Knowledge Graph Embedding Compression

Mrinmaya Sachan
Toyota Technological Institute at Chicago
mrinmaya@ttic.edu

Abstract

Knowledge graph (KG) representation learning techniques that learn continuous embeddings of entities and relations in the KG have become popular in many AI applications. With a large KG, the embeddings consume a large amount of storage and memory. This is problematic and prohibits the deployment of these techniques in many real world settings. Thus, we propose an approach that compresses the KG embedding layer by representing each entity in the KG as a vector of discrete codes and then composes the embeddings from these codes. The approach can be trained end-to-end with simple modifications to any existing KG embedding technique. We evaluate the approach on various standard KG embedding evaluations and show that it achieves 50-1000x compression of embeddings with a minor loss in performance. The compressed embeddings also retain the ability to perform various reasoning tasks such as KG inference.

1 Introduction

Knowledge graphs (KGs) are a popular way of storing world knowledge, lending support to a number of AI applications such as search (Singhal, 2012), question answering (Lopez et al., 2013; Berant et al., 2013) and dialog systems (He et al., 2017; Young et al., 2018). Typical KGs are huge, consisting of millions of entities and relations.

With the growth in use of KGs, researchers have explored ways to learn better representations of KGs in order to improve generalization and robustness in downstream tasks. In particular, there has been interest in learning embeddings of KGs in continuous vector spaces (Bordes et al., 2011, 2013; Socher et al., 2013). KG embedding approaches represent entities as learnable continuous vectors while each relation is modeled as an operation in the same space such as translation, projection, etc. (Bordes et al., 2013; Wang et al., 2014; Lin et al., 2015; Ji et al., 2015). These approaches give us a way to perform reasoning in KGs with simple numerical computation in continuous spaces.

Despite the simplicity and wide-applicability of KG embedding approaches, they have a few key issues. A major issue is that the number of embedding parameters grow linearly with the number of entities. This is challenging when we have millions or billions of entities in the KG, especially when there are a lot of sparse entities or relations in the KG. There is a clear redundancy in the continuous parameterization of embeddings given that many entities are actually similar to each other. This over-parameterization can lead to a drop in performance due to overfitting in downstream models. The large memory requirement of continuous representations also prevents models that rely on them from being deployed on modest user-facing computing devices such as mobile phones.

To address this issue, we propose a coding scheme that replaces the traditional KG embedding layer by representing each entity in the KG with a K-way D dimensional code (KD code) (van den Oord et al., 2017; Chen et al., 2018; Chen and Sun, 2019). Each entity in the KG is represented as a sequence of D codes where each code can take values in $\{1 \ldots K\}$. The codes for each entity are learnt in such a way that they capture the semantics and the relational structure of the KG – i.e., the codes that represent similar or related entities are typically also similar[1]. The coding scheme is much more compact than traditional KG embedding schemes.

We learn the discrete codes for entities using an autoencoder style model which learns a discretization function that maps continuous entity representations to discrete codes and a reverse-discretization function that maps the discrete codes

[1]For example, *Barack Obama* = "2-1-3-3" and *Michelle Obama* = "2-1-3-2" (for $D = 4$ and $K = 3$)

Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 2681–2691

back to continuous entity representations. The discretization and reverse-discretization functions are jointly learnt end-to-end. The inherent discreteness of the representation learning problem poses several learning issues. We tackle these issues by resorting to the straight-through estimator (Bengio et al., 2013) or the tempering softmax (Maddison et al., 2016; Jang et al., 2016) and using guidance from existing KG embeddings to smoothly guide learning of the discrete representations.

We evaluate our approach on various standard KG embedding evaluations and we find that we can massively reduce the size of the KG embedding layer while suffering only a minimal loss in performance (if at all). We show that the proposed approach for learning discrete KG representations leads to a good performance in the task of link prediction (cloze entity prediction) as well as in the task of KG reasoning and inference.

2 Preliminaries

2.1 Knowledge Graph Embeddings

A knowledge graph (KG) $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ can be formalized as a set of triplets (e_i, r, e_j) composed of head and tail entities e_i and e_j ($e_i, e_j \in \mathcal{E}$, $\mathcal{E}$ being the set of entities) and a relation $r \in \mathcal{R}$ ($\mathcal{R}$ being the set of relations) – $n_e = |\mathcal{E}|$, $n_r = |\mathcal{R}|$. The goal of learning KG embeddings is to learn vector embeddings $\mathbf{e} \in \mathbb{R}^{d_e}$ for each entity $e \in \mathcal{E}$ (and possibly also relation embeddings $\mathbf{r} \in \mathbb{R}^{d_r}$).

Typical KG embedding approaches are multi-layer neural networks which consist of an embedding component and a scoring component. The embedding component maps each entity to its corresponding embedding. The scoring component learns a scoring function $\mathbf{f} : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightarrow \mathbb{R}$ where $f(e_i, r, e_j)$ defines the score of the triplet (e_i, r, e_j). KG embeddings are learnt by defining a loss function $\mathcal{L}$ and solving the following optimization problem:

$$\min_{\Theta} \sum_{(e_i,r,e_j) \in \mathcal{G}} \mathcal{L}_{\Theta}\left(e_i, r, e_j\right) \tag{1}$$

Here Θ includes all embedding parameters and any other neural network parameters. The loss function typically encourages the score of a positive triplet (e_i, r, e_j) to be higher than that of a (corrupted) negative triplet. In Table 1, we summarize the scoring function for several existing KG embedding approaches as well as their corresponding entity (and relation) representation parameters.

In all the KG embedding models, the number of parameters grow super-linearly with the number of entities and relations in the KG as well as the size of their representations. This number can be very large and learning KG embeddings can be a challenge for large, sparse KGs. In this paper, we present a novel coding scheme that significantly reduces the number of embedding parameters. We do so by leveraging recent advances in discrete representation learning. We summarize them below.

2.2 Discrete Representation Learning

Typical deep learning methods define an embedding function as $F : \mathcal{V} \rightarrow \mathbb{R}^d$, where $\mathcal{V}$ denotes the vocabulary such as words, sub-words, entities, relations, etc. and each symbol in the vocabulary is mapped to a continuous vector in $\mathbb{R}^d$. The embedding function can be trained separate from the task in a completely unsupervised manner or jointly with other neural net parameters to optimize the target loss function. A common specification of the embedding function in NLP is a lookup table $L \in \mathbb{R}^{n \times d}$ with $n = |\mathcal{V}|$. The total number of bits used to represent this table is $\mathcal{O}(nd)$ ($32nd$ if each real number is represented by 32-bit floating point). This is problematic for large n and/or d.

Thus, various approaches have been proposed to compress embedding layers in neural networks. These include weight-tying (Press and Wolf, 2016; Inan et al., 2016; Li et al., 2018), matrix-factorization based approaches (Acharya et al., 2019), and approaches that rely on gumbel softmax (Baevski and Auli, 2018), vector quantization (Chen and Sun, 2019) and codebook learning (Shu and Nakayama, 2017). In this work, we build on discrete representation learning approaches (van den Oord et al., 2017; Chen et al., 2018; Chen and Sun, 2019). Discrete representation learning gives us a way to mitigate this issue by representing each symbol v in the vocabulary as a discrete vector $\mathbf{z}_v = [\mathbf{z}_v^{(1)}, \ldots \mathbf{z}_v^{(D)}]$. Discrete representations have another clear benefit that they are interpretable and are a natural fit for complex reasoning, planning and predictive learning tasks.

Learning discrete representations is challenging due to the inherent non-differentiability in the embedding layer. Thus, a number of solutions such as the gumbel softmax trick (Maddison et al., 2016; Jang et al., 2016) and the straight-through estimator (Bengio et al., 2013) have been proposed to tackle this issue. Making the discrete representation learn-

KG Embedding Model	Scoring function f	# Ent. Rel. Params
SE (Bordes et al., 2011)	$\|\|\mathbf{W}_r^{(L)}\mathbf{e}_i - \mathbf{W}_r^{(R)}\mathbf{e}_j\|\|_p$	$n_e d_e + 2n_r d_e d_r$
NTN (Socher et al., 2013)	$\mathbf{u}_r^T f\left(\mathbf{e}_i\mathbf{W}_r^{[1\ldots k]}\mathbf{e}_j + \mathbf{V}_r\begin{bmatrix}\mathbf{e}_i\\ \mathbf{e}_j\end{bmatrix} + \mathbf{b}_r\right)$	$n_e d_e + n_r(kd_e^2 + 2kd_e + k)$
TransE (Bordes et al., 2013)	$\|\|\mathbf{e}_i + \mathbf{r} - \mathbf{e}_j\|\|_p$	$(n_e + n_r)d$
TransH (Wang et al., 2014)	$\|\|(\mathbf{e}_i - \mathbf{w}_r^T\mathbf{e}_i\mathbf{w}_r) + \mathbf{d}_r - (\mathbf{e}_j - \mathbf{w}_r^T\mathbf{e}_j\mathbf{w}_r)\|\|_2^2$	$n_e d_e + 2n_r d_r$
TransR (Lin et al., 2015)	$\|\|\mathbf{e}_i\mathbf{W}_r + \mathbf{r} - \mathbf{e}_j\mathbf{W}_r\|\|_2^2$	$n_e d_e + n_r(d_r + d_r^2)$
TransD (Ji et al., 2015)	$-\|\|(\mathbf{r}_p\mathbf{e}_{i_p}^T + \mathbf{I})\mathbf{e}_i + \mathbf{r} - (\mathbf{r}_p\mathbf{e}_{j_p}^T + \mathbf{I})\mathbf{e}_j\|\|_2^2$	$2n_e d_e + 2n_r d_r$
DistMult (Yang et al., 2014)	$\langle\mathbf{e}_i, \mathbf{r}, \mathbf{e}_j\rangle$	$(n_e + n_r)d$
ComplEx (Trouillon et al., 2016)	$\text{Re}(\langle\mathbf{e}_i, \mathbf{r}, \bar{\mathbf{e}}_j\rangle)$	$2(n_e + n_r)d$
HolE (Nickel et al., 2016)	$\langle\mathbf{r}, \mathbf{e}_i \otimes \mathbf{e}_j\rangle$	$(n_e + n_r)d$
SimpleE (Kazemi and Poole, 2018)	$\frac{1}{2}(\langle\mathbf{e}_i^{(h)}, \mathbf{r}, \mathbf{e}_j^{(t)}\rangle + \langle\mathbf{e}_j^{(h)}, \mathbf{r}^{(inv)}, \mathbf{e}_i^{(t)}\rangle)$	$2(n_e + n_r)d$
ConvE (Dettmers et al., 2018)	$\langle\sigma(vec(\sigma([\mathbf{e}_i; \mathbf{r}] \circ \omega))\mathbf{W}), \mathbf{e}_j\rangle$	$n_e d_e + n_r d_r$
RotatE (Sun et al., 2019)	$-\|\|\mathbf{e}_i \bullet \mathbf{r} - \mathbf{e}_j\|\|^2$	$(n_e + n_r)d$
HypER (Balažević et al., 2019a)	$\langle\sigma(vec(\sigma(\mathbf{e}_i * vec^{-1}(\mathbf{w}_r\mathbf{H})))\mathbf{W}), \mathbf{e}_j\rangle$	$n_e d_e + n_r d_r$
TuckER (Balažević et al., 2019b)	$\mathcal{W} \times_1 \mathbf{e}_i \times_2 \mathbf{w}_r \times_3 \mathbf{e}_j$	$n_e d_e + n_r d_r$

Table 1: Scoring functions f of some popular knowledge graph embedding approaches in the literature and the number of entity and relation specific parameters. Here, $n_e = |\mathcal{E}|$ and $n_r = |\mathcal{R}|$ respectively denote the number of entities and relation types and d_e and d_r respectively denote the dimension of entity and relation representations. d is defined when (as) $d = d_e = d_r$. $\langle x^1, \ldots, x^k\rangle = \sum_i x_i^1 \ldots x_i^k$ denotes the generalized dot product, $\bar{}$ denotes the conjugate of a complex number and $\otimes$ denotes circular correlation, σ denotes an activation function, $\circ$ denotes the convolution operator, $\bullet$ denotes the hadamard product and $\times_k$ denotes tensor product along the k^{th} mode.

ing process differentiable enables end-to-end learning of discrete representations via optimizing some task-specific objectives from language modelling and machine translation. In this work, we use discrete representation learning to compress KG embeddings. We describe it below.

3 Discrete KG Representation Learning

In order to learn discrete KG representations, we define a quantization function $\mathcal{Q} : \mathbb{R}^d \rightarrow \mathbb{R}^d$, which (during training) takes raw KG embeddings and produces their quantized representations. $\mathcal{Q} = \mathcal{D} \circ \mathcal{R}$ is composed of two functions:

1. **A discretization function** $\mathcal{D} : \mathbb{R}^{d_e} \rightarrow \mathbb{Z}^D$ that maps the continuous KG embedding into a K-way D-dimensional discrete code with cardinality $|\mathbb{Z}| = K$ (we call this KD code)

2. **A reverse-discretization function** $\mathcal{R} : \mathbb{Z}^D \rightarrow \mathbb{R}^{d_e}$ that maps the KD code back to the continuous embedding.

During training, both $\mathcal{D}$ and $\mathcal{R}$ are learned. Then, every entity in the KG is represented by a KD code via applying the discretization function $\mathcal{D}$ to save space (compression). The continuous embeddings and the parameters of the discretization function are then no longer needed. In the test/inference stage, the reverse-discretization function $\mathcal{R}$ is used to decode the KD codes into regular embedding vectors for every entity. We use vector quantization (Chen et al., 2018; Chen and Sun, 2019) and codebook learning (Cai et al., 2010) to define the discretization and reverse-discretization functions $\mathcal{D}$ and $\mathcal{R}$. We describe them below.

3.1 Discretization Function $\mathcal{D}$

The goal of the discretization function is to map continuous KG embedding vectors into KD codes. We model the discretization function using nearest neighbor search (Cayton, 2008). Given continuous KG embeddings $\{\mathbf{e}_i | i = 1 \ldots n_e\}$ as query vectors, we define a set of K key vectors $\{\mathbf{k}_k | k = 1 \ldots K\}$ where $\mathbf{k}_k \in \mathbb{R}^{d_e}$.

In order to learn D-dimensional discrete codes, we partition the query and key vectors into D partitions where each partition corresponds to one of the D discrete codes – $\mathbf{e}_i^{(j)} \in \mathbb{R}^{n \times d_e/D}$ and $\mathbf{k}_k^{(j)} \in \mathbb{R}^{K \times d_e/D}, j = 1 \ldots D$.

Vector Quantization (VQ): Our first alternative for discretization is vector-quantization (Ballard, 1997), a classical quantization technique for data compression. We assume that the j^{th} discrete code of the i^{th} entity $z_i^{(j)}$ can be computed by calculating distances between the corresponding query vector partition $\mathbf{e}_i^{(j)}$ and various corresponding key vector partitions $\{\mathbf{k}_k^{(j)}\}$, and choosing the one with the minimum distance:

$$z_i^{(j)} = \arg\min_k dist\left(\mathbf{e}_i^{(j)}, \mathbf{k}_k^{(j)}\right) \quad (2)$$

We use the Euclidean distance function:

$dist(\mathbf{a}, \mathbf{b}) = ||\mathbf{a} - \mathbf{b}||_2^2$ in our experiments. Note that the argmin operation is inherently non-differentiable. The resulting quantization function $\mathcal{Q}$ has no gradient towards the input query vectors. Thus, we use the straight-through estimator (Bengio et al., 2013) to compute a pseudo gradient. This means that during the forward pass, we compute $\mathcal{Q}$ as defined here, but during the backward pass, we use the gradient of the query vectors.

Tempering Softmax (TS): Vector quantization is a popular method for learning discrete representations. Yet another popular approach is continuous relaxation of (2) via the tempering softmax (Maddison et al., 2016; Jang et al., 2016). We again use dot product and softmax for computing the proximity between query and key vectors:

$$z_i^{(j)} = \arg\max_k \frac{\exp\left(\langle \mathbf{e}_i^{(j)}, \mathbf{k}_k^{(j)} \rangle / \tau\right)}{\sum_{k'} \exp\left(\langle \mathbf{e}_i^{(j)}, \mathbf{k}_{k'}^{(j)} \rangle / \tau\right)}$$

Here, τ is the temperature and $\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^T \mathbf{b}$ denotes the dot product operation. Note that this function still carries an inherent non-differentiability. Hence, we relax the above and compute probability vectors $\bar{z}_i^{(j)}$ which represent the probability distribution of the j^{th} dimension of the discrete code for the i^{th} entity taking a particular value (say k). Given probabilistic vectors $\bar{z}_i^{(j)}$, we can compute the discrete codes $z_i^{(j)}$ simply by taking the argmax. To compute discrete KD codes, we set a small value of τ. As $\tau \to 0$, the softmax becomes spiky concentrated on the true $z_i^{(j)}$-th dimension. We again estimate pseudo gradients by setting a very small τ in the forward pass (i.e. close to the discrete case (eq. 1)) and $\tau = 1$ in the backward pass.

3.2 Reverse-discretization Function $\mathcal{R}$

The goal of the reverse-discretization function is to map discrete KD codes into continuous KG embedding vectors. We model the reverse-discretization process first by a simple linear model which maps the discrete codes to continuous vectors by looking up a learnt codebook. Then, we present an alternative – a non-linear model for reverse-discretization based on recurrent neural networks.

Codebook Lookup (CL): We first define the reverse-discretization function in a simple manner where we substitute every discrete code with a continuous vector from a codebook. Let $\mathcal{C}$ be a set of codebooks. $\mathcal{C}$ consists of a number of codebooks – a separate codebook $\mathcal{C}^{(j)}$ for each position $j = 1 \dots D$ in the KD code. We model each codebook simply as a set of vectors: $\mathcal{C}^{(j)} = \{\mathbf{c}_i^{(j)} | i = 1 \dots K\}$ where $\mathbf{c}_i^{(j)} \in \mathbb{R}^{d_e/D}$. We simply compute the embedding vector for the j^{th} dimension of the i^{th} entity as:

$$\mathbf{e}_i^{(j)} = \mathbf{c}_i^{(j)}$$

The final entity embedding vector $\mathbf{e}_i$ is achieved by the concatenation of the embedding vectors for each dimension: $\mathbf{e}_i = [\mathbf{e}_i^{(1)} \dots \mathbf{e}_i^{(D)}]$.

Non-linear Reconstruction (NL): While the codebook lookup approach is simple and efficient, due to its linear nature, the capacity of the generated KG embedding may be limited. Thus, we also employ neural network based non-linear approaches for embedding reconstruction. We propose a non-linear embedding reconstruction approach based on the Bi-LSTM network.

Given the KD code $\mathbf{z}_i$ as a sequence of codes $z_i^{(1)}, \dots, z_i^{(D)}$, we map the KD code to a continuous embedding vector by feeding the code to a Bi-LSTM followed by mean pooling.

Let $(\mathbf{h}_i^{(1)}, \dots, \mathbf{h}_i^{(D)}) = \textit{Bi-LSTM}\left(z_i^{(1)}, \dots, z_i^{(D)}\right)$ be the hidden state representations for the various Bi-LSTM cells. Finally, we reconstruct the entity embedding $\hat{\mathbf{e}}_i$ by mean-pooling the code embedding vectors followed by a linear transformation: $\mathbf{e}_i = \mathbf{W}_{rev}^T \left(\sum_j \mathbf{h}_i^{(j)}\right)$.

We also tried to map the KD code to a continuous embedding vector by feeding the code to variations of a character level CNN (Kim et al., 2016). However, the Char CNN model always performed worse than the Bi-LSTM model in our experiments. This was because our discretization function which discretizes contiguous partitions of the continuous representation better suits the Bi-LSTM reconstruction model. In the future, we would like to consider more complex discretization functions with other complex non-linear reconstruction models.

Storage Efficiency: A key motivation of learning discrete representations is that we can significantly compress the embedding layer at test time. The size of the embedding layer for typical KG representations is $32 n_e d_e$ (assuming a 32 bit representation) – this can be very large. In contrast, with discrete representation learning, we only need to store code embeddings $\{\mathbf{z}_i\}$ and the parameters used in the reverse-discretization function such as

the codebooks $\mathcal{C}$ or the parameters of the embedding reconstruction Bi-LSTM $\{\mathbf{\Theta_{LSTM}}, \mathbf{W}_{rev}\}$.

The entity codes require $n_e D \log_2 K$ bits.

The codebook lookup approach needs to also maintain codebooks which require $32Kd_e$ parameters and the non-linear reconstruction approach requires $Dd' \times 6$ parameters (two set of parameter matrices each for the input, output and forget gates) for the Bi-LSTM and $d_e d'$ parameters for storing $\mathbf{W}_{rev}$ – a total of $(6D + d_e)d'$ parameters. Here, d' is the size of the code embedding vectors.

In both codebook lookup and non-linear reconstruction formulations, discrete representation learning neatly decouples the KG size (number of entities) and dimensionality of the continuous embeddings. Thus, the discrete embedding layer can be compactly stored as typically D and $\log_2 K$ are smaller than $32d_e$ (considering only the dominating term n_e).

Test Time Inference of Embeddings: At test time, we retrieve continuous embeddings for an entity by looking up the codebook or running inference on the reconstruction model using its discrete representation. For codebook lookup, the steps involved are (a) looking up a simple index for each code, and (b) concatenation. Since only index lookups and concatenation are needed, the extra computation complexity and memory footprint are very small - $\mathcal{O}(D)$ time and memory. In the non-linear reconstruction setting, we need to run inference on the Bi-LSTM model. This requires $\mathcal{O}(D)$ matrix vector multiplications (to compute various LSTM gates) which takes $\mathcal{O}(Dd_e d')$ time. Finally, we have another linear transformation $\mathbf{W}_{rev}$ – this takes $\mathcal{O}(d_e d')$ time.

We can further cache the embedding lookups and various intermediate results such as matrix vector products to improve performance. We show in our results that the test time inference overhead is typically very small.

Learning: Similar to previous continuous KG representation learning methods, we learn discrete entity representations by minimizing the triplet loss function. We extend equation 1 as:

$$\min_{\{\mathbf{z}_e\},\theta,\Theta} \sum_{(e_i,r,e_j)\in\mathcal{G}} \mathcal{L}_{\{\mathbf{z}_e\},\theta,\Theta}\left(e_i, r, e_j|\theta, \Theta\right) \quad (3)$$

Here, $\mathbf{z}_e$ are code embeddings, θ are the parameters of the reverse-discretization function ($\mathcal{C}$ or $\{\theta_{LSTM}, \mathbf{W}_{rev}\}$) and Θ denotes parameters of the KG embedding approaches (listed in Table 1). The aforementioned loss function (eq 3) is differentiable w.r.t. the embedding parameters and parameters of entity representation learning methods. However, the discrete codes introduce a non-differentiability. Thus, we use straight-through (Bengio et al., 2013) or the tempering softmax (Maddison et al., 2016; Jang et al., 2016) to estimate pseudo-gradients as described before (section 3.1).

Guidance from KG embeddings: We find that even with sophisticated discrete representation learning methods, solving the above optimization problem can be challenging in practice. Due to discreteness of the problem, this can lead to a sub-optimal solution where discrete codes are not as good. Therefore, we also use guidance from continuous KG embeddings to solve (3) when provided[2]. The key idea is that in addition to optimizing (3), we can encourage the reconstructed embeddings from the learnt discrete codes to mimic continuous embeddings.

In order to provide this guidance from continuous embeddings, during the training, instead of using the reconstructed embedding vector generated from the discrete code, we use a weighted average of the reconstructed embeddings and continuous embeddings obtained using methods described in Table 1: $(1-\lambda)\mathcal{D} \circ \mathcal{R}(\mathbf{e}) + \lambda\mathbf{e}$. Here $\lambda \in (0, 1)$ is a linear interpolant for selecting between reconstructed embeddings and pre-learnt continuous embeddings. We initialize λ to 1 and gradually decrease λ as training proceeds. This enables the method to gradually rely more and more on reconstruction from discrete embeddings. We also add a regularization term $||\mathcal{D} \circ \mathcal{R}(\mathbf{e}) - \mathbf{e}||_2^2$ during the training to encourage the reconstructed embeddings to match the pre-learnt continuous embeddings. This procedure is similar to knowledge-distillation guidance (Hinton et al., 2015) in previous discrete representation learning works (Chen et al., 2018).

Here $\lambda \in (0, 1)$ is a linear interpolant for selecting between reconstructed embeddings and pre-learnt continuous embeddings. We initialize λ to 1 and gradually decrease λ as training proceeds. This enables the method to gradually rely more and more on reconstruction from discrete embeddings. We also add a regularization term $||\mathcal{D} \circ \mathcal{R}(\mathbf{e}) - \mathbf{e}||_2^2$ during the training to encourage the reconstructed embeddings to match the pre-learnt continuous em-

[2]We show in our experiments that this guidance, while helpful, is not always needed.

Dataset	Entities	Relations	Train	Valid	Test
FB15k	14,951	1,345	483,142	50,000	59,071
FB15k-237	14,541	237	272,115	17,535	20,466
WN18	40,943	18	141,442	5,000	5,000
WN18RR	40,943	11	86,835	3,034	3,134

Table 2: A summary of dataset statistics

beddings.

4 Experiments

We compare the baseline continuous representations described earlier in Table 1 with four discrete representation learning techniques described in this paper:

- **VQ-CL:** $\mathcal{D}$ = VQ and $\mathcal{R}$ = CL
- **VQ-NL:** $\mathcal{D}$ = VQ and $\mathcal{R}$ = NL
- **TS-CL:** $\mathcal{D}$ = TS and $\mathcal{R}$ = CL
- **TS-NL:** $\mathcal{D}$ = TS and $\mathcal{R}$ = NL

4.1 Datasets

We evaluate our approach on four standard link prediction datasets:

- **FB15k** (Bordes et al., 2013) is a subset of Freebase.
- **FB15k-237** (Toutanova et al., 2015) is a subset of the FB15k dataset created by removing inverse relations that cause test leakage.
- **WN18** (Bordes et al., 2013) is a subset of WordNet.
- **WN18RR** (Dettmers et al., 2018) is a subset of the WN18 dataset created by removing inverse relations.

We summarize all the data statistics in Table 2. We also use the **Countries** dataset (Bouchard et al., 2015) for some in-depth analysis of inference abilities of discrete representations.

4.2 Implementation Details

We implement discrete KG representation learning by extending *OpenKE* (Han et al., 2018), an open-source framework for learning KG embeddings implemented on PyTorch [3]. We train and test all our models on a single 2080Ti system. We set $K = 32$ and $D = 10$ in our experiments unless stated otherwise. For the linear embedding transformation function in the non-linear reconstruction approach, we use a hidden layer of 100 hidden units. We set λ as $\lambda = \frac{1}{\sqrt{t}}$ at the t^{th} epoch. We tune the regularization coefficient using grid search on the validation set.

[3] https://github.com/thunlp/OpenKE

4.3 Results

Link Prediction: We learn discrete representations corresponding to various continuous KG representations (described in Table 1) and compare the obtained discrete representations with their continuous counterparts. We use the same hyper-parameter settings as in the original KG embedding papers. We generate n_e candidate triples for each test triple by combining the test entity-relation pair with all possible entities $\mathcal{E}$. We use the filtered setting (Bordes et al., 2013), i.e. all known true triples are removed from the candidate set except for the current test triple. We use standard evaluation metrics previously used in the literature: mean reciprocal rank (MRR) and hits@10 (H@10). Mean reciprocal rank is the average of the inverse of the mean rank assigned to the true triple over all candidate triples. Hits@10 measures the percentage of times a true triple is ranked within the top 10 candidate triples. In addition, in order to report the compression efficiency of the discrete representations, we also report the compression ratio which is computed as follows:

$$CR = \frac{\textit{Storage(continuous)}}{\textit{Storage(discrete)}}$$

Here, *Storage(continuous)* is the storage used to store full continuous KG representations. *Storage(discrete)* is the storage used in the discrete representation learning method (during the testing stage). This includes discrete KG representations as well as parameters of the reverse-discretization function (i.e. codebook or Bi-LSTM parameters).

Tables 3, 4, 5 and 6 show our results on the link prediction task on the four datasets respectively. In Table 3, we compare various continuous representations with the four discrete representation learning techniques described in this paper. We find that the discrete representations sustain only minor losses in performance (and are sometimes actually better than their continuous counterparts) in terms of both evaluation metrics: MRR and H@10, while being able to obtain significant embedding compression (42x-585x). Table 3 also compares the different discrete representation learning approaches. We observe that TS-NL which uses tempering softmax and non-linear reconstruction performs the best in most of the settings. This observation was also

	Continuous		CR	VQ-CL		TS-CL		CR	VQ-NL		TS-NL	
	MRR	H@10	(CL)	MRR	H@10	MRR	H@10	(NL)	MRR	H@10	MRR	H@10
TransE	0.463	0.749	46.3	0.462	0.748	**0.467**	**0.749**	42.6	**0.463**	0.746	**0.477**	**0.755**
DistMult	0.798	0.893	77.6	0.750	0.859	0.775	0.864	71.4	0.756	0.868	0.790	0.882
HolE	0.524	0.739	112.6	0.515	0.708	0.517	0.711	103.8	0.517	0.717	**0.525**	0.726
ComplEx	0.692	0.840	262.3	0.651	0.802	0.653	0.814	228.4	0.670	0.818	0.678	0.833
ConvE	0.657	0.831	77.6	0.618	0.774	0.620	0.798	71.4	0.626	0.793	0.644	0.820
RotatE	0.797	0.884	585.3	0.765	0.840	0.782	0.876	495.2	0.789	0.878	**0.798**	0.881
HypER	0.790	0.734	177.5	0.743	0.706	0.754	0.715	161.1	0.758	0.718	0.763	0.726
TuckER	0.795	0.741	177.5	0.773	0.714	0.782	0.729	161.1	0.787	0.723	0.783	0.726

Table 3: Results of several models and our proposed discrete counterparts evaluated on the FB15K dataset

	Continuous		Discrete (TS-NL)		
	MRR	H@10	CR	MRR	H@10
TransE	0.495	0.943	103.3	**0.499**	0.940
DistMult	0.797	0.946	143.2	0.774	0.921
HolE	0.938	0.949	228.6	**0.938**	0.929
ComplEx	0.941	0.947	437.1	0.934	0.936
ConvE	0.943	0.956	143.2	0.933	0.936
RotatE	0.949	0.959	952.6	0.946	0.952
HypER	0.951	0.947	327.9	0.946	0.942
TuckER	0.953	0.949	327.9	0.924	0.920

Table 4: Results of several models and our proposed discrete counterpart (TS-NL) evaluated on the WN18 dataset

	Continuous		Discrete (TS-NL)		
	MRR	H@10	CR	MRR	H@10
TransE	0.294	0.465	43.1	**0.298**	0.463
DistMult	0.241	0.419	71.8	**0.241**	**0.422**
HolE	0.318	0.430	104.0	0.316	0.428
ComplEx	0.247	0.428	228.5	0.238	0.411
ConvE	0.325	0.501	71.8	0.321	0.488
RotatE	0.338	0.533	495.2	0.336	0.528
HypER	0.341	0.252	161.3	0.332	**0.286**
TuckER	0.358	0.266	161.3	0.331	**0.279**

Table 5: Results of several models and our proposed discrete counterpart (TS-NL) evaluated on the FB15K-237 dataset.

made on the other three datasets. Hence, in Tables 4, 5 and 6, we only compare TS-NL with the continuous representations. We again observe that TS-NL compresses the KG embeddings (71x-952x) while suffering only a minor loss in performance.

Logical Inference with Discrete representations: KG embeddings give us a way to perform logical inference and reason about knowledge. In this experiment, we explore if discrete representations retain the ability to perform inference and reasoning in KGs. We evaluate our models on the countries dataset (Bouchard et al., 2015) which was designed to test the logical inference capabilities of KG embedding models. We use the same evaluation protocol as in (Nickel et al., 2016) for our

	Continuous		Discrete (TS-NL)		
	MRR	H@10	CR	MRR	H@10
TransE	0.226	0.501	105.2	**0.230**	0.498
DistMult	0.430	0.490	143.8	0.423	0.476
HolE	0.338	0.438	228.8	**0.346**	0.435
ComplEx	0.440	0.510	437.2	0.433	0.494
ConvE	0.430	0.520	143.8	**0.431**	0.500
RotatE	0.476	0.571	952.6	0.452	0.546
HypER	0.465	0.436	328.0	0.460	**0.437**
TuckER	0.470	0.443	328.0	0.452	0.442

Table 6: Results of several models and our proposed discrete counterpart (TS-NL) evaluated on the WN18RR dataset

experiments. The countries dataset contains 2 relations and 272 entities (244 countries, 5 regions and 23 subregions) and 3 tasks are posed, requiring subsequently longer and harder inference than the previous one:

1. Task S1 poses queries of the form locatedIn(c; ?), and the answer is one of the five regions.

2. Task S2 poses queries of the form neighborOf(c1; c2) $\wedge$ locatedIn(c2; r) $\implies$ locatedIn(c1; r)

3. Task S3 poses queries of the form neighborOf(c1; c2) $\wedge$ locatedIn(c2; s) $\wedge$ locatedIn(s; r) $\implies$ locatedIn(c1; r):

We use the AUC-PR metric, which was also used in previous works (Bouchard et al., 2015; Nickel et al., 2016). Table 7 shows our results. We find that TS-NL is a very good KG representation for KG inference. Infact, we find that TS-NL outperforms many of their continuous counterparts.

Additional Inference Cost: A tradeoff in learning discrete KG representations is that the inference time increases as we need to decompress discrete representations into continuous embeddings for every entity before using them by looking up the

	S1		S2		S3	
	Ct.	Dis.	Ct.	Dis.	Ct.	Dis.
TransE	0.93	**0.95**	0.56	**0.59**	0.34	**0.41**
DistMult	1.00	0.97	0.72	0.71	0.52	**0.55**
HolE	1.00	0.97	0.77	**0.80**	0.70	**0.74**
ComplEx	0.97	0.97	0.57	0.56	0.43	**0.45**
ConvE	1.00	1.00	0.99	0.96	0.86	**0.87**
RotatE	1.00	1.00	1.00	0.98	0.95	0.91
HypER	1.00	0.97	0.76	**0.80**	0.68	**0.75**
TuckER	1.00	1.00	0.85	**0.88**	0.75	**0.79**

Table 7: Results of several continuous representations (ct.) and discrete TS-NL (dis.) evaluated on the three tasks (S1, S2 and S3) of logical inference on countries dataset.

codebook or running inference on the LSTM reconstruction model. In practice, we found that this additional inference cost was very small. For example, the additional inference cost of running TransE on the entire FB15K test set was $\approx$ 1 minute for codebook lookup and $\approx$ 2.5 minutes for non-linear reconstruction approach on our single 2080Ti system. The additional inference cost for the other continuous KG representations were similarly low.

Varying K and D: There is an evident tradeoff between the extent of compression (which is dictated by the choice of K and D) and model performance. In order to explore this tradeoff, we plot heatmaps of performance (MRR) and compression ratio (CR) on the FB15K test set as we vary K and D for TransE in Figure 1. Not surprisingly, the performance drops as the compression increases. Plotting these heat maps would allow the end user to pick K and D depending on their tolerance to loss in performance.

Dependence on guidance from continuous embeddings: We evaluate the contribution of the guidance from continuous embeddings in learning discrete KG representations. Figure 2 compares the test MRR for TS-NL as training proceeds on the FB-15K dataset when we do or do not have guidance from the continuous representation (TransE). We observe that learning in the unguided model is much slower than the guided model. However, the guided model achieves almost similar performance in the end. Thus, we conclude that while guidance helps us achieve faster and more stable convergence, it is not necessary to learn discrete representations.

Quality of the Discrete representations: We also assess the quality of the learnt discrete entity representations directly as features for the link predic-

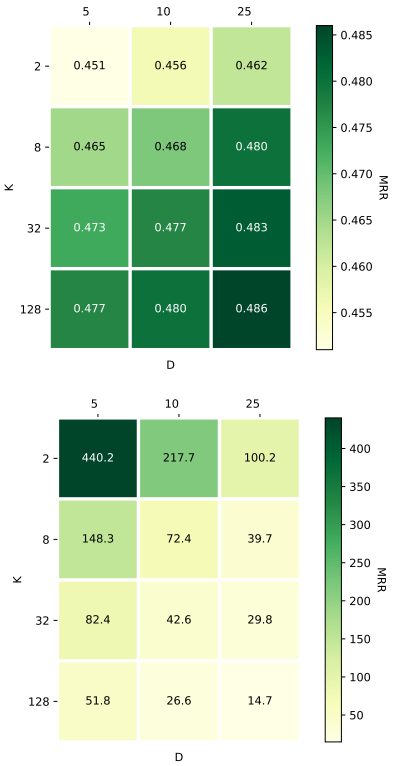

Figure 1: Heatmaps of performance (MRR) and CR for TS-NL on FB15K dataset as we vary K and D – darker is better.

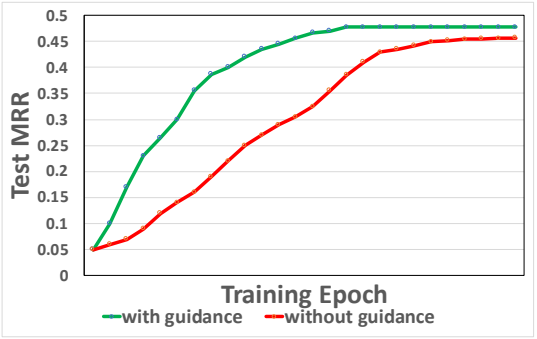

Figure 2: Test MRR for TS-NL as training proceeds on FB-15K dataset with and without guidance from continuous embeddings.

	MRR	H@10
TransE	0.472	0.740
DistMult	0.778	0.870
HolE	0.497	0.706
ComplEx	0.654	0.816
ConvE	0.596	0.797
RotatE	0.743	0.828
HypER	0.722	0.694
TuckER	0.742	0.705

Table 8: Transfer results on FB15K dataset.

	Code	Entities
WN	3-7-0-6-X	animalize, work_animal, farm_animal, animal_husbandry, offspring, animal, invertebrate, marine_animal, animal_kingdom, predator
	5-3-0-X-1	jabalpur, calcutta, bombay, hyderabad, chennai, lucknow, mysore
FB	2-X-7-4-1	novelist, dramatist, actor, writer, cartoonist, poet, songwriter, musician
	2-5-X-4-1	albert_einstein, voltaire, isaac_newton, nikola_tesla

Table 9: Example learned codes (K=8, D=5, $X \in \{0, 7\}$) for Freebase (FB) and Wordnet (WN). Similar entities are assigned to close-by codes.

tion task. In this case, we only retain the discrete entity representations learnt by TS-NL and learn a new LSTM based non-linear reverse-discretization on the validation set. Then, we obtain the link-prediction performance on the test set as before (see Table 8 for transfer results on the FB15K dataset). We observe that the performance of this "transfer" model is close to that of the original model which used a pre-trained reverse-discretization model (compare Table 8 with the shaded part of Table 3). Note that, in the "transfer" setting, we can achieve much higher compression as we do not even need to store the reverse-discretization model.

Interpretability of discrete representations: The discrete codes provide us with additional interpretability which continuous representations can lack. In Table 9, we show a sample of learned codes for the two datasets. We observe that semantically similar entities are assigned to close-by codes.

5 Related Work

Deep learning model compression has attracted many research efforts in the last few years (Han et al., 2015). These efforts include network pruning (Reed, 1993; Castellano et al., 1997), weight sharing (Ullrich et al., 2017), quantization (Lin et al., 2016), low-precision computation (Hwang and Sung, 2014; Courbariaux et al., 2015) and knowledge distillation (Hinton et al., 2015) These techniques can also be used for embedding compression. Press and Wolf (2016) and Inan et al. (2016) propose weight-tying approaches that learn input and output representations jointly. Matrix factorization-based methods (Acharya et al., 2019; Shu and Nakayama, 2017; Li et al., 2018) have also been proposed which approximate an embedding matrix with smaller matrices or clusters. Closest to our work are (Shu and Nakayama, 2017; Chen et al., 2018; Chen and Sun, 2019) who present similar approaches to learn discrete codings for word embeddings using multiple codebooks, i.e. product quantization (Jegou et al., 2010). Similar techniques have used been used by van den Oord et al. (2017) who extend VAEs to learn discrete representations using vector quantization in the image domain. This allows the VAE model to circumvent its well known issues of "posterior collapse". All these previous works have been applied to the image domain, and sometimes in language to learn discrete word embeddings. In this work, we present the first results on compressing KG embeddings and also show how the compressed embeddings can be used to support various knowledge based applications such as KG inference.

6 Conclusion

The embedding layer contains majority of the parameters in any representation learning approach on knowledge graphs. This is a barrier in successful deployment of models using knowledge graphs at scale on user-facing computing devices. In this work, we proposed novel and general approaches for KG embedding compression. Our approaches learn to represent entities in a KG as a vector of discrete codes in an end-to-end fashion. At test time, the discrete KG representation can be cheaply and efficiently converted to a dense embedding and then used in any downstream application requiring the use of a knowledge graph. We evaluated our proposed methods on different link prediction and KG inference tasks and show that the proposed methods for KG embedding compression can effectively compress the KG embedding table without suffering any significant loss in performance. In this work, we only considered the problem of learning discrete entity representations. In the future, we would like to jointly learn discrete representations of entities as well as relations.

Acknowledgments

MS would like to thank the anonymous reviewers, along with Karen Livescu, Kevin Gimpel and Shuning Jin for their valuable comments and suggestions on this work.

References

Anish Acharya, Rahul Goel, Angeliki Metallinou, and Inderjit Dhillon. 2019. Online embedding compression for text classification using low rank matrix factorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6196–6203.

Alexei Baevski and Michael Auli. 2018. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853*.

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

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

Dana H Ballard. 1997. *An introduction to natural computation*. MIT Press.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

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*, pages 2787–2795.

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. 2011. Learning structured embeddings of knowledge bases. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Guillaume Bouchard, Sameer Singh, and Theo Trouillon. 2015. On approximate reasoning capabilities of low-rank vector spaces. In *2015 AAAI Spring Symposium Series*.

Hongping Cai, Fei Yan, and Krystian Mikolajczyk. 2010. Learning weights for codebook in image classification and retrieval. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2320–2327. IEEE.

Giovanna Castellano, Anna Maria Fanelli, and Marcello Pelillo. 1997. An iterative pruning algorithm for feedforward neural networks. *IEEE transactions on Neural networks*, 8(3):519–531.

Lawrence Cayton. 2008. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the 25th international conference on Machine learning*, pages 112–119. ACM.

Ting Chen, Martin Renqiang Min, and Yizhou Sun. 2018. Learning k-way d-dimensional discrete codes for compact embedding representations. *arXiv preprint arXiv:1806.09464*.

Ting Chen and Yizhou Sun. 2019. Differentiable product quantization for end-to-end embedding compression.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Low precision arithmetic for deep learning. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*.

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

Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Xu Han, Shulin Cao, Lv Xin, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. 2018. Openke: An open toolkit for knowledge embedding. In *Proceedings of EMNLP*.

He He, Anusha Balakrishnan, Mihail Eric, and Percy Liang. 2017. Learning symmetric collaborative dialogue agents with dynamic knowledge graph embeddings. *arXiv preprint arXiv:1704.07130*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Kyuyeon Hwang and Wonyong Sung. 2014. Fixed-point feedforward deep neural network design using weights +1, 0, and -1. In *SiPS*, pages 174–179. IEEE.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128.

Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the*

53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pages 687–696.

Seyed Mehran Kazemi and David Poole. 2018. Simple embedding for link prediction in knowledge graphs. In *Advances in Neural Information Processing Systems*, pages 4284–4295.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Zhongliang Li, Raymond Kulhanek, Shaojun Wang, Yunxin Zhao, and Shuang Wu. 2018. Slim embedding layers for recurrent neural language models. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. 2016. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-ninth AAAI conference on artificial intelligence*.

Vanessa Lopez, Christina Unger, Philipp Cimiano, and Enrico Motta. 2013. Evaluating question answering over linked data. *Web Semantics Science Services And Agents On The World Wide Web*, 21:3–13.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. 2016. Holographic embeddings of knowledge graphs. In *Thirtieth Aaai conference on artificial intelligence*.

Aaron van den Oord, Oriol Vinyals, et al. 2017. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6306–6315.

Ofir Press and Lior Wolf. 2016. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*.

Russell Reed. 1993. Pruning algorithms-a survey. *IEEE transactions on Neural Networks*, 4(5):740–747.

Raphael Shu and Hideki Nakayama. 2017. Compressing word embeddings via deep compositional code learning. *arXiv preprint arXiv:1711.01068*.

Amit Singhal. 2012. Introducing the knowledge graph: things, not strings. 2012 (accessed: 16 May 2012).

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, pages 926–934.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509.

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

Karen Ullrich, Edward Meeds, and Max Welling. 2017. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *Twenty-Eighth AAAI conference on artificial intelligence*.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*.

Tom Young, Erik Cambria, Iti Chaturvedi, Hao Zhou, Subham Biswas, and Minlie Huang. 2018. Augmenting end-to-end dialogue systems with commonsense knowledge. In *Thirty-Second AAAI Conference on Artificial Intelligence*.