Y}_{1n} \\ \vdots & \ddots & \vdots \\ \mathbf{X}_{m1}\mathbf{Y}_{m1} & \dots & \mathbf{X}_{mn}\mathbf{Y}_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}, \tag{3}$$

where $\mathbf{X}_{ij}$ **only interacts** with $\mathbf{Y}_{ij}$. For any matrix $\mathbf{Z} \in \mathbb{R}^{p \times q}$, the Kronecker Product (KP) $\mathbf{X} \otimes \mathbf{Z}$ is a block matrix:

$$\mathbf{X} \otimes \mathbf{Z} = \begin{bmatrix} \mathbf{X}_{11}\mathbf{Z} & \dots & \mathbf{X}_{1n}\mathbf{Z} \\ \vdots & \ddots & \vdots \\ \mathbf{X}_{m1}\mathbf{Z} & \dots & \mathbf{X}_{mn}\mathbf{Z} \end{bmatrix} \in \mathbb{R}^{mp \times nq}, \tag{4}$$

where **every element** of $\mathbf{X}$ **interacts with every element** of $\mathbf{Z}$. In contrast to the Hadamard product, the Kronecker product is not commutative, i.e., $\mathbf{X} \otimes \mathbf{Z} \neq \mathbf{Z} \otimes \mathbf{X}$ most commonly holds. For more details, we refer to [Graham 2018; Van Loan 2000]. Numerous works have shown that KP defined in Equation 4 can be effectively applied to decompose a large matrix into two smaller matrices [Cohen et al. 2019; Greenewald et al. 2013, 2016; Tahaei et al. 2021; Zhang et al. 2021]. In the Kronecker Decomposition (KD), a large weight matrix of $\mathbf{W} \in R^{mp \times nq}$ can be decomposed into any

$\mathbf{X} \in \mathbb{R}^{m_1 \times n_1}$ and $\mathbf{Z} \in \mathbb{R}^{\frac{mp}{m_1} \times \frac{nq}{n_1}}$. Different compression factors can be achieved through different shape configurations of smaller matrices. Tahaei et. al. [Tahaei et al. 2021] have shown that the following formulation can be effectively used to compute a linear transformation encoded in a weight matrix $\mathbf{W}$ via KD:

$$\Big(\mathbf{X} \otimes \mathbf{Z}\Big)x = \mathcal{V}\Big(\mathbf{Z}\, \mathcal{R}_{\frac{nq}{n_1} \times n_1}(x)\mathbf{X}^\top\Big), \tag{5}$$

where $x \in \mathbb{R}^{nq}$ represent input feature vector, $\mathcal{V} : \mathbb{R} \rightarrow \mathbb{R}^{mp}$ flattens an input matrix into a vector, $\mathcal{R}_{\frac{nq}{n_1} \times n_1}$ converts x to a $\frac{nq}{n_1}$ by n_1 matrix by dividing the vector to columns of size $\frac{nq}{n_1}$ and concatenating the resulting columns together [Lutkepohl 1997; Tahaei et al. 2021]. The computation defined in 5 reduces the number of floating point operations required to perform KD from $(2m_1m_2 - 1)n_1n2$ to $min((2n_2 - 1)m_2n_1 + (2n_1 - 1)m_2m_1, (2n_1 - 1)n_2m_1 + (2n_2 - 1)m_2m_1)$, where $n_1 = \frac{mp}{m1}$ and $n_2 = \frac{nq}{n_1}$ [Tahaei et al. 2021]. In the next section, we elucidate our methodology on applying KD in KGE.

4 METHODOLOGY

Previous works have shown that pre-trained overparameterized language models can be effectively decomposed into smaller weight matrices [Jiao et al. 2019; Rashid et al. 2021; Sanh et al. 2019; Sun et al. 2020]. Particularly, the Kronecker Decomposition (KD) elucidated in Section 3.4 leads a significant reduction in the number of parameters with at most a mild cost of predictive accuracy [Edalati et al. 2021; Tahaei et al. 2021]. Analogous to the aforementioned two works, findings of Zhang et al. [2021] and Wu [2016] suggest that training a neural network via KD on weight matrices results in a significant parameter efficiency. We are interested in learning compressed KGE by applying KD during training. We aim to design a generic technique that can be applied in existing knowledge graph embedding (KGE) models to reduce the number of explicitly stored parameters while retaining their expressiveness. Importantly, through KD on embedding matrices, we aim to capture interactions within an embedding vector without requiring additional parameters. This is expected to encourage parameter reuse and reduces redundancy in model's parameters [Huang et al. 2017].

4.1 Kronecker Decompression for Knowledge Graph Embeddings

Most KGE models are designed as a parametrized scoring function (say DistMult [Yang et al. 2015]) $\phi_\Theta : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightarrow \mathbb{R}$ that maps an input triple $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$ to a scalar value that is mapped to the unit interval using the sigmoid/logistic function. This normalized scalar value is interpreted to reflect the likelihood of $(\mathsf{h}, \mathsf{r}, \mathsf{t})$ is true which is denoted with $\hat{p}(\mathsf{t} \mid \mathsf{h}, \mathsf{r})$ in Section 3.3. Θ denotes the parameters of ϕ that consists of an entity embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d}$, a relation embedding matrix $\mathbf{R} \in \mathbb{R}^{|\mathcal{R}| \times d}$, and other trainable parameters, such as affine transformation over input embeddings, convolutions, batch normalization or instance normalization. Assume that $\phi =:$ DistMult and $\Theta := \{\mathbf{E}, \mathbf{R}\}$, for a given $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$, a score is computed as

$$\text{DistMult}(\mathsf{h}, \mathsf{r}, \mathsf{t}) = \mathbf{h} \circ \mathbf{r} \cdot \mathbf{t}. \tag{6}$$

In Equation 6, a triple score is computed thorough elementwise interactions between 3 d-dimensional real-valued embedding vectors, e.g., given $d = 2$ and, $a, b, c, d, e, f \in \mathbb{R}$, a triple score is $[a\ \ b] \circ [c\ \ d] \cdot [e\ \ f] = ace + bdf$. During training, the gradient of the loss function (see Equation 2) w.r.t. 3 d-dimensional embedding vectors $\mathbf{h}, \mathbf{r}, \mathbf{t}$ is computed. The gradient of the loss w.r.t. the first item a in $\mathbf{h}$ is computed in two steps: the gradient of the loss. w.r.t. the prediction is computed and distributed over the addition operation and the elementwise multiplication via ce. Yet, the interaction between (a, b) as well as between (a, a) are ignored, although a and b constituted $\mathbf{h}$ together. This stems from computing a scalar value via elementwise operations. These ignored interactions can be captured through

applying KD on embedding vectors as follows

$$\begin{aligned}[aa\ ab\ ab\ bb] \circ [cc\ cd\ cd\ dd] \cdot [ee\ ef\ ef\ ff] = aacce \\ + 2(abcdef) \\ + bbddff\ .\end{aligned} \tag{7}$$

Through the middle term in Equation 7, interactions between (a, b) and (a, a) are incorporated without requiring additional parameters. Hence, we argue that DistMult defined in Equation 6 is less expressive than its KD variation defined in Equation 7. Consequently, if the both models are trained properly, the latter model is expected to perform at least as well as the former model in the link prediction problem. Importantly, the number of possible interactions within an embedding vector grows by $(d+1)^2$ when d grows by 1. Although these interactions are expected to encourage *feature reuse* and *reduce redundancy* in model's parameters, the computation of KP on embedding vectors may become a bottleneck due to high computationally complexity of KD. In this setting, $\mathbf{E}$ and $\mathbf{R}$ can be seen as compressed embeddings of entities and relations. Hence, given a triple $(\mathsf{h}, \mathsf{r}, \mathsf{t})$, their compressed embeddings are retrieved and a triple score is computed via their decompressed embeddings. Our technique can be readily applied on many multiplicative based KGE models. For the numerical stability, the batch normalisation or layer normalization can be applied to reduce the dependence of gradients on the scale of embedding vectors as these normalization techniques have beneficial effect on the gradient flow through models [Ba et al. 2016; Ioffe and Szegedy 2015].

4.2 Kronecker Decomposition for Relation Embeddings

KD on relation embeddings of DistMult can be applied as

$$\text{KD-Rel-DistMult}(\mathsf{h}, \mathsf{r}, \mathsf{t}) = \mathbf{h} \circ (\mathbf{r} \otimes \mathbf{r}) \cdot \mathbf{t}, \tag{8}$$

where $\mathbf{h}, \mathbf{t} \in \mathbf{E} : \mathbf{E} \in \mathbb{R}^d$, and $\mathbf{r} \in \mathbf{R} : \mathbf{R} \in \mathbb{R}^{\sqrt{d}}$. The parameter gain can be computed as $(d - \sqrt{d}) \times (|\mathcal{R}|)$. Note that scores of triples are still computed based on 3 d-dimensional embedding vectors. As the size of the input graph and the number of relations grow, the parameter gain becomes more tangible.

4.3 Kronecker Decomposition of Embeddings for 1vsAll

Most KGs contain more entities than relations (see Table 1). Hence, the potential parameter gain of applying KD on entity embeddings is expectedly larger than applying KD on relation embeddings. Yet, applying KD on tail entities embeddings in 1vsAll or KvsAll increase the runtime and memory requirements as these training strategies require considering all entities to compute an incurred loss for a single prediction (see Section 3.2). To alleviate this limitation, KD can be applied as

$$\text{KD-DistMult}(\mathsf{h}, \mathsf{r}, \mathsf{t}) = \sum_{i}^{d} R\big((\mathbf{h} \otimes \mathbf{h}) \circ (\mathbf{r} \otimes \mathbf{r})\big)_i \cdot \mathbf{t}, \tag{9}$$

where $R(\cdot)$ reshapes a d^2 dimensional vector into d by d matrix. The matrix vector product of the resulting vector and $\mathbf{t}$ is summed to obtain a score for an input triple $(\mathsf{h}, \mathsf{r}, \mathsf{t})$.

In Section 4.2 and 4.3, we elucidated applying KD on embedding vectors. Yet, KD can be also applied to decompose linear transformation weight matrices in feed-forward and convolutional based embedding models during training.

Table 1. Overview of datasets.

Dataset	$\|\mathcal{E}\|$	$\|\mathcal{R}\|$	$\|\mathcal{G}^{\text{Train}}\|$	$\|\mathcal{G}^{\text{Val.}}\|$	$\|\mathcal{G}^{\text{Test}}\|$
UMLS	136	93	10,432	1304	1965
KINSHIP	105	51	17,088	2136	3210

5 EXPERIMENTAL SETUP

5.1 Training and Optimization

We trained approaches with the 1vsAll training strategy (see Section 3.2) as commonly done in the literature [Lacroix et al. 2018; Ruffinelli et al. 2019]. All models are trained with the ADAM optimizer [Kingma and Ba 2014] and minimize the cross entropy loss function (see Equation 2). We use the same loss function for all approaches on all datasets as Mohamed et al. [2019] previously showed that generalization performance of KGE models can be significantly influenced by the choice of the loss function. Moreover, we apply the batch normalization [Ioffe and Szegedy 2015] to facilitate numerical stability and accelerate convergence during training. We trained the model for 1000 epochs with a learning rate of .01 and a batch size of 1024. We further optimized the embedding vector sizes in $\{4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 289, 324, 361, 400\}$ using a grid search. For label smoothing and label relaxation, we consider $\alpha \in \{.1, .2\}$. This results in training models with soften target values, i.e., $0 < \mathbf{y}^{(i)} < 1$. Through large parameter sweep in the embedding vector size, we aim to obtain a fine-grained performance analysis. As elucidated in Section 4, we hypothesize that the impact of learning compressed embeddings becomes more tangible as the size of the embedding vector increases. Importantly, we also share test and train performance of all operations with all configurations. By doing so, we aim to quantify the impact of applying label smoothing and label relaxation during training and testing separately. We do not apply an explicit regularization (e.g. L1, L2 regularization or the Dropout technique [Srivastava et al. 2014]) as regularization techniques may not allow to observe any regularization effect of label smoothing and label relaxation.

5.2 Datasets

We evaluated our proposed models on standard link prediction benchmark datasets (KINSHIP and UMLS datasets) [Trouillon and Nickel 2017]. The Kinships knowledge graph describes the 26 different kinship relations of the Alyawarra tribe and the unified medical language system (UMLS) knowledge graph describes 135 medical entities via 49 relations describing [Trouillon and Nickel 2017]. Note that we omitted WN18RR, FB15K-237, and YAGO3-10 from our experiments for two reasons: First, we aim to conduct experiments to quantify the impact of Kronecker Decomposition in a fine-grained many unique configurations with very large number of epochs (1000). KINSHIP and UMLS datasets are considerably smaller datasets compared to WN18RR, FB15K-237, and YAGO3-10. Hence, training three models with 21 unique configurations for 1000 epochs (see Section 5.1) on WN18RR, FB15K-237, and YAGO3-10 can take up to a month. Moreover, two recent works showed that these datasets involve entities on the validation and test data splits that do not occur in the train split [Broscheit et al. 2020; Demir and Ngomo 2021b].

5.3 Evaluation metrics and Baseline Selection

We use the standard metrics *filtered* Mean Reciprocal Rank (MRR) and hits at N (H@N) for link prediction [Balažević et al. 2019a,b; Dettmers et al. 2018]. We also evaluate link prediction performances of approaches with respect to number of

parameters. In 1vsAll or KvsAll, for each test triple (h, r, t), the score of (h, r, x) triples for all $x \in \mathcal{E}$ is computed. Based on these scores, the filtered ranking $rank_t$ of the triple having t is obtained. Then the MRR: $\frac{1}{|\mathcal{G}^{\text{test}}|} \sum_{(h,r,t) \in \mathcal{G}^{\text{test}}} \frac{1}{rank_t}$ is computed. Next, Hi@1, H@3, and H@10 are computed in literature [Balažević et al. 2019b; Dettmers et al. 2018; Ruffinelli et al. 2019]. The number of parameters consists of $|\mathbf{E}|$, $|\mathbf{R}|$, and all other trainable parameters. Moreover, we illustrate our methodology with DistMult for three reasons: (1) many recent KGE model can be seen as an effective extension of DistMult, i.e., the Hadamard product followed by an inner product of input embeddings (see Section 2). (2) Findings of Ruffinelli et al. [2019] indicate that DistMult perform competitive performance provided that it is properly trained. (3) DistMult requires less floating point operations than more sophisticated recent models. Hence, it can be trained in less time [Costabello et al. 2019; Valeriani 2020].

5.4 Implementation Details and Reproducibility

We built a hardware-agnostic knowledge graph embedding framework [3] based on PyTorch Lightning [Falcon et al. 2019] and DASK [Rocklin 2015]. In our experiments, the seed for the pseudo-random generator was fixed to 1. To alleviate the hardware requirements for the reproducibility of our results, we provide hyperparameter optimization, training and evaluation scripts along with pretrained models.

6 RESULTS

6.1 Standard Link Prediction

Table 2 reports link prediction results on benchmark datasets. Overall results indicate that applying KD on embedding matrices reduces the number of required parameters, while yielding a competitive performance. KD-DistMult outperforms DistMult and KD-Rel-DistMult in 9 out of 10 metrics in Table 2 with surprisingly less parameters. Specifically, KD-DistMult outperforms DistMult and KD-DistMult, while requiring **18.2**× and **11.4**× fewer number of parameters than DistMult and KD-Rel-DistMult on UMLS, respectively. Similarly, KD-DistMult requires **14.6**× and **14.2**× fewer number of parameters than DistMult and KD-Rel-DistMult on KINSHIP, respectively. After observing these results, we delved into the details of training process to validate the existence of overfitting. To this end, we evaluated each model on the training datasets and added the MRR and Hit@N scores in Table 2.

Table 2. Link prediction results on UMLS and KINSHIP. $|\Theta|$ denotes the number of parameters. Bold entries denote best results.

	UMLS					**KINSHIP**				
	$\|\Theta\|$	MRR	@1	@3	@10	$\|\Theta\|$	MRR	@1	@3	@10
DistMult	67,915	.517	.441	.536	.659	46,818	.568	.500	.593	.693
on training set		.995	.992	1.00	1.00		.919	.876	.952	.991
KD-Rel-DistMult	42,619	.531	.432	.584	**.684**	45,420	.562	.493	.588	.694
on training set		.996	.993	.999	1.00		.914	.867	.953	.992
KD-DistMult	**3,728**	**.541**	.447	**.598**	**.684**	**3,200**	**.599**	**.534**	**.631**	**.709**
on training set		.814	.704	.904	.989		.705	.572	.804	.950

These results suggest that (a) all models seem to suffer from overfitting and (b) increasing embedding vector size does not proportionally increase the expressiveness of the model. DistMult with a **59**% more number of parameters than

[3] https://github.com/dice-group/dice-embeddings

KD-Rel-DistMult does not lead to a significant change in MRR and Hit@N scores on the training datasets. These results also highlight the importance of applying extensive extensive hyperparameter optimization and model calibration to increase generalization performances.

6.2 Link Prediction with Model Calibration

To observe the impact of model calibration, we retrained models with Label Smoothing and Label Relaxation (see Section 3.3). Table 3 shows that performances of models are improved with model calibration on UMLS, whereas performances are not increased and in some circumstances decreased. Importantly, model calibration seem to help more on those models that suffer greatly from the overfitting. For instance, models seem to perform better without model calibration on KINSHIP, where the overfitting is less severe.

Table 3. Link prediction results on UMLS and KINSHIP with model calibration. Rows with on training set report the performances on the training dataset. $|\Theta|$, LS, LR denote the number of parameters, Label Smoothing and Label Relaxation, respectively. Bold entries denote best results.

	UMLS					**KINSHIP**				
	$\|\Theta\|$	MRR	@1	@3	@10	$\|\Theta\|$	MRR	@1	@3	@10
DistMult	67,915	.517	.441	.536	.658	46,818	.568	.499	.593	.693
on training set		.995	.992	1.00	1.00		.919	.876	.952	.991
with LS $\alpha = .1$		.568	.499	.602	.707		.515	.417	.563	.685
on training set		.996	.993	1.00	1.00		.918	.845	.954	.992
with LS $\alpha = .2$		.548	.475	.582	.690		.502	.398	.555	.667
on training set		.995	.992	.999	1.00		.916	.871	.952	.992
with LR $\alpha = .1$		.552	.436	.630	.723		.567	.499	.592	.694
on training set		.995	.992	1.00	1.00		.919	.875	.952	.992
with LR $\alpha = .2$		**.579**	**.487**	**.648**	.725		.567	.499	.592	.695
on training set		.995	.992	1.00	1.00		.917	.873	.952	.992
KD-Rel-DistMult	42,619	.531	.432	.584	.684	32,946	.556	.487	.580	.689
on training set		.996	.993	.999	1.00		.913	.865	.951	.992
with LS $\alpha = .1$		.565	.493	.611	.704		.500	.394	.555	.677
on training set		.996	.992	.999	1.00		.913	.866	.951	.991
with LS $\alpha = .2$		.592	.531	.617	.704		.504	.403	.556	.671
on training set		.994	.990	.999	1.00		.907	.855	.949	.991
with LR $\alpha = .1$		.543	.444	.602	.692		.555	.486	.577	.692
on training set		.996	.993	.999	1.00		.912	.864	.952	.992
with LR $\alpha = .2$		.562	.476	.600	.718		.555	.487	.576	.689
on training set		.993	.999	1.00	1.00		.912	.863	.952	.992
KD-DistMult	3,728	.541	.447	.598	.684	3,200	**.599**	**.534**	**.631**	.709
on training set		.814	.704	.904	.989		.665	.519	.774	.936
with LS $\alpha = .1$		.592	.511	.627	.751		.519	.382	**.621**	.705
on training set		.796	.677	.897	.984		.640	.491	.745	.928
with LS $\alpha = .2$		.572	.475	.623	**.754**		.486	.326	.605	.704
on training set		.781	.655	.888	.980		.629	.476	.730	.930

6.3 Link Prediction under Noise

We were interested to observe the impact of adding noisy triples into the training dataset in the link prediction task, since many real-world KGs contains noisy triples. To this end, we add 10% noise in the training splits and evaluate models. Table 4 suggest that 10% noise in the input data decrease the standard DistMult model by absolute 7% in MRR on KINSHIP, whereas performance of KD-Rel-DistMult and KD-DistMult are more robust against the input noise. Surprisingly, KD-Rel-DistMult reaches the highest MRR, Hit@1 and Hit@3 scores throughout our experiments with 10% noisy data. Bishop [Bishop 1995] showed that the addition of noise to the numerical input training data lead to significant improvements in generalization performance. Our results signal that adding additional noise in the structured data may have the similar effect.

Table 4. Link prediction results on noisy UMLS and KINSHIP. $|\Theta|$ denotes the number of parameters. Bold entries denote best results.

	UMLS					**KINSHIP**				
	$\|\Theta\|$	MRR	@1	@3	@10	$\|\Theta\|$	MRR	@1	@3	@10
DistMult	23,500	.448	.319	.484	.741	16,200	.523	.452	.538	.665
	28,435	.470	.382	.497	.677	19,602	.512	.444	.523	.655
	33,840	.423	.335	.411	.621	23,328	.508	.439	.517	.658
	39,715	.518	.418	.556	.741	27,378	.510	.440	.524	.658
	46,060	.489	.355	.586	.738	31,752	.512	.438	.529	.668
	52,875	.465	.339	.500	.677	36,450	.518	.445	.534	.671
	60,160	.422	.340	.413	.591	41,472	.520	.447	.536	.679
	67,915	.485	.396	.497	.699	46,818	.517	.443	.534	.679
	76,140	.532	.436	.562	**.746**	52,488	.523	.447	.547	.674
	84,835	.448	.341	.467	.695	58,482	.529	.453	.551	.684
	94,000	.466	.343	.541	.662	64,800	.529	.454	.556	.685
KD-Rel-DistMult	15,130	.521	.428	.534	.730	11,610	.540	.477	.554	.674
	18,205	.550	.467	.578	.690	13,992	.516	.449	.528	.655
	21,564	.456	.349	.499	.676	16,596	.507	.442	.514	.644
	25,207	.469	.338	.591	.718	19,422	.500	.439	.502	.631
	29,134	.515	.422	.561	.701	22,470	.503	.437	.511	.645
	33,345	.580	.500	.617	.701	25,740	.503	.436	.509	.647
	37,840	.526	.432	.544	.718	29,232	.506	.438	.515	.644
	42,619	.567	.493	.607	.693	32,946	.505	.431	.519	.655
	47,682	.439	.322	.482	.678	36,882	.511	.440	.529	.661
	53,029	.554	.490	.574	.671	41,040	.511	.441	.524	.657
	58,660	**.583**	**.503**	**.632**	.728	45,420	.514	.442	.531	.666
KD-DistMult	2,330	.461	.339	.527	.651	1,600	.562	.487	.592	.699
	2,563	.331	.233	.343	.496	1,760	.573	.501	.609	**.703**
	2,796	.376	.252	.448	.622	1,920	.567	.497	.599	.696
	3,029	.346	.254	.370	.506	2,080	.576	.505	.604	.702
	3,262	.374	.274	.391	.541	2,240	.567	.498	.589	**.703**
	3,495	.393	.259	.480	.559	2,400	.569	.502	.597	.690
	3,728	.452	.329	.529	.626	2,560	.572	.503	.603	.697
	3,961	.374	.276	.385	.655	2,720	.566	.501	.590	.694
	4,427	.358	.247	.376	.589	2,880	.578	.509	.607	**.703**
	4,194	.545	.438	.596	.744	3,040	.584	.520	.609	.698
	4,660	.366	.258	.406	.541	3,200	**.586**	**.521**	**.612**	.612
avg. DistMult	55,225	.470	.364	.501	.691	38,070	.518	.446	.535	.670
avg. KD-Rel-DistMult	34,765	**.524**	**.431**	**.565**	**.700**	26,850	.510	.443	.521	.653
avg. KD-DistMult	**3,495**	.398	.287	.441	.594	**2,400**	**.573**	**.504**	**.601**	**.691**

6.4 Parameter Analysis

Table 5 and Table 6 report performances with a wide range of embedding vector sizes. Overall, our results corroborate our hypothesis, namely, as the size of embedding vectors decreases, benefits of applying KD becomes less tangible. More specifically, Table 5 suggests that as $|\Theta|$ grows KD-Rel-DistMult and KD-DistMult perform quite well compared to DistMult on the both benchmark datasets.

Table 5. Link prediction results on UMLS and KINSHIP with the highest half of the the parameter sweep in the number of parameters $|\Theta|$. Bold entries denote best results.

	UMLS					**KINSHIP**				
	$\|\Theta\|$	MRR	@1	@3	@10	$\|\Theta\|$	MRR	@1	@3	@10
DistMult	28,435	.430	.346	.430	.650	19,602	.556	.487	.585	.687
	33,840	.439	.357	.455	.545	23,328	.563	.494	.589	.695
	39,715	.425	.344	.435	.596	27,378	.562	.493	.587	.695
	46,060	.484	.419	.480	.585	31,752	.563	.494	.588	.699
	52,875	.507	.439	.525	.646	36,450	.565	.498	.589	.693
	60,160	.459	.363	.487	.646	41,472	.567	.498	.598	.700
	67,915	.517	.441	.536	.659	46,818	.568	.500	.593	.693
	76,140	.471	.374	.485	.677	52,488	.548	.460	.596	.697
	84,835	.454	.354	.496	.632	58,482	.567	.498	.598	.697
	94,000	.436	.353	.439	.587	64,800	.563	.493	.591	.697
KD-Rel-DistMult	18,205	.544	**.475**	.576	.665	13,992	.546	.478	.568	.684
	21,564	.532	.426	.576	**.735**	16,596	.544	.462	.580	.691
	25,207	.455	.368	.473	.639	19,422	.546	.476	.567	.687
	29,134	.477	.373	.508	.661	22,470	.544	.472	.571	.688
	33,345	**.525**	.452	.548	.642	25,740	.548	.475	.576	.691
	37,840	.516	.441	.549	.669	29,232	.553	.484	.578	.689
	42,619	.531	.432	**.584**	.684	32,946	.556	.487	.580	.689
	47,682	.483	.397	.519	.640	36,882	.390	.155	.578	.694
	53,029	**.525**	.447	.554	.643	41,040	.559	.488	.587	.693
	58,660	.435	.348	.438	.598	45,420	.562	.493	.588	.694
KD-DistMult	2,563	.357	.275	.374	.453	1,760	.556	.487	.581	.686
	2,796	.403	.276	.468	.637	1,920	.579	.509	.613	.702
	3,029	.354	.271	.375	.492	2,080	.567	.493	.606	.698
	3,262	.357	.267	.372	.470	2,240	.577	.511	.605	.701
	3,495	.361	.252	.382	.618	2,400	.572	.501	.607	.706
	3,728	.541	.447	.598	.684	2,560	.591	.526	.621	.703
	3,961	.407	.282	.474	.645	2,720	.587	.521	.619	.701
	4,194	.379	.302	.392	.461	2,880	.591	.526	.620	.708
	4,427	.382	.285	.390	.605	3,040	**.599**	**.535**	.628	.707
	4,660	.505	.397	.581	.697	3,200	**.599**	.534	**.631**	**.709**
avg. DistMult	58,397	.462	.379	.477	.622	40,257	.562	.491	.591	.695
avg. KD-Rel-DistMult	36,728	**.502**	**.416**	**.532**	**.658**	28,374	.535	.447	.577	.690
avg. KD-DistMult	**3,611**	.405	.305	.441	.576	**2,480**	**.582**	**.514**	**.613**	**.702**

Table 6. Link prediction results on UMLS and KINSHIP with the lowest half of the the parameter sweep in the number of parameters |Θ|. Bold entries denote best results.

	UMLS					KINSHIP				
	|Θ|	MRR	@1	@3	@10	|Θ|	MRR	@1	@3	@10
DistMult	940	.475	.378	.501	.669	648	.450	.291	.559	.686
	2,115	.550	.472	.579	.668	1,458	.582	.516	.607	.707
	3,760	.493	.336	.633	.757	2,592	.607	.554	.634	.712
	5,875	.584	.498	.626	.749	4,050	.613	.554	.640	.714
	8,460	.488	.363	.567	.681	5,832	.612	.555	.631	.709
	11,515	.409	.326	.417	.601	7,938	.597	.533	.625	.710
	15,040	.473	.378	.502	.679	10,368	.581	.519	.605	.699
	19,035	.548	.476	.590	.709	13,122	.560	.492	.584	.690
	23,500	.489	.409	.518	.598	16,200	.549	.480	.570	.683
KD-Rel-DistMult	754	.385	.220	.550	.683	546	.508	.428	.532	.679
	1,557	.537	.444	.582	.692	1,152	.382	.137	.581	.694
	2,644	.531	.453	.559	.646	1,980	.579	.508	.609	.710
	4,015	.547	.466	.579	.694	3,030	.591	.523	.621	.712
	5,670	.457	.367	.487	.635	4,302	.599	.536	.620	.709
	7,609	.560	.431	.657	.742	5,796	.592	.529	.615	.709
	9,832	.555	.477	.576	.716	7,512	.581	.517	.607	.703
	12,339	.507	.415	.550	.679	9,450	.575	.512	.593	.698
	15,130	.565	.503	.586	.687	11,610	.560	.493	.581	.693
KD-DistMult	466	.354	.266	.384	.539	320	.358	.358	.400	.528
	699	.321	.215	.309	.564	480	.428	.380	.421	.508
	932	.347	.252	.357	.643	640	.492	.422	.505	.643
	1,165	.360	.260	.381	.503	800	.524	.454	.547	.660
	1,398	.335	.225	.347	.620	960	.527	.457	.553	.655
	1,631	.375	.225	.493	.651	1,120	.550	.483	.573	.676
	1,864	.357	.248	.362	.686	1,280	.575	.503	.607	.708
	2,097	.355	.244	.373	.648	1,440	.572	.500	.603	.707
	2,330	.533	.433	.588	.737	1,600	.557	.488	.587	.689
avg. DistMult	10,026	.501	.404	.548	.679	6,912	**.572**	**.499**	**.606**	**.701**
avg. KD-Rel-DistMult	6,616	**.516**	**.420**	**.569**	**.686**	5,042	.552	.465	.595	**.701**
avg. KD-DistMult	**1,398**	.371	.263	.399	.621	**960**	.509	.449	.534	.642

7 DISCUSSION

We conjecture that all KGE models may benefit from an extensive hyperparameter optimization. Yet, here, we were interested in relative link prediction performances under optimizing solely the embedding vector size. We aimed to observe possible benefits of learning compressed embeddings in link prediction task. Table 2 and Table 6 suggest that benefits of learning compressed embeddings becomes more beneficial as the embedding vector size grows. Increasing the size of embedding vectors in DistMult does not decrease training loss as well as MRR and Hit@N scores on training datasets. This may stem from the fact that increasing the size of embeddings increases redundancy in the parameters, hence does not improve training and test performance of DistMult. Yet, Table 3 shows that model calibration consistently improved generalization performances on UMLS, while performances on KINSHIP are not improved as such. In our experiments, performance of KD-Rel-DistMult and KD-DistMult suggest that learning compressed knowledge graph

embeddings does not only lead to a competitive and sometimes superior performance but also decrease the the need of overparameterization. We argue that a comprehensive study including many recent state-of-the-art knowledge graph embedding models on benchmark datasets from different domains is needed to further analyse the benefits of learning compressed embeddings via KD.

8 CONCLUSION

Most KGE models learn embeddings of entities and relations tailored towards the link prediction problem. Recent results signal an ever increasing predictive ability with the cost of over-parameterization and computationally complexity. Here, we designed a generic technique based on the Kronecker Decomposition (KD) to find a remedy for the former problem. Through KD, interactions within an embedding vector can be incorporated in the learning process without requiring requiring additional parameters. This encourages feature reuse and reduce redundancy in the embedding vectors. We showed that our technique can be readily applied in existing embedding models. Our experiments suggest that applying KD on entity and relation embeddings during training makes models more robust against overfitting and noise in input knowledge graphs. Benefits of KD becomes more tangible as the size of the input vector grows.

ACKNOWLEDGMENTS

This work has been supported by the German Federal Ministry of Education and Research (BMBF) within the project DAIKIRI under the grant no 01IS19085B and by the the German Federal Ministry for Economic Affairs and Energy (BMWi) within the project RAKI under the grant no 01MD19012B.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

Ivana Balažević, Carl Allen, and Timothy M Hospedales. 2019a. Hypernetwork knowledge graph embeddings. In *International Conference on Artificial Neural Networks*. Springer, 553–565.

Ivana Balažević, Carl Allen, and Timothy M Hospedales. 2019b. Tucker: Tensor factorization for knowledge graph completion. *arXiv preprint arXiv:1901.09590* (2019).

Chris M Bishop. 1995. Training with noise is equivalent to Tikhonov regularization. *Neural computation* 7, 1 (1995), 108–116.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*. 2787–2795.

Samuel Broscheit, Daniel Ruffinelli, Adrian Kochsiek, Patrick Betz, and Rainer Gemulla. 2020. LibKGE - A Knowledge Graph Embedding Library for Reproducible Research. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. 165–174. https://www.aclweb.org/anthology/2020.emnlp-demos.22

Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

Israel Cohen, Jacob Benesty, and Jingdong Chen. 2019. Differential Kronecker product beamforming. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 27, 5 (2019), 892–902.

Luca Costabello, Sumit Pai, Chan Le Van, Rory McGrath, Nick McCarthy, and Pedro Tabacof. 2019. AmpliGraph: a Library for Representation Learning on Knowledge Graphs. https://doi.org/10.5281/zenodo.2595043

Caglar Demir, Diego Moussallem, Stefan Heindorf, and Axel-Cyrille Ngonga Ngomo. 2021. Convolutional Hypercomplex Embeddings for Link Prediction. In *Asian Conference on Machine Learning*. PMLR, 656–671.

Caglar Demir and Axel-Cyrille Ngonga Ngomo. 2021a. Convolutional Complex Knowledge Graph Embeddings. In *Eighteenth Extended Semantic Web Conference - Research Track*. https://openreview.net/forum?id=6T45-4TFqaX

Caglar Demir and Axel-Cyrille Ngonga Ngomo. 2021b. Out-of-Vocabulary Entities in Link Prediction. *arXiv preprint arXiv:2105.12524* (2021).

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Ali Edalati, Marzieh Tahaei, Ahmad Rashid, Vahid Partovi Nia, James J Clark, and Mehdi Rezagholizadeh. 2021. Kronecker Decomposition for GPT Compression. *arXiv preprint arXiv:2110.08152* (2021).

Jeffrey Scott Eder. 2012. Knowledge graph based search system. US Patent App. 13/404,109.

William Falcon et al. 2019. Pytorch lightning. *GitHub. Note: https://github. com/PyTorchLightning/pytorch-lightning* 3 (2019), 6.

Vladimir Gligorijević, P Douglas Renfrew, Tomasz Kosciolek, Julia Koehler Leman, Daniel Berenberg, Tommi Vatanen, Chris Chandler, Bryn C Taylor, Ian M Fisk, Hera Vlamakis, et al. 2021. Structure-based protein function prediction using graph convolutional networks. *Nature communications* 12, 1 (2021), 1–14.

Alexander Graham. 2018. *Kronecker products and matrix calculus with applications.* Courier Dover Publications.

Kristjan Greenewald, Theodoros Tsiligkaridis, and Alfred O Hero. 2013. Kronecker sum decompositions of space-time data. In *2013 5th IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*. IEEE, 65–68.

Kristjan Greenewald, Edmund Zelnio, and Alfred Hero Hero. 2016. Robust SAR STAP via Kronecker decomposition. *IEEE Trans. Aerospace Electron. Systems* 52, 6 (2016), 2612–2625.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. 2017. On calibration of modern neural networks. In *International Conference on Machine Learning*. PMLR, 1321–1330.

Stefan Heindorf, Martin Potthast, Hannah Bast, Björn Buchhold, and Elmar Haussmann. 2017. WSDM cup 2017: Vandalism detection and triple scoring. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. 827–828.

Stefan Heindorf, Martin Potthast, Benno Stein, and Gregor Engels. 2016. Vandalism detection in wikidata. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 327–336.

Stefan Heindorf, Yan Scholten, Gregor Engels, and Martin Potthast. 2019. Debiasing vandalism detection models at wikidata. In *The World Wide Web Conference*. 670–680.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, et al. 2020. Knowledge graphs. *arXiv preprint arXiv:2003.02320* (2020).

Sara Hooker. 2021. The hardware lottery. *Commun. ACM* 64, 12 (2021), 58–65.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

Eyke Hüllermeier and Weiwei Cheng. 2015. Superset Learning Based on Generalized Loss Minimization. In *Proceedings of the Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*. 260–275.

Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S Yu. 2020. A Survey on Knowledge Graphs: Representation, Acquisition and Applications. *arXiv preprint arXiv:2002.00388* (2020).

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351* (2019).

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

Bhushan Kotnis and Vivi Nastase. 2017. Analysis of the impact of negative sampling on link prediction in knowledge graphs. *arXiv preprint arXiv:1708.06816* (2017).

Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. 2018. Canonical tensor decomposition for knowledge base completion. *arXiv preprint arXiv:1806.07297* (2018).

Julian Lienen and Eyke Hüllermeier. 2021. Credal Self-Supervised Learning. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems, NeurIPS*, Vol. 34. Curran Associates, Inc., 14370–14382.

Julian Lienen and Eyke Hüllermeier. 2021. From Label Smoothing to Label Relaxation. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI*. 8583–8591.

Michal Lukasik, Srinadh Bhojanapalli, Aditya Krishna Menon, and Sanjiv Kumar. 2020. Does label smoothing mitigate label noise?. In *Proceedings of the 37th International Conference on Machine Learning, ICML*. 6448–6458.

Helmut Lutkepohl. 1997. Handbook of matrices. *Computational statistics and Data analysis* 2, 25 (1997), 243.

Sameh K Mohamed, Vít Novácek, Pierre-Yves Vandenbussche, and Emir Muñoz. 2019. Loss Functions in Knowledge Graph Embedding Models. *DL4KG@ ESWC* 2377 (2019), 1–10.

Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. 2019. When does label smoothing help?. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems, NeurIPS*. 4696–4705.

Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. 2015. Obtaining Well Calibrated Probabilities Using Bayesian Binning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2901–2907.

Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. 2017. A novel embedding model for knowledge base completion based on convolutional neural network. *arXiv preprint arXiv:1712.02121* (2017).

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2015. A review of relational machine learning for knowledge graphs. *Proc. IEEE* 104, 1 (2015), 11–33.

Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. 2016. Holographic embeddings of knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30.

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A Three-Way Model for Collective Learning on Multi-Relational Data.. In *ICML*, Vol. 11. 809–816.

Peyman Passban, Yimeng Wu, Mehdi Rezagholizadeh, and Qun Liu. 2020. Alp-kd: Attention-based layer projection for knowledge distillation. *arXiv preprint arXiv:2012.14022* (2020).

Ahmad Rashid, Vasileios Lioutas, and Mehdi Rezagholizadeh. 2021. MATE-KD: Masked adversarial TExt, a companion to knowledge distillation. *arXiv preprint arXiv:2105.05912* (2021).

Matthew Rocklin. 2015. Dask: Parallel Computation with Blocked algorithms and Task Scheduling. In *Proceedings of the 14th Python in Science Conference*, Kathryn Huff and James Bergstra (Eds.). 130 – 136.

Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. 2019. You CAN teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984* (2020).

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2818–2826.

Pedro Tabacof and Luca Costabello. 2019. Probability calibration for knowledge graph embedding models. *arXiv preprint arXiv:1912.10000* (2019).

Marzieh S Tahaei, Ella Charlaix, Vahid Partovi Nia, Ali Ghodsi, and Mehdi Rezagholizadeh. 2021. KroneckerBERT: Learning Kronecker Decomposition for Pre-trained Language Models via Knowledge Distillation. *arXiv preprint arXiv:2109.06243* (2021).

Théo Trouillon and Maximilian Nickel. 2017. Complex and holographic embeddings of knowledge graphs: a comparison. *arXiv preprint arXiv:1707.01475* (2017).

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*. 2071–2080.

Angelica Sofia Valeriani. 2020. Runtime Performances Benchmark for Knowledge Graph Embedding Methods. *arXiv preprint arXiv:2011.04275* (2020).

Charles F Van Loan. 2000. The ubiquitous Kronecker product. *Journal of computational and applied mathematics* 123, 1-2 (2000), 85–100.

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. 2017. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering* 29, 12 (2017), 2724–2743.

Jia-Nan Wu. 2016. Compression of fully-connected layer in neural network by kronecker product. In *2016 Eighth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE, 173–179.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*.

Bianca Zadrozny and Charles Elkan. 2002. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 694–699.

Aston Zhang, Yi Tay, Shuai Zhang, Alvin Chan, Anh Tuan Luu, Siu Cheung Hui, and Jie Fu. 2021. Beyond Fully-Connected Layers with Quaternions: Parameterization of Hypercomplex Multiplications with $1/n$ Parameters. *arXiv preprint arXiv:2102.08597* (2021).

Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. 2019. Quaternion knowledge graph embeddings. In *Advances in Neural Information Processing Systems*. 2731–2741.