Analysis and Methods for Knowledge Graph Embeddings

A THESIS

SUBMITTED FOR THE DEGREE OF

Doctor of Philosophy

IN THE

Faculty of Engineering

BY

Chandrahas

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

December, 2021

Declaration of Originality

I, **Chandrahas**, with SR No. **04-04-00-10-12-15-1-12859** hereby declare that the material presented in the thesis titled

Analysis and Methods for Knowledge Graph Embeddings

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2015-2021**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Saturday 25th December, 2021 Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Partha Pratim Talukdar Advisor Signature

DEDICATED TO

My Family

for their unconditional love and support

Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Partha Pratim Talukdar for their invaluable advice, continuous support, and patience during my PhD study. I am thankful to them for sharing their deep knowledge and extensive experience that will help me shape my research career. I am deeply grateful to my teachers at Indian Institute of Science for imparting a strong foundation of basic concepts during my coursework. It played a crucial role in the completion of this thesis.

Collaboration is a great way of learning and I was fortunate enough to work with multiple collaborators. I would like to thank my collaborators Srinivas Ravishankar, Aditya Sharma, Tathagata Sengupta, Cibi Pragadeesh, and Nilesh Agrawal. I would also like to thank the members of MALL Lab for the insightful discussions, comments, suggestions, and help. I am also thankful to my friends for making this journey more cheerful. Finally, I am deeply thankful to my parents for providing me the freedom and support for pursuing PhD.

Abstract

Knowledge Graphs (KGs) are multi-relational graphs where nodes represent entities, and typed edges represent relationships among entities. These graphs store real-world facts such as (*Lionel Messi, plays-for-team, Barcelona*) as edges, called triples. KGs such as NELL, YAGO, Freebase, and WikiData have been very popular and support many applications such as Web Search, Query Recommendation, and Question Answering. Although popular, these KGs suffer from incompleteness. Learning Knowledge Graph Embeddings (KGE) is a common approach for predicting missing edges (i.e., link prediction) and representing entities and relations in downstream tasks. While numerous KGE methods have been proposed in the past decade, our understanding and analysis of such embeddings have been limited. Further, such methods only work well with ontological KGs. In this thesis, we address these gaps.

Firstly, we study various KGE methods and present an extensive analysis of these methods, resulting in many insights. Next, we address an under-explored problem of link prediction in Open Knowledge Graphs (OpenKGs) and present a novel approach that improves the type compatibility of predicted edges. Lastly, we present an adaptive interaction framework for learning KG embeddings that generalizes many existing methods.

In the first part, we present a macro and a micro analysis of embeddings learned by various KGE methods. Despite the popularity and effectiveness of KG embeddings, their geometric understanding (i.e., arrangement of entity and relation vectors in vector space) is unexplored. We initiate a study to analyze the geometry of KG embeddings and correlate it with task performance and other hyper-parameters. Firstly, we present a set of metrics (e.g., Conicity, ATM) to analyze the geometry of a group of vectors. Using these metrics, we find sharp differences between the geometry of embeddings learned by different classes of KGE methods. The vectors learned by a multiplicative model lie in a narrow cone, unlike additive models where the vectors are spread out in the space. This behavior of multiplicative models is amplified by increasing the number of negative samples used for training. Further, a very high Conicity value is negatively correlated with the performance on the link prediction task. We also study the problem of understanding KG embeddings' semantics and propose an approach to learn

more coherent dimensions. A dimension is coherent if the top entities have similar types (e.g., person). In this work, we formalize the notion of coherence using entity co-occurrence statistics and propose a regularizer term that maximizes coherence while learning KG embeddings. The proposed approach significantly improves coherence while having a comparable performance with baseline in the link prediction and triple classification tasks. Further, based on the human evaluation, we demonstrate that the proposed approach learns more coherent dimensions than the baseline.

In the second part, we address the problem of learning KG embeddings for Open Knowledge Graphs (OpenKGs), focusing on improving link prediction. An OpenKG refers to a set of (head noun phrase, relation phrase, tail noun phrase) triples such as (tesla, return to, new york) extracted from a text corpus using OpenIE tools. While OpenKGs are easy to bootstrap for a domain, they are very sparse. Therefore, link prediction becomes an important step while using these graphs in downstream tasks. Learning OpenKG embeddings is one approach for link prediction that has received some attention lately. However, on careful examination, we find that current algorithms often predict noun phrases (NPs) with incompatible types for given noun and relation phrases. We address this problem and propose OKGIT that improves OpenKG link prediction using novel type compatibility score and type regularization. With extensive experiments on multiple datasets, we show that the proposed method achieves state-of-the-art performance while producing type compatible NPs in the link prediction task.

In the third part, we address the problem of improving the KGE models. Firstly, we show that the performance of existing approaches vary across different datasets, and a simple neural network-based method can consistently achieve better performance on these datasets. Upon analysis, we find that KGE models depend on fixed sets of interactions among the dimensions of entity and relation vectors. Therefore, we investigate ways to learn such interactions automatically during training. We propose an adaptive interaction framework for learning KG embeddings, which can learn appropriate interactions while training. We show that some of the existing models could be seen as special cases of the proposed framework. Based on this framework, we also present two new models, which outperform the baseline models on the link prediction task. Further analysis demonstrates that the proposed approach can adapt to different datasets by learning appropriate interactions.

Publications based on this Thesis

1. **OKGIT: Open Knowledge Graph Link Prediction with Implicit Types**.
   Chandrahas, Partha Pratim Talukdar.
   In Findings of the Association for Computational Linguistics: ACL 2021.

2. **Learning to Interact: An Adaptive Interaction Framework for Knowledge Graph Embeddings**.
   Chandrahas, Nilesh Agrawal, Partha Pratim Talukdar.
   In Proceedings of the 17th International Conference on Natural Language Processing (ICON) 2020.

3. **Inducing Interpretability in Knowledge Graph Embeddings**.
   Chandrahas, Tathagata Sengupta, Cibi Pragadeesh, Partha Pratim Talukdar.
   In Proceedings of the 17th International Conference on Natural Language Processing (ICON) 2020.

4. **Towards Understanding the Geometry of Knowledge Graph Embeddings**.
   Chandrahas, Aditya Sharma, Partha Pratim Talukdar.
   In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL) 2018.

5. **Revisiting Simple Neural Networks for Learning Representations of Knowledge Graphs**.
   Srinivas Ravishankar, Chandrahas, Partha Pratim Talukdar.
   In Proceedings of the 6th Workshop on Automated Knowledge Base Construction (AKBC) 2017.

Softwares released based on this Thesis

1. **Open Knowledge Graph Link Prediction**.
   Paper - OKGIT: Open Knowledge Graph Link Prediction with Implicit Types
   In Findings of the Association for Computational Linguistics: ACL 2021.
   Source - https://github.com/Chandrahasd/OKGIT

2. **A tool for Geometrical Analysis of Knowledge Graph Embeddings**.
   Paper - Towards Understanding the Geometry of Knowledge Graph Embeddings
   In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL) 2018.
   Source - https://github.com/malllabiisc/kg-geometry

3. **Knowledge Graph Embeddings**
   Paper - Learning to Interact: An Adaptive Interaction Framework for Knowledge Graph Embeddings.
   In Proceedings of the 17th International Conference on Natural Language Processing (ICON) 2020.
   Source - https://github.com/Chandrahasd/AdaKGE

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Knowledge Graphs (KGs) are multi-relational graphs where nodes represents entities and typed edges represent the relationships among entities. These graphs store facts such as *(Lionel Messi, plays-for-team, Argentina National Football Team)* as edges, also called triples. An example snippet of a KG is shown in Figure 1.1. Here, *Lionel Messi, Argentina National Football Team, FC Barcelona,* and *Argentina* are the entities and *plays-for-team, born-in,* and *has-football-team* are relations among these entities. Construction of KGs have been an active area of research in the past decades and it has led to the development of many KGs such as DBPedia [2], YAGO [61], Freebase [6], and NELL [46]. These KGs have been very popular in supporting many applications like Web Search Query Recommendation [30], Question Answering [77], Visual Question Answering [57] etc. Despite their popularity, they suffer from incompleteness [21], and it becomes important to predict the missing edges in the graph. The task of predicting missing edges in a KG is called link prediction.

Knowledge Graph Embedding (KGE) methods have been a popular approach for representing KG entities and relations. KG Embeddings are useful for KG tasks such as link prediction, triple classification, and node classification, as well as extrinsic tasks such as question answering. Most of these methods represent entities and relations in the KG as vectors in a vector space. A score function uses these vectors to assign a real-valued score to a triple. This score is then used to distinguish correct triples (high scores) from the incorrect ones (low scores). Learning KG Embeddings have received much attention in the recent years and many different methods have been proposed. These methods vary based on the modeling of entities and relations, the score function, and the loss function used for training. Most of these methods can be broadly categorized into three categories. The first category of methods use additive operations as score functions. Some examples of additive methods are TransE [8], TransH [70], TransR [39], STransE [48], and ITransF [73]. The second category of methods use multiplicative operations (e.g., dot

Figure 1.1: An example snippet of a Knowledge Graph (left) and its embeddings (right).

product, Hadamard product, circular correlation, etc.) in their score functions. Examples of multiplicative models include RESCAL [49], DistMult [75], HolE [50], ComplEx [65], and TuckER [3]. Neural methods form the third category that use various neural network architectures. Some examples of these category are ER-MLP [21], ConvE [19], InteractE [67], KG-BERT [76], and CoKE [69]. There are also some methods which fit in more than one category. For example, RotatE [62] score function is a mix of additive and multiplicative operations.

While numerous KG Embedding methods have been proposed in the literature, there are several unaddressed shortcomings. Firstly, our understanding and analysis of such embeddings have been limited. It is not clear how the learned vectors are arranged in the vector space, how it varies with method and hyper-parameters, and how these arrangements are correlated with the effectiveness of the method. Secondly, most of these methods work well with KGs with underlying ontology (Ontological KGs) but not for Open Knowledge Graphs (OpenKGs). An OpenKG is a set of triples extracted from a text corpus using Open Information Extraction (OpenIE) tools such as ReVerb [24]. While structurally similar to Ontological KGs, OpenKGs do not have an underlying ontology and the same entities and relations can be mentioned with multiple surface forms. It makes the link prediction in OpenKGs difficult. Lastly, many existing methods rely on a fixed model-specific set of interactions among entity and relation vectors. Having fixed set of vector interactions not only limits the capability of the method to adapt to different datasets, but it also leaves the choice of appropriate model to the user. We address these shortcomings in this thesis.

In the first part of the thesis, we study various KGE methods and present an extensive analysis of embeddings learned using these methods. We also address the problem of inducing interpretability in KG Embeddings. Specifically, we address the following problems.

1. **Geometry of KG Embeddings:** We study the geometry, i.e., arrangement of vectors learned by various KGE methods. Understanding of the geometry can lead to various insights about the behavior of KGE methods, and it can be helpful in model selection. Further, studying the correlation of the geometry with hyper-parameters and performance of KGE methods can help in selecting hyper-parameters suitable for a task. Therefore, we initiate the study of KG Embeddings' geometry with this work and develop tools that enable such an analysis. Using these tools, we present various insights that improve our understanding of the differences among classes of KGE methods.

2. **Interpretability of KG Embeddings:** Understanding the semantics of the KG embeddings' dimensions has been an under-explored problem. While KG Embeddings have been effective in various tasks, little is known about the interpretability or semantics of those learned embeddings. Interpretable representations enable a better understanding of a model and its predictions. Therefore, we address the problem of inducing interpretability in KG Embeddings and present a method for learning such representations.

In the second part of the thesis, we focus on learning embeddings for Open Knowledge Graphs (OpenKGs). OpenKGs are created from text corpora using OpenIE tools such as ReVerb [24]. It makes OpenKGs suitable for newer domains as they can be easily bootstrapped from domain-specific corpora. However, they come with another set of challenges. OpenKGs are extremely sparse and can have multiple surface forms for the same underlying entity or relation. Therefore, they may not be directly usable for an end-task. Further, it makes the link prediction task even more difficult. OpenKG link prediction has not received much attention in the past. While there are a few methods (e.g., CaRE [27]), they struggle to maintain type compatibility in the link prediction. In this part, we address the problem of link prediction in OpenKGs, focusing on improving type compatibility of the predictions.

Lastly, we address the problem of improving KGE methods in the third part of the thesis. The score function in most of the KGE methods depend on the interactions among the dimensions of the entity and relation vectors. For example, TransE [8] uses additive interactions while DistMult [75] uses multiplicative interactions. These interactions are fixed and manually specified by a method. Having a fixed set of interactions restricts a model's capability to adapt to different dataset. Thus, the choice of appropriate model for a task is left to the user. In this part, we address this problem and present a framework that enables learning relevant interactions adaptively during training. Many of the existing methods can be seen as special cases of this framework.

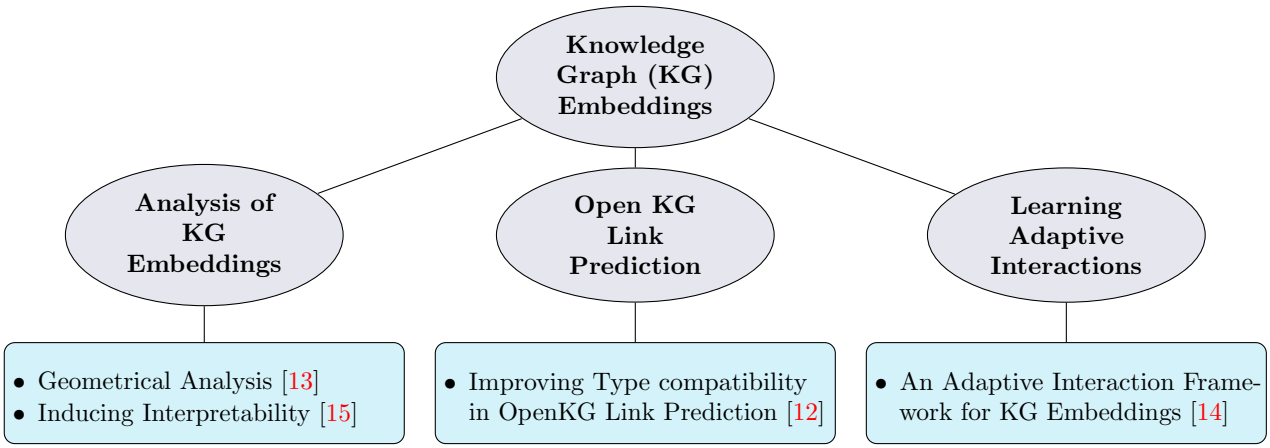


Figure 1.2: Contributions from the thesis.

1.1 Summary of Contributions

The contributions from this thesis can be summarized as follows:

Analysis of Knowledge Graph Embeddings: Despite the popularity and effectiveness of KG embeddings, a detailed analysis of these methods are missing from the literature. We address this gap and present a macro and a micro analysis of KG embeddings learned by various KGE methods. Firstly, we present a geometrical analysis (i.e., arrangement of a group of vectors in vector space) of KG Embeddings. We present a set of metrics (i.e., Conicity, ATM) to enable such an analysis. Using these metric, we discover various insights and correlate it with task performance and other hyper-parameters. In the second half, we analyse the semantics of KG embeddings and propose an approach to learn more coherent dimensions. We formalize the notion of coherence using entity co-occurrence statistics and propose a regularizer term that maximizes coherence while learning KG embeddings.

Improved Link Prediction in Open Knowledge Graphs: We find that current OpenKG link prediction algorithms often predict noun phrases (NPs) with incompatible types for given noun and relation phrases. We address this problem in this work and propose OKGIT that improves OpenKG link prediction using novel type compatibility score and type regularization. To the best of our knowledge, this is the first work that addresses this problem. OKGIT can utilize NP canonicalization information while improving the type compatibility of predictions. With extensive experiments on multiple datasets, we show that the proposed method achieves state-of-the-art performance while producing type compatible NPs in the link prediction task.

An Adaptive Interaction Framework for Knowledge Graph Embeddings: Most of the existing KGE methods depend on a fixed set of interactions among dimensions of the head

and relation vectors. We address the problem of learning such interactions adaptively during training. We propose an adaptive interaction framework for KG embeddings that enables learning appropriate interactions while training. Some of the existing models can be seen as special cases of the proposed framework. We also present two new models based on this framework that outperforms the corresponding baselines on the link prediction task. Further analysis demonstrates the adaptability of these models to different datasets by learning appropriate interactions.

1.2 Organization of Thesis

The thesis is organized as follows: We present the basic notations and KGE learning framework in Chapter 2. In Chapter 3, we present the geometrical analysis of KG Embeddings. In Chapter 4, we present a method to induce interpretability in KG Embeddings. Then we present OKGIT, a method for OpenKG link prediction in Chapter 5. In Chapter 6, we present an adaptive interaction framework for learning KG Embeddings. Finally, we present our conclusions in Chapter 7 and point to some possible future directions of work.

Chapter 2

Background: Knowledge Graph Embeddings

This chapter provides a brief overview of Knowledge Graph Embedding methods, their training, and evaluation.

2.1 Introduction

Learning Knowledge Graph Embeddings (KGE) is a popular approach for representing KG entities and relations in various tasks. KG Embeddings capture local and global information about entities and relations in the KG, which are then helpful for KG tasks (e.g., link prediction, node classification) and extrinsic tasks (e.g., question answering). A central component of these methods is a score function that distinguishes correct triples from incorrect ones. Existing methods differ based on the form of the score function. The embeddings are then learned using a loss function (e.g., Binary Cross-Entropy). We describe the score function, training and evaluation of KG Embeddings in the following sections.

2.2 Notation

We denote a Knowledge Graph (KG) as $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$ where $\mathcal{E}$ is the set of entities, $\mathcal{R}$ is the set of relations and $\mathcal{T} \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ is the set of triples stored in the graph. Each triple $(h, r, t) \in \mathcal{T}$ denotes a relationship $r \in \mathcal{R}$ between two entities $h, t \in \mathcal{E}$. Here, h is called the subject or the head entity while t is called the object or the tail entity of the triple. r is the relation between these two entities. Most of the KG embedding methods learn vectors $\mathbf{e} \in \mathbb{R}^{d_e}$ for each entity $e \in \mathcal{E}$, and $\mathbf{r} \in \mathbb{R}^{d_r}$ for each relation $r \in \mathcal{R}$. Usually, $d_e = d_r$. Some methods also learn projection matrices $M_r \in \mathbb{R}^{d_r \times d_e}$ for relations. We use italics characters (e.g., h, r) to represent entities and

relations, and corresponding bold characters to represent their vector embeddings (e.g., $\mathbf{h}$, $\mathbf{r}$). The correctness of a triple is evaluated using a model specific score function $\psi : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightarrow \mathbb{R}$. For learning the embeddings, a loss function $\mathcal{L}(\mathcal{T}, \mathcal{T}'; \theta)$, defined over a set of positive triples $\mathcal{T}$, set of (sampled) negative triples $\mathcal{T}'$, and the parameters θ is optimized.

2.3 Score Function

A score function $\psi : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \mapsto \mathbb{R}$ is used for distinguishing correct triples in the KG from incorrect ones. It uses the vector embeddings for the entities h, t and relation r to generate a real-valued score $\psi(h, r, t)$ denoting the plausibility of the triple (h, r, t). It assigns high scores to correct triples and low scores to incorrect triples. For example, TransE [8] uses the following score function

$$\psi(h, r, t) = - \left\| \mathbf{h} + \mathbf{r} - \mathbf{t} \right\|_1 .$$

Based on the form of the score function, KGE methods can be categorized as an additive, multiplicative, or neural methods.

2.4 Training KG Embeddings

KGs contain only correct triples. However, a set of incorrect triples is required for training KG Embeddings. Therefore, negative sampling is used for generating incorrect triples [8]. A negative triple is generated by corrupting either the head or the tail entity in a triple $(h, r, t) \in \mathcal{T}$. Thus, for each correct triple $(h, r, t) \in \mathcal{T}$, we add two incorrect triples $(h', r, t) \notin \mathcal{T}$ and $(h, r, t') \notin \mathcal{T}$ where $h', t' \in \mathcal{E}$ are randomly selected entities. Let $\mathcal{T}'$ denote the set of all such negative triples. Finally, the set of positive and negative triples are used in a loss function $\mathcal{L}(\mathcal{T}, \mathcal{T}', \theta)$ for learning the embeddings.

Various loss functions have been used in the literature for training the KG Embeddings. Two of the most commonly used loss functions are described below.

Margin-based Ranking Loss [75]: The training is posed as a ranking problem where the positive triples are required to have higher scores than the negative triples, with a margin γ. The loss function is given as follows.

$$\mathcal{L}(\mathcal{T}, \mathcal{T}', \theta) = \sum_{(h,r,t) \in \mathcal{T}} \sum_{(h',r,t') \in \mathcal{T}'} \max\{\psi(h', r, t') - \psi(h, r, t) + \gamma, 0\}.$$

Binary Cross-Entropy Loss [19]: The training is posed as a classification problem where the positive triples have label $y = 1$ and the negative triples have label $y = 0$. The loss function is

given as follows.

$$\mathcal{L}(\mathcal{T}, \mathcal{T}', \theta) = \sum_{(h,r,t) \in \mathcal{T} \cup \mathcal{T}'} y_{(h,r,t)} \cdot \log(\sigma(\psi(h,r,t))) + (1 - y_{(h,r,t)}) \cdot \log(1 - \sigma(\psi(h,r,t)))$$

Here, σ denotes the logistic sigmoid function.

2.5 Evaluation of KG Embeddings: Link Prediction

The most common task for evaluating KG Embeddings is link prediction, i.e., the task of predicting unseen triples. It is posed as a ranking problem and ranking metrics such as mean rank (MR) and mean reciprocal rank (MRR) are used for evaluation. Given a held-out triple (h_i, r_i, t_i), all the entities $e \in \mathcal{E}$ are ranked as candidate tail entity based on their score $\psi(h_i, r_i, e)$. Let $rank_i^t$ denote the rank of the correct tail entity t. Similarly, the ranks can also be calculated for predicting head entity instead of tail entity. Let $rank_i^h$ denote the rank of the correct head entity h. These ranks can then be used for evaluating the KG embeddings using various ranking metrics. Commonly used metrics are given below.

$$\text{Mean Reciprocal Rank (MRR)} = \frac{1}{2 \times n_{test}} \sum_{i=1}^{n_{test}} \left(\frac{1}{rank_i^h} + \frac{1}{rank_i^t} \right),$$

$$\text{Mean Rank (MR)} = \frac{1}{2 \times n_{test}} \sum_{i=1}^{n_{test}} \left(rank_i^h + rank_i^t \right), \text{and}$$

$$\text{Hits@k} = \frac{1}{2 \times n_{test}} \sum_{i=1}^{n_{test}} \left(\mathbb{1}(rank_i^h \leq k) + \mathbb{1}(rank_i^t \leq k) \right).$$

Here, n_{test} denotes the number of held-out triples, and $\mathbb{1}$ is the indicator function. For Hits@k, commonly used values of k are 1, 3, and 10. A higher value of MRR and Hits@k and a lower value of MR indicate better KG Embeddings.

Chapter 3

Towards Understanding the Geometry of Knowledge Graph Embeddings

In this chapter, we present a geometrical analysis of KG Embeddings. Despite their popularity and effectiveness in various tasks (e.g., link prediction), geometric understanding of such embeddings (i.e., arrangement of entity and relation vectors in vector space) is unexplored. We initiate a study to analyze the geometry of KG embeddings and correlate it with task performance and other hyperparameters. To the best of our knowledge, this is the first study of its kind. Through extensive experiments on real-world datasets, we discover several insights. For example, we find that there are sharp differences between the geometry of embeddings learnt by different classes of KG embeddings methods. We hope that this initial study will inspire other follow-up research on this important but unexplored problem.

3.1 Introduction

The problem of learning KG Embeddings has received significant attention in recent years, with several methods being proposed [8, 39, 48, 50, 65, 19]. These methods represent entities and relations in a KG as vectors in high dimensional space. These vectors can then be used for various tasks, such as, link prediction, entity classification etc. Starting with TransE [8], there have been many KG embedding methods such as TransH [70], TransR [39] and STransE [48] which represent relations as translation vectors from head entities to tail entities. These are *additive models*, as the vectors interact via addition and subtraction. Other KG embedding models, such as, DistMult [75], HolE [50], and ComplEx [65] are *multiplicative* where entity-relation-entity triple likelihood is quantified by a multiplicative score function. All these methods employ a score function for distinguishing correct triples from incorrect ones.

In spite of the existence of many KG embedding methods, our understanding of the geometry and structure of such embeddings is very shallow. A recent work [45] analyzed the geometry of word embeddings. However, the problem of analyzing geometry of KG embeddings is still unexplored – we fill this important gap. In this chapter, we analyze the geometry of such vectors in terms of their lengths and conicity, which, as defined in Section 3.4, describes their positions and orientations in the vector space. We later study the effects of model type and training hyperparameters on the geometry of KG embeddings and correlate geometry with performance. We make the following contributions:

- We initiate a study to analyze the geometry of various Knowledge Graph (KG) embeddings. To the best of our knowledge, this is the first study of its kind. We also formalize various metrics which can be used to study the geometry of a set of vectors.
- Through extensive analysis, we discover several insights about the geometry of KG embeddings. For example, we find systematic differences between the geometries of embeddings learned by additive and multiplicative KG embedding methods.
- We also study the relationship between geometric attributes and predictive performance of the embeddings, resulting in several new insights. For example, in case of multiplicative models, we observe that for entity vectors generated with a fixed number of negative samples, lower conicity (as defined in Section 3.4) or higher average vector length lead to higher performance.

Source code of all the analysis tools developed as part of this chapter is available at https://github.com/malllabiisc/kg-geometry. We are hoping that these resources will enable one to quickly analyze the geometry of any KG embedding, and potentially other embeddings as well.

3.2 Related Work

In spite of the extensive and growing literature on both KG and non-KG embedding methods, very little attention has been paid towards understanding the geometry of the learned embeddings. A recent work [45] is an exception to this which addresses this problem in the context of word vectors. This work revealed a surprising correlation between word vector geometry and the number of negative samples used during training. Instead of word vectors, in this work we focus on understanding the geometry of KG embeddings. In spite of this difference, the insights we discover in this work generalizes some of the observations in the work of [45]. Please see Section 3.6.2 for more details.

Type	Model	Score Function $\psi(h, r, t)$
Additive	TransE [8]	$-\|\mathbf{h}+\mathbf{r}-\mathbf{t}\|_1$
	TransR [39]	$-\|M_r\mathbf{h}+\mathbf{r}-M_r\mathbf{t}\|_1$
	STransE [48]	$-\left\|M_r^1\mathbf{h}+\mathbf{r}-M_r^2\mathbf{t}\right\|_1$
Multiplicative	DistMult [75]	$\mathbf{r}^\top(\mathbf{h}\odot\mathbf{t})$
	HolE [50]	$\mathbf{r}^\top(\mathbf{h}\star\mathbf{t})$
	ComplEx [65]	$\mathbf{Re}(\mathbf{r}^\top(\mathbf{h}\odot\bar{\mathbf{t}}))$

Table 3.1: Summary of various Knowledge Graph (KG) embedding methods used in the chapter. Please see Section 3.3 for more details.

Since KGs contain only positive triples, negative sampling has been used for training KG embeddings. Effect of the number of negative samples in KG embedding performance was studied by [64]. In this work, we study the effect of the number of negative samples on KG embedding geometry as well as performance.

In addition to the additive and multiplicative KG embedding methods already mentioned in Section 3.1, there is another set of methods where the entity and relation vectors interact via a neural network. Examples of methods in this category include NTN [59], CONV [64], ConvE [19], R-GCN [56], ER-MLP [21] and ER-MLP-2n [53]. In this work, we primarily focus on the analysis of the geometry of additive and multiplicative KG embedding models. We also present our findings on neural models.

3.3 Overview of KG Embedding Methods

For our analysis, we consider six representative KG embedding methods: TransE [8], TransR [39], STransE [48], DistMult [75], HolE [50] and ComplEx [65]. We refer to TransE, TransR and STransE as *additive* methods because they learn embeddings by modeling relations as translation vectors from one entity to another, which results in vectors interacting via the addition operation during training. On the other hand, we refer to DistMult, HolE and ComplEx as *multiplicative* methods as they quantify the likelihood of a triple belonging to the KG through a multiplicative score function. The score functions optimized by these methods are summarized in Table 3.1.

Notation: Following Chapter 2, let $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$ be a Knowledge Graph (KG) where $\mathcal{E}$ is the set of entities, $\mathcal{R}$ is the set of relations and $\mathcal{T} \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ is the set of triples stored in the graph. Most of the KG embedding methods learn vectors $\mathbf{e} \in \mathbb{R}^{d_e}$ for $e \in \mathcal{E}$, and $\mathbf{r} \in \mathbb{R}^{d_r}$ for $r \in \mathcal{R}$. Some methods also learn projection matrices $M_r \in \mathbb{R}^{d_r \times d_e}$ for relations. The correctness of a triple is evaluated using a model specific score function $\psi : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \to \mathbb{R}$. For learning the embeddings, a loss function $\mathcal{L}(\mathcal{T}, \mathcal{T}'; \theta)$, defined over a set of positive triples $\mathcal{T}$, set of (sampled) negative triples $\mathcal{T}'$, and the parameters θ is optimized.

We use small italics characters (e.g., h, r) to represent entities and relations, and corresponding bold characters to represent their vector embeddings (e.g., $\mathbf{h}$, $\mathbf{r}$). We use bold capitalization (e.g., $\mathbf{V}$) to represent a set of vectors. Matrices are represented by capital italics characters (e.g., M).

3.3.1 Additive KG Embedding Methods

This is the set of methods where entity and relation vectors interact via additive operations. The score function for these models can be expressed as below

$$\psi(h, r, t) = -\left\| M_r^1 \mathbf{h} + \mathbf{r} - M_r^2 \mathbf{t} \right\|_1 \tag{3.1}$$

where $\mathbf{h}, \mathbf{t} \in \mathbb{R}^{d_e}$ and $\mathbf{r} \in \mathbb{R}^{d_r}$ are vectors for head entity, tail entity and relation respectively. $M_r^1, M_r^2 \in \mathbb{R}^{d_r \times d_e}$ are projection matrices from entity space $\mathbb{R}^{d_e}$ to relation space $\mathbb{R}^{d_r}$.

TransE [8] is the simplest additive model where the entity and relation vectors lie in same $d-$dimensional space, i.e., $d_e = d_r = d$. The projection matrices $M_r^1 = M_r^2 = \mathcal{I}_d$ are identity matrices. The relation vectors are modeled as translation vectors from head entity vectors to tail entity vectors. Pairwise ranking loss is then used to learn these vectors. Since the model is simple, it has limited capability in capturing many-to-one, one-to-many and many-to-many relations.

TransR [39] is another translation-based model which uses separate spaces for entity and relation vectors allowing it to address the shortcomings of TransE. Entity vectors are projected into a relation specific space using the corresponding projection matrix $M_r^1 = M_r^2 = M_r$. The training is similar to TransE.

STransE [48] is a generalization of TransR and uses different projection matrices for head and tail entity vectors. The training is similar to TransE. STransE achieves better performance than the previous methods but at the cost of more number of parameters.

Equation 3.1 is the score function used in STransE. TransE and TransR are special cases of STransE with $M_r^1 = M_r^2 = \mathcal{I}_d$ and $M_r^1 = M_r^2 = M_r$, respectively.

3.3.2 Multiplicative KG Embedding Methods

This is the set of methods where the vectors interact via multiplicative operations (usually dot product). The score function for these models can be expressed as

$$\psi(h, r, t) = \mathbf{r}^\top f(\mathbf{h}, \mathbf{t}) \tag{3.2}$$

where $\mathbf{h}, \mathbf{t}, \mathbf{r} \in \mathbb{F}^d$ are vectors for head entity, tail entity and relation respectively. $f(\mathbf{h}, \mathbf{t}) \in \mathbb{F}^d$

measures compatibility of head and tail entities and is specific to the model. $\mathbb{F}$ is either real space $\mathbb{R}$ or complex space $\mathbb{C}$. Detailed descriptions of the models we consider are as follows.

DistMult [75] models entities and relations as vectors in $\mathbb{R}^d$. It uses an entry-wise product ($\odot$) to measure compatibility between head and tail entities, while using logistic loss for training the model.

$$\psi_{DistMult}(h, r, t) = \mathbf{r}^\top(\mathbf{h} \odot \mathbf{t}) \tag{3.3}$$

Since the entry-wise product in (3.3) is symmetric, DistMult is not suitable for asymmetric and anti-symmetric relations.

HolE [50] also models entities and relations as vectors in $\mathbb{R}^d$. It uses circular correlation operator ($\star$) as compatibility function defined as

$$[\mathbf{h} \star \mathbf{t}]_k = \sum_{i=0}^{d-1} \mathbf{h}_i \mathbf{t}_{(k+i) \bmod d}$$

The score function is given as

$$\psi_{HolE}(h, r, t) = \mathbf{r}^\top(\mathbf{h} \star \mathbf{t}) \tag{3.4}$$

The circular correlation operator being asymmetric, can capture asymmetric and anti-symmetric relations, but at the cost of higher time complexity ($\mathcal{O}(d \log d)$). For training, we use pairwise ranking loss.

ComplEx [65] represents entities and relations as vectors in $\mathbb{C}^d$. The compatibility of entity pairs is measured using entry-wise product between head and complex conjugate of tail entity vectors.

$$\psi_{ComplEx}(h, r, t) = \mathbf{Re}(\mathbf{r}^\top(\mathbf{h} \odot \bar{\mathbf{t}})) \tag{3.5}$$

In contrast to (3.3), using complex vectors in (3.5) allows ComplEx to handle symmetric, asymmetric and anti-symmetric relations using the same score function. Similar to DistMult, logistic loss is used for training the model.

3.4 Metrics

For our geometrical analysis, we first define a term **'alignment to mean'** (**ATM**) of a vector $\mathbf{v}$ belonging to a set of vectors $\mathbf{V}$, as the cosine similarity[1] between $\mathbf{v}$ and the mean of all vectors

[1] $\text{cosine}(u, v) = \frac{u^\top v}{\|u\|\|v\|}$

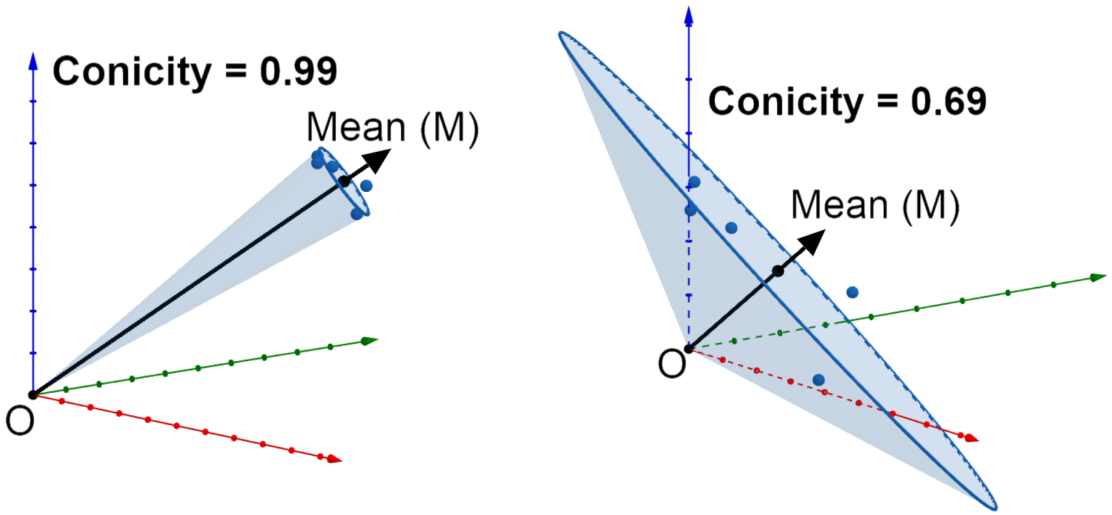


Figure 3.1: Comparison of high vs low Conicity. Randomly generated vectors are shown in blue with their sample mean vector M in black. Figure on the left shows the case when vectors lie in narrow cone resulting in high Conicity value. Figure on the right shows the case when vectors are spread out having relatively lower Conicity value. We skipped very low values of Conicity as it was difficult to visualize. The points are sampled from 3d Spherical Gaussian with mean (1,1,1) and standard deviation 0.1 (left) and 1.3 (right). Please refer to Section 3.4 for more details.

in $\mathbf{V}$.

$$\text{ATM}(\mathbf{v}, \mathbf{V}) = \text{cosine}\left(\mathbf{v}, \frac{1}{|\mathbf{V}|}\sum_{\mathbf{x}\in\mathbf{V}}\mathbf{x}\right)$$

We also define '**conicity**' of a set $\mathbf{V}$ as the mean ATM of all vectors in $\mathbf{V}$.

$$\text{Conicity}(\mathbf{V}) = \frac{1}{|\mathbf{V}|}\sum_{\mathbf{v}\in\mathbf{V}}\text{ATM}(\mathbf{v}, \mathbf{V})$$

By this definition, a high value of Conicity($\mathbf{V}$) would imply that the vectors in $\mathbf{V}$ lie in a narrow cone centered at origin. In other words, the vectors in the set $\mathbf{V}$ are highly aligned with each other. In addition to that, we define the variance of ATM across all vectors in $\mathbf{V}$, as the '**vector spread**'(VS) of set $\mathbf{V}$,

$$\text{VS}(\mathbf{V}) = \frac{1}{|\mathbf{V}|}\sum_{\mathbf{v}\in\mathbf{V}}\left(\text{ATM}(\mathbf{v}, \mathbf{V}) - \text{Conicity}(\mathbf{V})\right)^2$$

Figure 3.1 visually demonstrates these metrics for randomly generated 3-dimensional points. The left figure shows high Conicity and low vector spread while the right figure shows low

Dataset		FB15k	WN18
#Relations		1,345	18
#Entities		14,541	40,943
#Triples	Train	483,142	141,440
	Validation	50,000	5,000
	Test	59,071	5,000

Table 3.2: Summary of datasets used in the chapter.

Conicity and high vector spread.

We define the length of a vector $\mathbf{v}$ as L_2-norm of the vector $\|\mathbf{v}\|_2$ and **'average vector length'** (**AVL**) for the set of vectors $\mathbf{V}$ as

$$\text{AVL}(\mathbf{V}) = \frac{1}{|\mathbf{V}|} \sum_{\mathbf{v} \in \mathbf{V}} \|\mathbf{v}\|_2$$

3.5 Experimental Setup

Datasets: We run our experiments on subsets of two widely used datasets, viz., Freebase [6] and WordNet [44], called FB15k and WN18 [8], respectively. We detail the characteristics of these datasets in Table 3.2.

Please note that while the results presented in Section 3.6 are on the FB15K dataset, we reach the same conclusions on WN18. The plots for our experiments on WN18 can be found in Section 3.7.4.

Hyperparameters: We experiment with multiple values of hyperparameters to understand their effect on the geometry of KG embeddings. Specifically, we vary the dimension of the generated vectors between $\{50, 100, 200\}$ and the number of negative samples used during training between $\{1, 50, 100\}$. For other algorithm specific hyperparameters, we use the best reported values for hyper-parameters[1]. We have listed these hyper-parameters and their values in Table 3.3.

Frequency Bins: We follow [45] for entity and relation samples used in the analysis. Multiple bins of entities and relations are created based on their frequencies and 100 randomly sampled vectors are taken from each bin. These set of sampled vectors are then used for our analysis. For more information about sampling vectors, please refer to [45].

[1]For training, we used codes from https://github.com/Mrlyk423/Relation_Extraction (TransE, TransR), https://github.com/datquocnguyen/STransE (STransE), https://github.com/mnick/holographic-embeddings (HolE) and https://github.com/ttrouill/complex (ComplEx and DistMult).

Params	Additive Models		
	TransE	**TransR**	**STransE**
Rate	0.01	0.001	0.0001
Norm	L^1	L^1	L^1
#Epochs	1000	1000	1000
Loss	Pair-Loss	Pair-Loss	Pair-Loss
Margin	1	1	1
	Multiplicative Models		
	DistMult	**HolE**	**ComplEx**
L^2 **Reg.**	0.01	0	0.01
Rate	0.5	0.1	0.5
#Epochs	1000	500	1000
#Batches	100	100	100
Opt. algo.	AdaGrad	SGD	AdaGrad
Loss	Log-Loss	Pair-Loss	Log-Loss
Margin	-	0.2	-

Table 3.3: Hyper-parameter for all methods. Here Log-Loss and Pair-Loss refer to logistic loss and pairwise ranking loss respectively.

3.6 Results and Analysis

In this section, we evaluate the following questions.

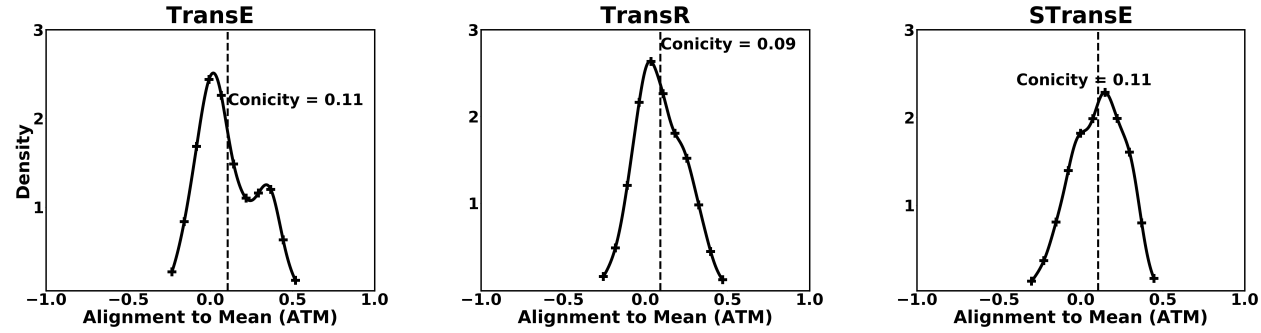
- Does model type (e.g., additive vs multiplicative) have any effect on the geometry of embeddings? (Section 3.6.1)
- Does negative sampling have any effect on the embedding geometry? (Section 3.6.2)
- Does the dimension of embedding have any effect on its geometry? (Section 3.6.3)
- How is task performance related to embedding geometry? (Section 3.6.4)

In each subsection, we summarize the main findings at the beginning, followed by evidence supporting those findings.

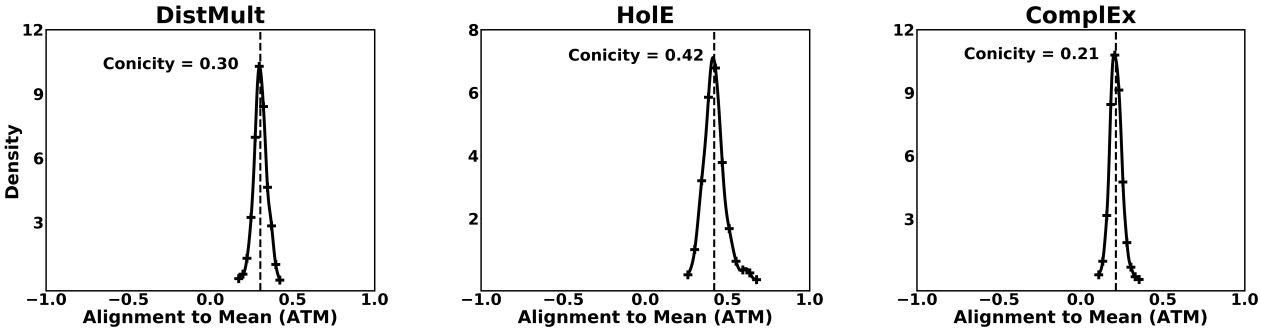
3.6.1 Effect of Model Type on Geometry

Summary of Findings:
Additive: Low conicity and high vector spread.
Multiplicative: High conicity and low vector spread.

In this section, we explore whether the type of the score function optimized during the training has any effect on the geometry of the resulting embedding. For this experiment, we set



(a) Additive Models



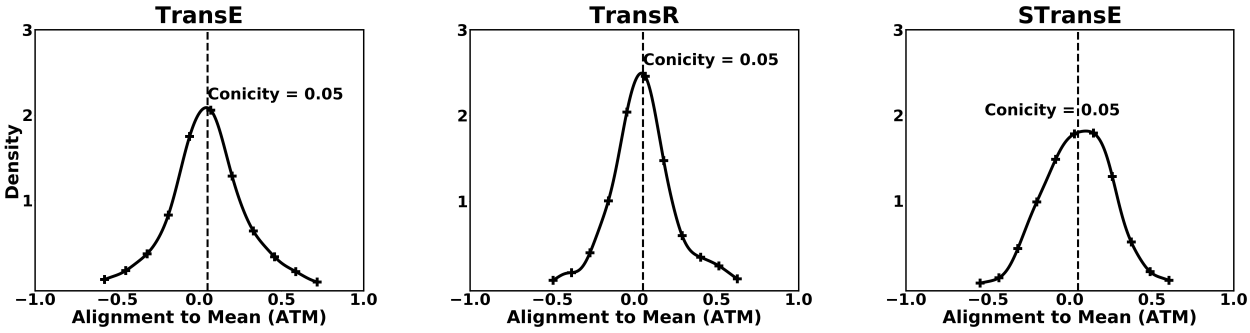
(b) Multiplicative Models

Figure 3.2: Alignment to Mean (ATM) vs Density plots for entity embeddings learned by various additive (top row) and multiplicative (bottom row) KG embedding methods. For each method, a plot averaged across entity frequency bins is shown. From these plots, we conclude that entity embeddings from additive models tend to have low (positive as well as negative) ATM and thereby low Conicity and high vector spread. Interestingly, this is reversed in case of multiplicative methods. Please see Section 3.6.1 for more details.

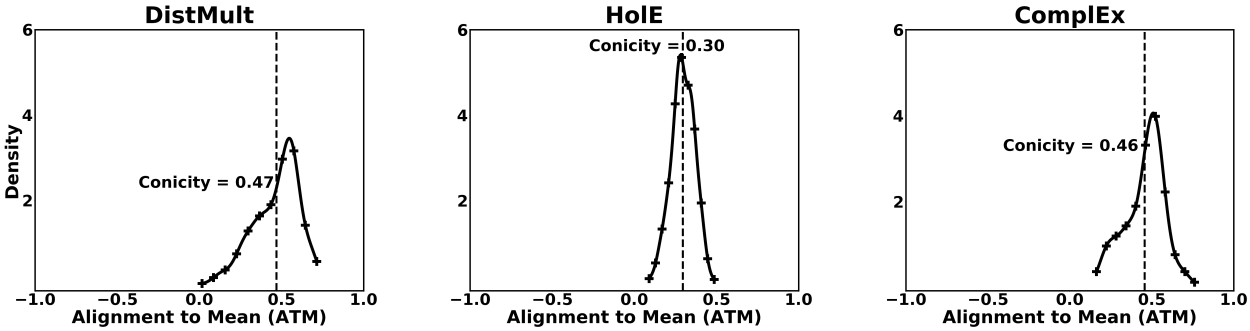
the number of negative samples to 1 and the vector dimension to 100 (we got similar results for 50-dimensional vectors). Figure 3.2 and Figure 3.3 show the distribution of ATMs of these sampled entity and relation vectors, respectively.[1]

Entity Embeddings: As seen in Figure 3.2, there is a stark difference between the geometries of entity vectors produced by additive and multiplicative models. The ATMs of all entity vectors produced by multiplicative models are positive with very low vector spread. Their high conicity suggests that they are not uniformly dispersed in the vector space, but lie in a narrow cone along the mean vector. This is in contrast to the entity vectors obtained from additive models which are both positive and negative with higher vector spread. From the lower values of conicity, we conclude that entity vectors from additive models are evenly dispersed in the vector space. This observation is also reinforced by looking at the high vector spread of additive models

[1]We also tried using the *global* mean instead of mean of the sampled set for calculating cosine similarity in ATM, and got very similar results.



(a) Additive Models



(b) Multiplicative Models

Figure 3.3: Alignment to Mean (ATM) vs Density plots for relation embeddings learned by various additive (top row) and multiplicative (bottom row) KG embedding methods. For each method, a plot averaged across entity frequency bins is shown. Trends in these plots are similar to those in Figure 3.2. Main findings from these plots are summarized in Section 3.6.1.

in comparison to that of multiplicative models. We also observed that additive models are sensitive to the frequency of entities, with high frequency bins having higher conicity than low frequency bins. However, no such pattern was observed for multiplicative models and conicity was consistently similar across frequency bins. For clarity, we have not shown different plots for individual frequency bins.

Relation Embeddings: As in entity embeddings, we observe a similar trend when we look at the distribution of ATMs for relation vectors in Figure 3.3. The conicity of relation vectors generated using additive models is almost zero across frequency bands. This coupled with the high vector spread observed, suggests that these vectors are scattered throughout the vector space. Relation vectors from multiplicative models exhibit high conicity and low vector spread, suggesting that they lie in a narrow cone centered at origin, like their entity counterparts.

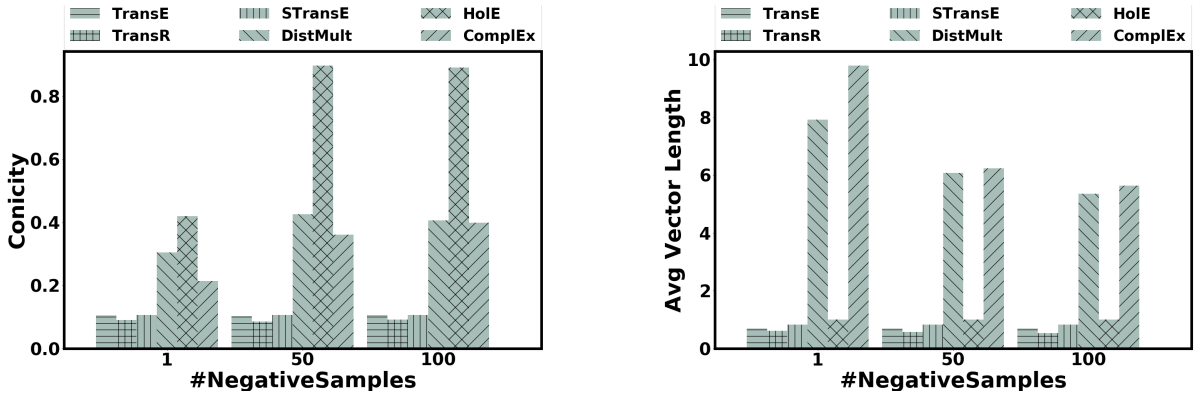


Figure 3.4: Conicity (left) and Average Vector Length (right) vs Number of negative samples for entity vectors learned using various KG embedding methods. In each bar group, first three models are additive, while the last three are multiplicative. Main findings from these plots are summarized in Section 3.6.2

3.6.2 Effect of Number of Negative Samples on Geometry

Summary of Findings:

Additive: Conicity and average length are invariant to changes in #NegativeSamples for both entities and relations.

Multiplicative: Conicity increases while average vector length decrease with increasing #NegativeSamples for entities. Conicity decreases, while average vector length remains constant (except HolE) for relations.

For experiments in this section, we keep the vector dimension constant at 100.

Entity Embeddings: As seen in Figure 3.4 (left), the conicity of entity vectors increases as the number of negative samples is increased for multiplicative models. In contrast, conicity of the entity vectors generated by additive models is unaffected by change in number of negative samples and they continue to be dispersed throughout the vector space. From Figure 3.4 (right), we observe that the average length of entity vectors produced by additive models is also invariant of any changes in number of negative samples. On the other hand, increase in negative sampling decreases the average entity vector length for all multiplicative models except HolE. The average entity vector length for HolE is nearly 1 for any number of negative samples, which is understandable considering it constrains the entity vectors to lie inside a unit ball [50]. This constraint is also enforced by the additive models: TransE, TransR, and STransE.

Relation Embeddings: Similar to entity embeddings, in case of relation vectors trained using additive models, the average length and conicity do not change while varying the number of negative samples. However, the conicity of relation vectors from multiplicative models decreases

with increase in negative sampling. The average relation vector length is invariant for all multiplicative methods, except for HolE. We see a surprisingly big jump in average relation vector length for HolE going from 1 to 50 negative samples, but it does not change after that. We refer the reader to the Section 3.7.3.1 for plots discussing the effect of number of negative samples on geometry of relation vectors.

We note that the multiplicative score between two vectors may be increased by either increasing the alignment between the two vectors (i.e., increasing Conicity and reducing vector spread between them), or by increasing their lengths. It is interesting to note that we see exactly these effects in the geometry of multiplicative methods analyzed above.

3.6.2.1 Correlation with Geometry of Word Embeddings

Our conclusions from the geometrical analysis of entity vectors produced by multiplicative models are similar to the results in [45], where increase in negative sampling leads to increased conicity of word vectors trained using the skip-gram with negative sampling (SGNS) method. On the other hand, additive models remain unaffected by these changes.

SGNS tries to maximize a score function of the form $\mathbf{w}^T \cdot \mathbf{c}$ for positive word context pairs, where $\mathbf{w}$ is the word vector and $\mathbf{c}$ is the context vector [43]. This is very similar to the score function of multiplicative models as seen in Table 3.1. Hence, SGNS can be considered as a multiplicative model in the word domain.

Hence, we argue that our result on the increase in negative samples increasing the conicity of vectors trained using a multiplicative score function can be considered as a generalization of the one in [45].

3.6.3 Effect of Vector Dimension on Geometry

Summary of Findings:
Additive: Conicity and average length are invariant to changes in dimension for both entities and relations.
Multiplicative: Conicity decreases for both entities and relations with increasing dimension. Average vector length increases for both entities and relations, except for HolE entities.

Entity Embeddings: To study the effect of vector dimension on conicity and length, we set the number of negative samples to 1, while varying the vector dimension. From Figure 3.5 (left), we observe that the conicity for entity vectors generated by any additive model is almost invariant of increase in dimension, though STransE exhibits a slight decrease. In contrast, entity vector from multiplicative models show a clear decreasing pattern with increasing dimension.

As seen in Figure 3.5 (right), the average lengths of entity vectors from multiplicative models

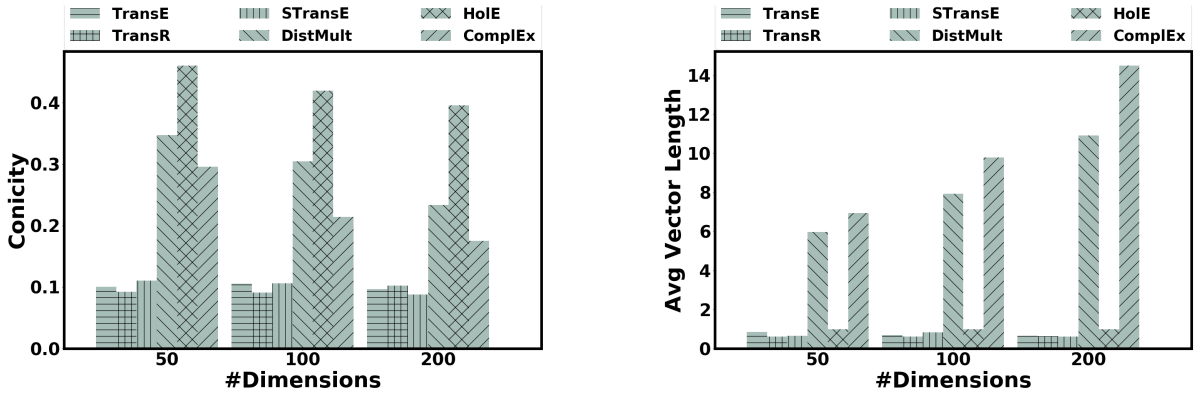


Figure 3.5: Conicity (left) and Average Vector Length (right) vs Number of Dimensions for entity vectors learned using various KG embedding methods. In each bar group, first three models are additive, while the last three are multiplicative. Main findings from these plots are summarized in Section 3.6.3.

increase sharply with increasing vector dimension, except for HolE. In case of HolE, the average vector length remains constant at one. Deviation involving HolE is expected as it enforces entity vectors to fall within a unit ball. Similar constraints are enforced on entity vectors for additive models as well. Thus, the average entity vector lengths are not affected by increasing vector dimension for all additive models.

Relation Embeddings: We reach similar conclusion when analyzing against increasing dimension the change in geometry of relation vectors produced using these KG embedding methods. In this setting, the average length of relation vectors learned by HolE also increases as dimension is increased. This is consistent with the other methods in the multiplicative family. This is because, unlike entity vectors, the lengths of relation vectors of HolE are not constrained to be less than unit length. The plots for relation vectors can be found in Section 3.7.3.2.

3.6.4 Relating Geometry to Performance

Summary of Findings:
Additive: Neither entities nor relations exhibit correlation between geometry and performance.
Multiplicative: Keeping negative samples fixed, lower conicity or higher average vector length for entities leads to improved performance. No relationship for relations.

In this section, we analyze the relationship between geometry and performance on the Link prediction task, using the same setting as in [8]. Figure 3.6 (left) presents the effects of conicity of entity vectors on performance, while Figure 3.6 (right) shows the effects of average entity

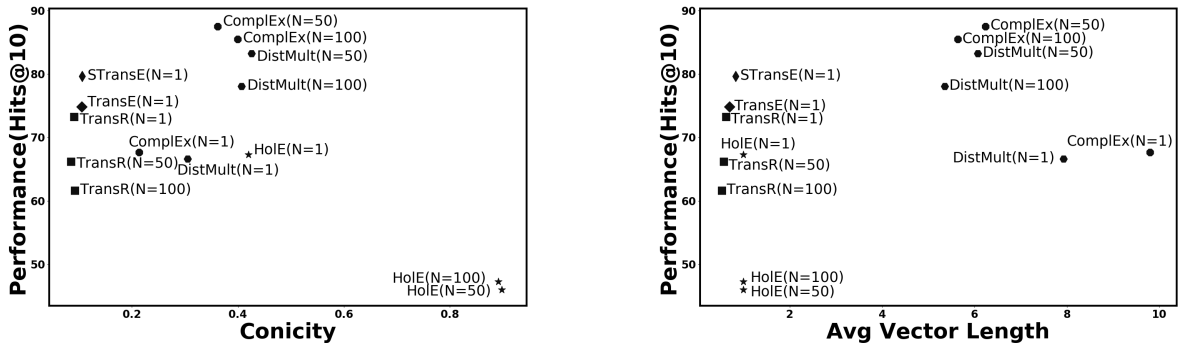


Figure 3.6: Relationship between Performance (HITS@10) on a link prediction task vs Conicity (left) and Avg. Vector Length (right). For each point, N represents the number of negative samples used. Main findings are summarized in Section 3.6.4.

vector length.[1]

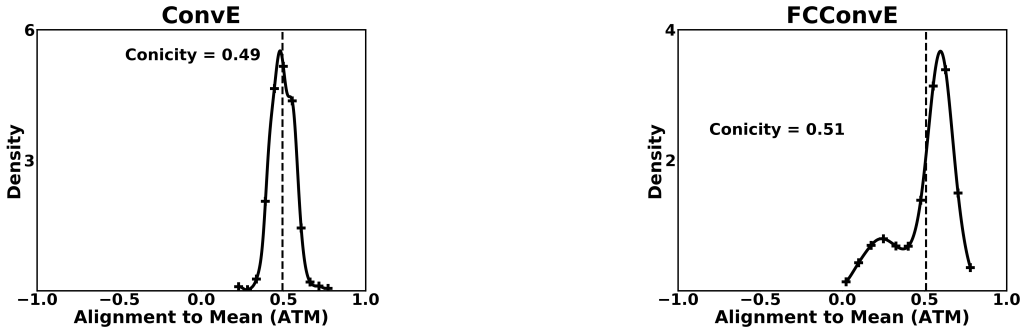
As we see from Figure 3.6 (left), for fixed number of negative samples, the multiplicative model with lower conicity of entity vectors achieves better performance. This performance gain is larger for higher numbers of negative samples (N). Additive models don't exhibit any relationship between performance and conicity, as they are all clustered around zero conicity, which is in-line with our observations in previous sections. In Figure 3.6 (right), for all multiplicative models except HolE, a higher average entity vector length translates to better performance, while the number of negative samples is kept fixed. Additive models and HolE don't exhibit any such patterns, as they are all clustered just below unit average entity vector length.

The above two observations for multiplicative models make intuitive sense, as lower conicity and higher average vector length would both translate to vectors being more dispersed in the space.

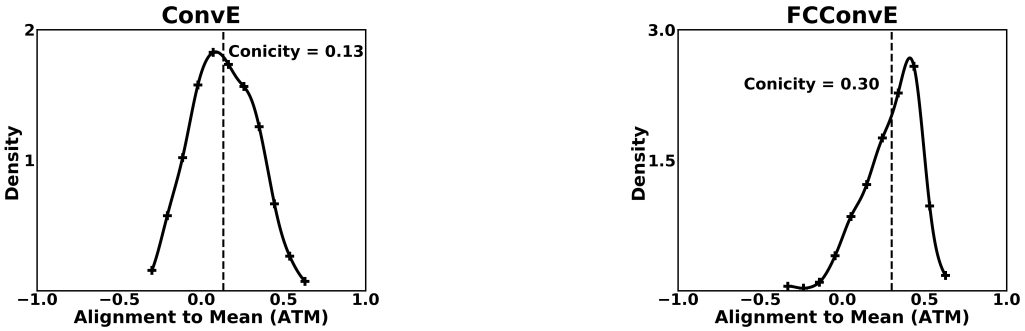
We see another interesting observation regarding the high sensitivity of HolE to the number of negative samples used during training. Using a large number of negative examples (e.g., $N = 50$ or 100) leads to very high conicity in case of HolE. Figure 3.6 (right) shows that average entity vector length of HolE is always one. These two observations point towards HolE's entity vectors lying in a tiny part of the space. This translates to HolE performing poorer than all other models in case of high numbers of negative sampling.

We also did a similar study for relation vectors, but did not see any discernible patterns.

[1] A more focused analysis for multiplicative models is presented in Section 3.7.2.



(a) Entity Embeddings



(b) Relation Embeddings

Figure 3.7: Alignment to Mean (ATM) vs Density plots for entity and relation embeddings learned by neural KG embedding methods. From these plots, we conclude that the density patterns for neural models are similar to multiplicative models. However, neural methods have higher conicity than multiplicative models. Please see Section 3.7.1.1 for more details.

3.7 Further Experiments and Results

This section presents an analysis of neural models, and additional results on relation vectors and the WN18 dataset.

3.7.1 Analysis of Neural Methods

In this section, we apply our analysis method on neural models. The neural models use various neural network architectures (e.g., Convolutional Neural Networks) for scoring triples in the KG. We use two representative methods ConvE [19] and FCConvE [14] for our analysis. These methods use Convolutional Neural Networks for scoring the triples. Please refer to Section 6.3 for more details about these methods. We perform the same experiments with neural methods (with same hyper-parameter values wherever applicable). The results are described in the following sections.

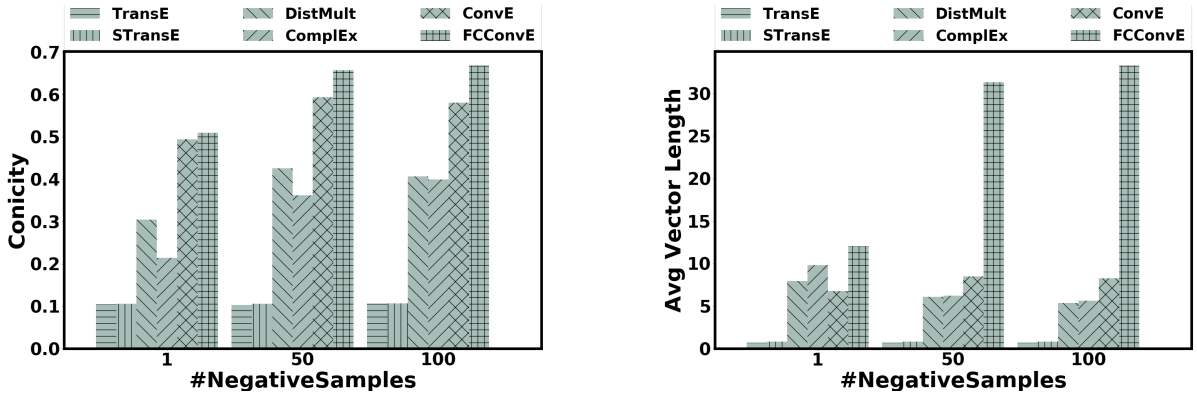


Figure 3.8: Conicity (left) and Average Vector Length (right) vs Number of negative samples for entity vectors. In each bar group, first two models are additive, the next two models are multiplicative, and the last two models are neural. For neural models, the Conicity and the average vector length increases with the number of negative samples (Section 3.7.1.2).

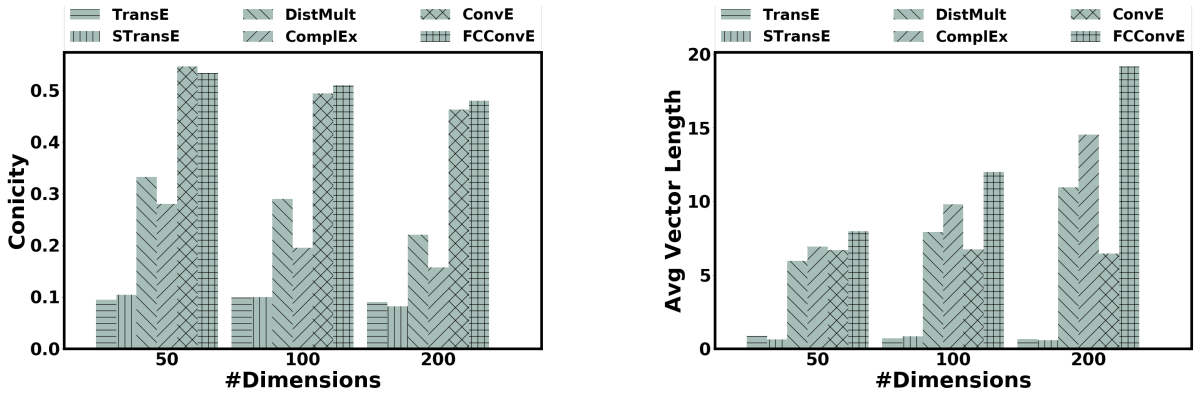


Figure 3.9: Conicity (left) and Average Vector Length (right) vs Number of Dimensions for entity vectors. Neural models show patterns similar to multiplicative models (Section 3.7.1.3).

3.7.1.1 Conicity of Neural Methods

Following the analysis from Section 3.6.1, we show the distribution of alignment to mean entity and relation vectors learned using neural models in Figure 3.7. For entity vectors, the ATMs are positive with low vector spread, resulting in high conicity. It suggests that the entity vectors lie in a narrow cone around the mean vector. While this pattern is similar to multiplicative models, neural models result in entity vectors with higher conicity.

For relation vectors, the ATMs have mixed value with moderate vector spread, resulting in medium to high conicity. Compared to additive and multiplicative models, the neural models show a mixed pattern for relation vectors. For both conicity and vector spread, the neural models have values in between additive and multiplicative methods.

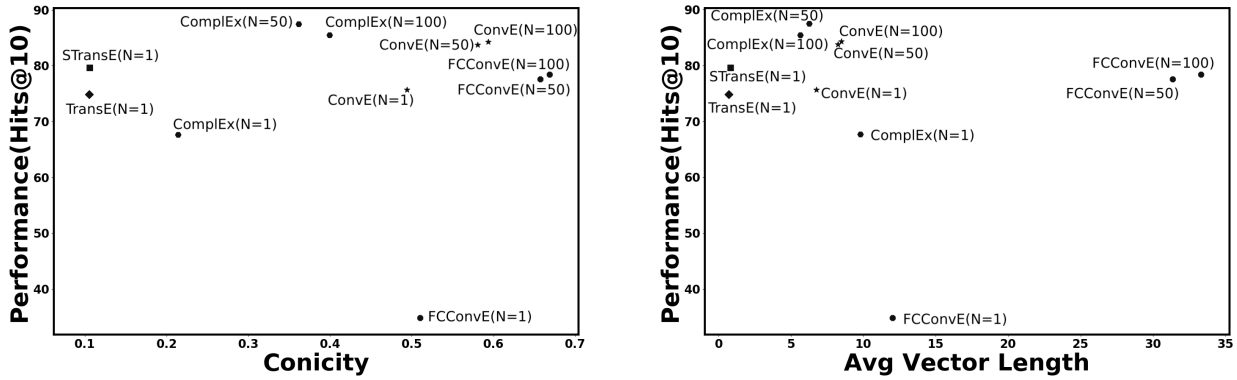


Figure 3.10: Relationship between Performance (HITS@10) on a link prediction task vs Conicity (left) and Average Vector Length (right). For each point, N represents the number of negative samples used during training. We find that neural methods show high Conicity and high average vector length (especially FCConvE) while having comparable performance with multiplicative models. Main findings are summarized in Section 3.7.1.4.

3.7.1.2 Effect of Number of Negative Samples

We study the effect of number of negative samples on the conicity and average length of entity vectors generated from neural models. We train the models with 100-dimensional vectors using varying number of negative samples (i.e., 1, 50, and 100). The results are shown in Figure 3.8. The conicity increases with number of negative samples for neural models. This pattern is similar to multiplicative models. However, the average vector length also increases with number of negatives for neural models, contrasting with multiplicative models. Further, this increase is more significant for FCConvE that ConvE.

3.7.1.3 Effect of Vector Dimension

Similar to Section 3.6.3, we study the effect of vector dimension on the conicity and average length of the learned entity vectors for neural models. The results are shown in Figure 3.9 The conicity of vectors decreases while the average vector length increases (except ConvE) with increasing number of dimension. This pattern is very similar to the multiplicative models. However, the average vector length for ConvE is almost invariant to changes in the number of dimension, similar to additive models.

3.7.1.4 Correlation with Performance

In this experiment, we analyse the correlation between the geometry of neural models with the performance on the link prediction task. We train the neural models with 100 dimensions and varying number of negative samples (shown as N in the figure). The correlation of performance

with conicity and average vector length is presented in Figure 3.10. For the clarity of presentation, we have used a small subset of additive and multiplicative models for this experiment. Similar to multiplicative models, we find that neural models with lower conicity have higher performance for a fixed number of negative samples. The same pattern is observed for average vector length as well. For a fixed number of negatives, the neural model with lower average vector length have higher performance.

3.7.2 Correlating Geometry with Performance for multiplicative models

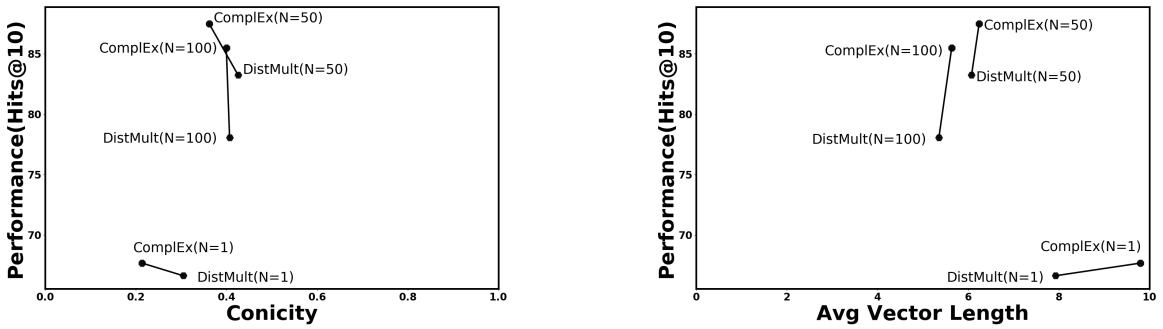


Figure 3.11: Relationship between Performance (HITS@10) on a link prediction task vs Conicity (left) and Avg. Vector Length (right) on FB15k dataset. For each point, N represents the number of negative samples used. Models with same number of negative samples are connected by line segment. This demonstrates that model performance has negative correlation with Conicity while positive correlation with average vector length for fixed number of negatives. Main findings are summarized in Section 3.6.4.

In this section, we present a subset of results already presented in Section 3.6.4 focusing on multiplicative models.[1] Figure 3.11 shows the correlation between model performance and geometry. Models which use same number of negative samples are connected with a line segment. In the left figure, the line segments have negative gradients. This suggests a negative correlation between model performance and Conicity for fixed number of negatives. In contrast, the line segments in the right figure have positive gradients which suggests a positive correlation between model performance and average vector length for fixed number of negatives. In both the cases, the magnitudes of the gradients of the line segments are larger for higher number of negative samples. This suggests that performance gain is more sensitive to geometry for higher number of negative samples.

[1]We have excluded HolE for clarity of the plots. The correlation between performance and conicity holds for HolE as well.

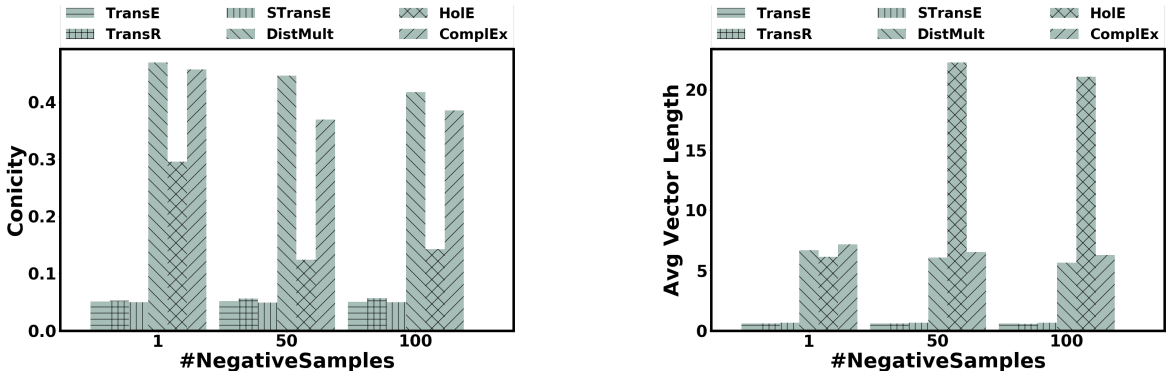


Figure 3.12: Conicity (left) and Average Vector Length (right) vs number of negative samples for relation vectors learned using various KG embedding methods on FB15k dataset. In each bar group, first three models are additive, while the last three are multiplicative.

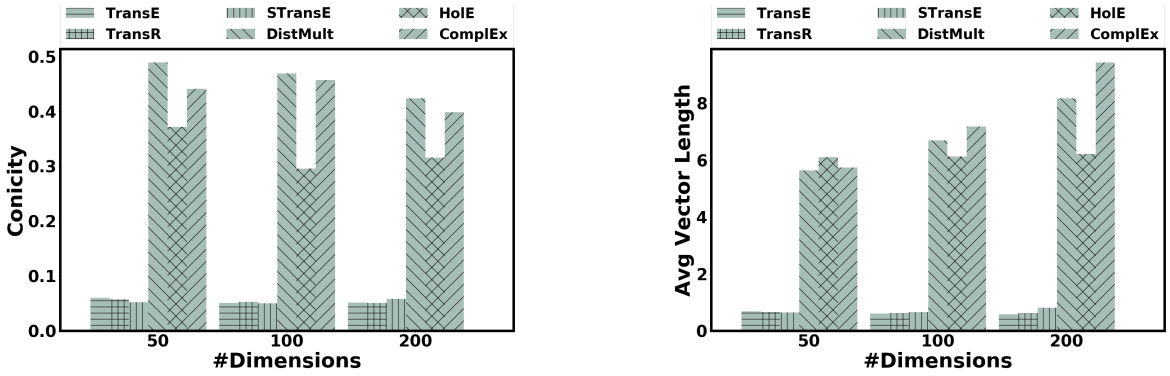


Figure 3.13: Conicity (left) and Average Vector Length (right) vs number of dimensions for relation vectors learned using various KG embedding methods on FB15k dataset. In each bar group, first three models are additive, while the last three are multiplicative.

3.7.3 Relation Vector Analysis for FB15k

In this section, we present the analysis of the relation vectors for FB15k with respect to number of negative samples and dimensions. They show very similar behavior as entity vectors and described in following sections.

3.7.3.1 Effect of Number of Negative Samples

Figure 3.12 (left) shows how the conicity of relation vectors varies with number of negative samples used. As observed in case of entity vectors, the conicity for relation vectors is invariant for additive models. However, the multiplicative models are sensitive to number of negative samples and the conicity of relation vectors show a small drop while increasing number of negative samples.[1] Similarly, as shown in Figure 3.12 (right), average vector length of relation

[1]Please note that all of these methods use negative sampling only for entities, not relations.

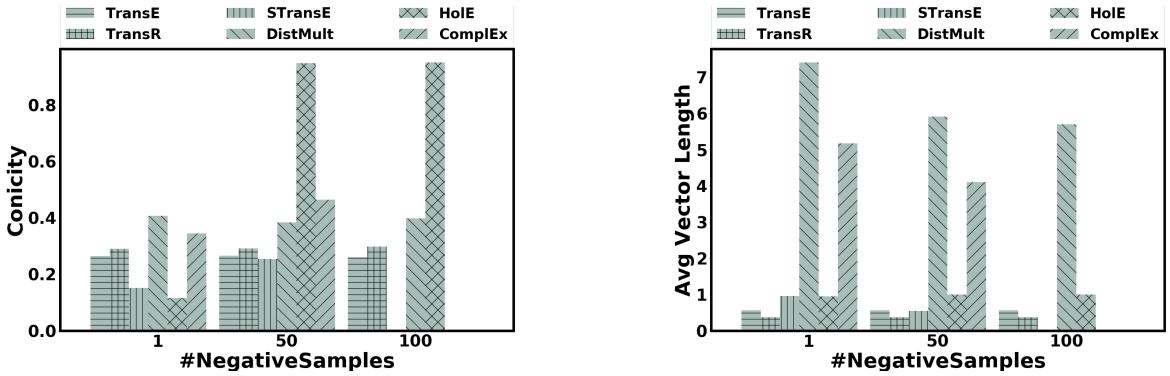


Figure 3.14: Conicity (left) and Average Vector Length (right) vs number of negative samples for entity vectors learned using various KG embedding methods on WN18 dataset. In each bar group, first three models are additive, while the last three are multiplicative.

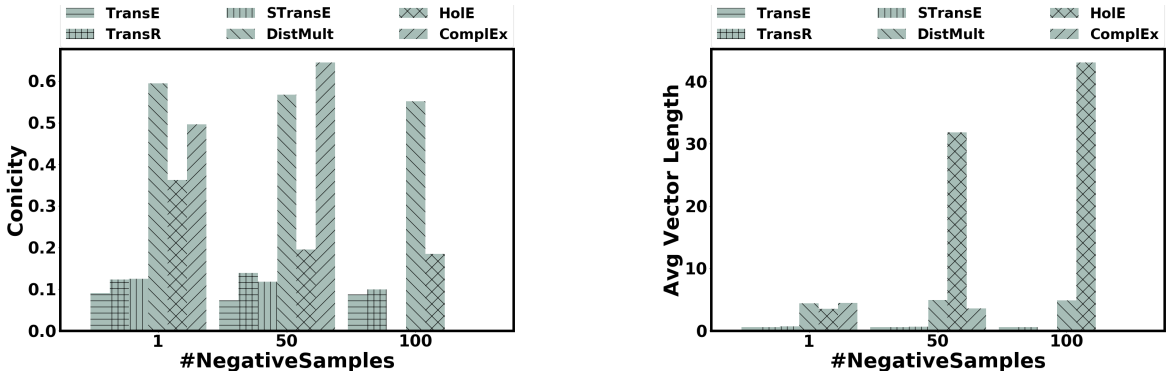


Figure 3.15: Conicity (left) and Average Vector Length (right) vs number of negative samples for relation vectors learned using various KG embedding methods on WN18 dataset. In each bar group, first three models are additive, while the last three are multiplicative.

vectors is not sensitive to the number of negative samples for additive models. Except HolE, average vector length is invariant to the number of negative samples for multiplicative models.

3.7.3.2 Effect of Vector Dimension on Geometry

Figure 3.13 demonstrates the effect of vector dimensions on the conicity (left) and average vector length (right) of relation vectors. As we can see from the figure, relation vectors behave very similar to entity vectors. The conicity of vectors generated from additive methods is almost invariant to increase in vector dimension. In contrast, multiplicative models show a decreasing pattern with increase in vector dimesion.

Similar to conicity, the average vector length of relation vectors generated from additive models is almost invariant to increase in vector dimension, except for STransE which shows a small increase for 200 dimension. However, average vector length of relation vectors generated

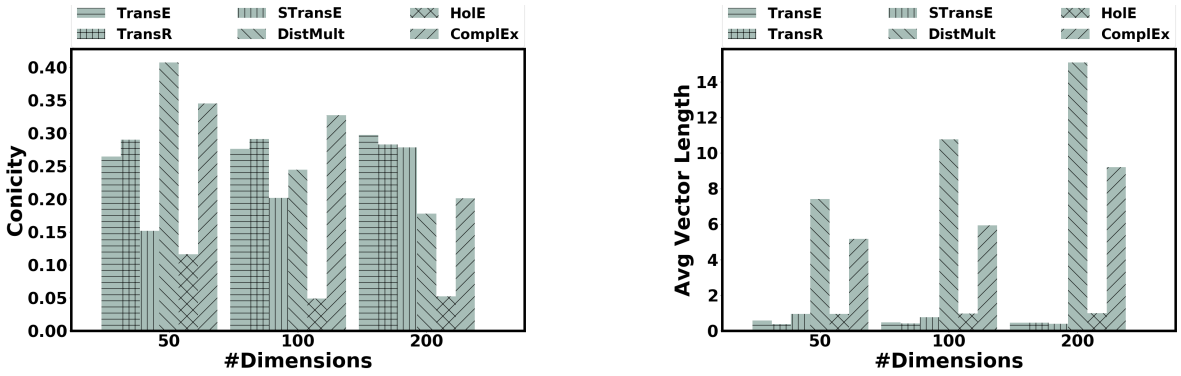


Figure 3.16: Conicity (left) and Average Vector Length (right) vs number of dimensions for entity vectors learned using various KG embedding methods on WN18 dataset. In each bar group, first three models are additive, while the last three are multiplicative.

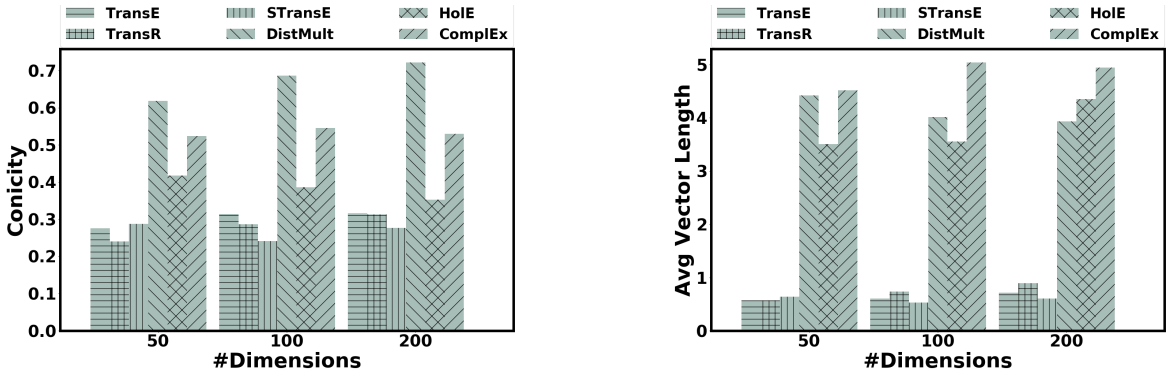


Figure 3.17: Conicity (left) and Average Vector Length (right) vs number of dimensions for relation vectors learned using various KG embedding methods on WN18 dataset. In each bar group, first three models are additive, while the last three are multiplicative.

from multiplicative models show a clear increasing pattern with increasing vector dimension.

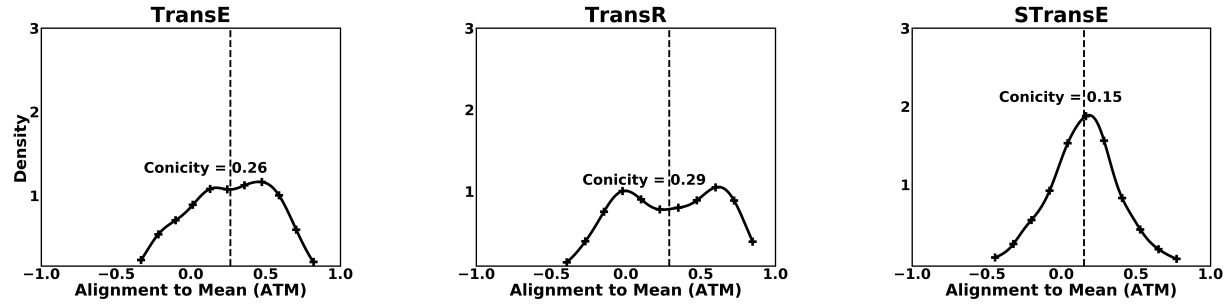
3.7.4 Analysis for WN18

In this section, we present the analysis of entity and relation embeddings for WN18 dataset. The observations are very similar to FB15k dataset and described in following sections.

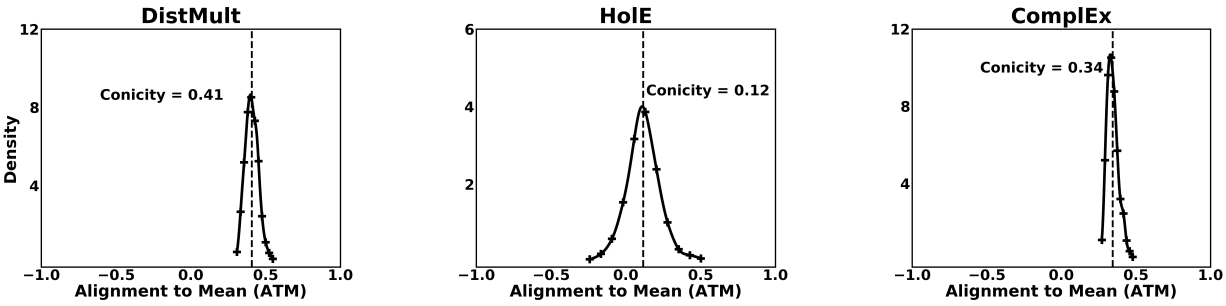
3.7.4.1 Effect of Number of Negative Samples on Geometry

For experiments in this section, we keep the vector dimesion constant at 50.

Entity Embeddings: The effect of number of negative samples on entity vector conicity (left) and average vector length (right) is shown in Figure 3.14. As seen from these figures, the conicity of entity vectors increases while average vector length decreases with number of negative samples for multiplicative models. Additive models, however, are not affected by increasing number of



(a) Additive Models



(b) Multiplicative Models

Figure 3.18: ATM vs Density plots for entity embeddings learned by various additive (top row) and multiplicative (bottom row) KG embedding methods on WN18 dataset. For each method, a plot averaged across entity frequency bins is shown.

negative samples. These observations are in agreement with observations from FB15k dataset.

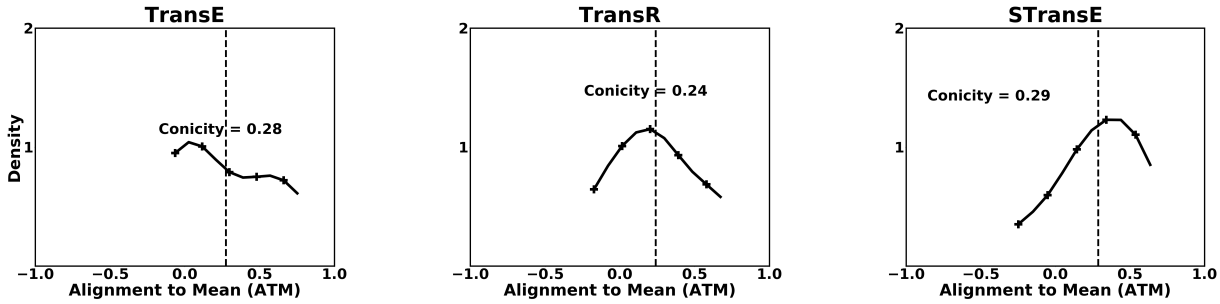
Relation Embeddings: As seen from Figure 3.15 (left), the conicity of relation vectors is invariant to number of negative samples for additive models. Except ComplEx, the conicity of relation vectors generated from multiplicative models show a decreasing pattern with increasing number of negative samples. Similar to FB15k, the average vector length of relation vectors is invariant to number of negative samples for both set of methods except HolE. HolE shows increase in average vector length with increasing number of negative samples, which is again consistent with FB15k.

For this analysis, we have skipped STransE and ComplEx models with 100 negative samples as these models reached system memory limit.

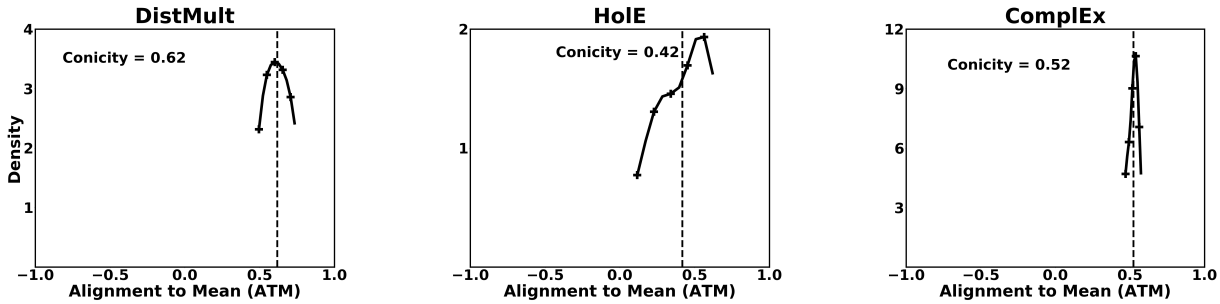
3.7.4.2 Effect of Vector Dimension on Geometry

For analysis in this section, we fixed the number of negative samples to 1.

Entity Embeddings: The effect of increasing vector dimension on conicity (left) and average vector length (right) is shown on Figure 3.16. Similar to FB15k, the conicity of entity vectors show a decreasing pattern for multiplicative models. Unlike TransR, TransE and STransE show an increasing pattern. In case of average vector length, all additive models are invariant to



(a) Additive Models



(b) Multiplicative Models

Figure 3.19: ATM vs Density plots for relation embeddings learned by various additive (top row) and multiplicative (bottom row) KG embedding methods on WN18 dataset. For each method, a plot averaged across relation frequency bins is shown.

increase in vector dimension. Except for HolE, average vector length of entity vectors increase with vector dimension for multiplicative models.

Relation Embeddings: Figure 3.17 shows the effect of increasing vector dimension on conicity (left) and average vector length (right). We do not see any discernible pattern here. The conicity increases slightly with vector dimensions for TransE, TransR and DistMult while it decreases for HolE. In case of average vector length, a slight increase is observed for TransE, TransR and HolE while DistMult shows a decreasing pattern.

3.7.4.3 Effect of Model Type on Geometry

Entity Embeddings: Figure 3.18 shows the distribution of ATM for entity vectors. Additive models (top row) exhibit high vector spread while multiplicative models (bottom row) show low vector spread. Except HolE, multiplicative models have higher conicity than additive models. This reinforces our observation that the vectors generated from multiplicative models tend to lie inside a narrow cone.

Relation Embeddings: Similar behaviour is observed for relation vectors in Figure 3.19 where all the multiplicative models have higher conicity and lower vector spread than additive models. Also, these observations are consistent with observations on FB15k dataset.

3.8 Conclusion

In this chapter, we have initiated a systematic study into the important but unexplored problem of analyzing geometry of various Knowledge Graph (KG) embedding methods. To the best of our knowledge, this is the first study of its kind. Through extensive experiments on multiple real-world datasets, we are able to identify several insights into the geometry of KG embeddings. We have also explored the relationship between KG embedding geometry and its task performance. We have released the source code to foster further research in this area.

Chapter 4

Inducing Interpretability in Knowledge Graph Embeddings

In the previous chapter, we studied a macro-behavior of a group of vectors and presented various insights. In this chapter, we focus on a micro-analysis of individual vectors. Specifically, we study the problem of inducing interpretability in Knowledge Graph (KG) embeddings. While there are many KG Embedding methods, most of these methods do not address the interpretability (semantics) of individual dimensions of the learned embeddings. In this chapter, we study this problem and propose a method for inducing interpretability in KG embeddings using entity co-occurrence statistics. The proposed method significantly improves the interpretability, while maintaining comparable performance in other KG tasks.

4.1 Introduction

Learning KG Embeddings have been an active area of research and many methods have been proposed, such as, [8, 54, 75, 64, 65, 56, 19, 3], etc. Some of these methods like [54] and [64] are capable of exploiting additional text data apart from the KG, resulting in better representations. These methods learn representations for entities and relations as vectors in a vector space, capturing global information about the KG. Although these methods have shown improved performance in the end task, they do not address the interpretability, i.e., understanding semantics of individual dimensions of the KG embedding. Such representations enable a better understanding of the model and can be helpful for explaining a model's decision on an end application.

In this work, we focus on incorporating interpretability in KG embeddings. Specifically, we aim to learn interpretable embeddings for KG entities by incorporating additional entity

co-occurrence statistics from text data. The notion of dimension-wise interpretability can help identify entities having the same underlying property and therefore reasoning about the predictions made using these embeddings. Similar notions have been used for the tasks of entity list completion and entity ranking [33]. Further, when a semantically coherent dimension represents a sensitive attribute, it can help estimate fairness in models that uses these embeddings [23].

This work is motivated by [38] who presented automated methods for evaluating topics learned via topic modelling methods. We adapt these methods for KG embedding models and propose a method to directly maximize them while learning KG embedding. As demonstrated by the experiments, we find that such modeling significantly improves interpretability, supporting our choice of using topic coherence for embedding dimensions. To the best of our knowledge, this work presents the first regularization term which induces interpretability in KG embeddings.

4.2 Related Work

Several methods have been proposed for learning KG embeddings. They differ on the modeling of entities and relations, usage of text data and interpretability of the learned embeddings. We summarize some of these methods in following sections.

4.2.1 KG Embedding models

Most of the KG embedding models represent entities and relations as vectors in $\mathbb{R}^{d_e}$ and $\mathbb{R}^{d_r}$ respectively (usually, d_e=d_r). A score function uses these vectors to calculate the correctness of a given triple. Based on the score function, these methods can be categorized as additive models [8, 39, 72, 73], multiplicative models [49, 75, 65, 3] and neural models [21, 19]. There are other methods which are able to incorporate text data while learning KG embeddings. For example, the method proposed in [54] assumes a combined universal schema of relations from KG as well as text. This method is further improved in [64] using textual relation encoder allowing parameter sharing among similar textual relations. However, none of these methods address the interpretability of the embeddings.

4.2.2 Interpretability of Embeddings

While the KG embedding models perform well in many tasks, the semantics of learned representations are not directly clear. This problem for word embeddings has been addressed in [47, 25, 60] where they apply a set of constraints inducing interpretability. A similar task of learning semantic features for entities and relations is KG was addressed in [71]. However, their approach is not applicable for the much popular KG embedding methods. The model proposed

in [73] can generate interpretable embeddings for relations, but not entities. Another approach, as proposed in [28], is to generate weighted Horn rules as explanations for link prediction. We refer the reader to Section 4 of [5] for further reading in this direction.

Our method differs from the previous works in the following aspects. Firstly, we focus on learning interpretable embeddings for KG entities rather than relations. Second, we incorporate side information about entities instead of constraints for inducing interpretability. Third, we use vector space modeling rather than probabilistic modelling (as in [71]) allowing the proposed method to be applicable to many existing KG embedding models.

4.3 Proposed Method

The proposed method is motivated by a measure of coherence in topic modeling literature [38]. This measure allows an automated evaluation of the quality of topics learned by topic modeling methods by using additional Point-wise Mutual Information (PMI) for word pairs. It was also shown to have a high correlation with the human evaluation of topics.

Based on this measure of coherence, we propose a regularization term. This term can be used with existing KG embedding methods for inducing interpretability. It is described in the following sections. The notations in this chapter are similar to Chapter 2.

4.3.1 Coherence

In topic models, the coherence of a topic can be determined by semantic relatedness among top entities within the topic. This idea can also be used in vector space models by treating dimensions of the vector space as topics. With this assumption, we can use a measure of coherence defined in the following section for evaluating the interpretability of the embeddings.

4.3.1.1 *Coherence@k*

Coherence for top k entities along dimension l is defined as follows.

$$Coherence@k^{(l)} = \sum_{i=2}^{k} \sum_{j=1}^{i-1} p_{ij} \tag{4.1}$$

where p_{ij} is PMI score between entities e_i and e_j extracted from text data. It is given as follows

$$p_{ij} = \log \left(\frac{Pr(e_i, e_j)}{Pr(e_i) \times Pr(e_j)} \right). \tag{4.2}$$

Here, $Pr(e_i, e_j)$ represents the joint probability of co-occurrence of entities e_i and e_j, while $Pr(e_i)$ and $Pr(e_j)$ represent the corresponding marginal probabilities, pre-computed using an

auxiliary corpus.

Coherence@k has been shown to have a high correlation with human interpretability of topics learned via various topic modeling methods [38]. Hence, we can expect interpretable embeddings by maximizing it.

Coherence@k for the entity embedding matrix θ_e is defined as the average over all dimensions.

$$Coherence@k = \frac{1}{d}\sum_{l=1}^{d} Coherence@k^{(l)}. \tag{4.3}$$

4.3.1.2 Inducing coherence while learning embeddings

We want to learn an embedding matrix θ_e that has high coherence (i.e., which maximizes *Coherence*@k). Since θ_e changes during training, the set of top k entities along each dimension varies over iterations. Hence, directly maximizing *Coherence*@k may not be feasible.

An alternative approach could be to promote higher values for entity pairs having a high PMI score p_{ij}. This will result in an embedding matrix θ_e with a high value of *Coherence*@k since high PMI entity pairs are more likely to be among top k entities.

This idea can be captured by following coherence term

$$\mathcal{C}(\theta_e, P) = \sum_{i=2}^{n}\sum_{j=1}^{i-1} \|\mathbf{e_i}^\intercal \mathbf{e_j} - p_{ij}\|^2 \tag{4.4}$$

where P is entity-pair PMI matrix and the boldface letter $\mathbf{e}$ denote vector for the entity e. This term can be used in the objective function defined in (4.7).

4.3.2 Entity Model (Model-E)

We use the Entity Model proposed in [54] for learning KG embeddings. However, it should be noted that the proposed regularizer can be used along with any KG embedding model which represents entities as vectors. Also, as pointed in [35, 55, 32], various KG embedding models achieve similar performances when trained properly. Therefore, we select Model-E, which is simple yet effective. This model assumes a vector $\mathbf{e}$ for each entity e and two vectors $\mathbf{r_s}$ and $\mathbf{r_o}$ for each relation r of the KG. The score for the triple (h, r, t) is given by,

$$\psi(h, r, t) = \mathbf{h}^\intercal \mathbf{r_s} + \mathbf{t}^\intercal \mathbf{r_o}. \tag{4.5}$$

Training these vectors requires incorrect triples. So, we use the closed world assumption. For each triple $x \in \mathcal{T}$, we create two negative triples x_o^- and x_s^- by corrupting the object and

subject of the triples, respectively, such that the corrupted triples do not appear in training, test or validation data. The loss for a triple pair is defined as $loss(x, x^-) = -\log(\sigma(\psi(x) - \psi(x^-)))$. Then, the aggregate loss function is defined as

$$L(\theta_e, \theta_r, \mathcal{T}) = \frac{1}{|\mathcal{T}|} \sum_{x \in \mathcal{T}} \left(loss(x, x_o^-) + loss(x, x_s^-) \right). \tag{4.6}$$

4.3.3 Objective

The overall loss function can be written as follows

$$L(\theta_e, \theta_r, \mathcal{T}) + \lambda_c \mathcal{C}(\theta_e, P) + \lambda_r \mathcal{R}(\theta_e, \theta_r) \tag{4.7}$$

where $\mathcal{R}(\theta_e, \theta_r) = \frac{1}{2} \left(\|\theta_e\|^2 + \|\theta_r\|^2 \right)$ is the $L2$ regularization term and λ_c and λ_r are hyper-parameters controlling the trade-off among different terms in the objective function.

4.4 Experiments and Results

4.4.1 Datasets

We use the FB15k-237 [63] dataset, a factual KG, for experiments. It contains 14541 entities and 237 relations. The triples are split into training, validation and test set having 272115, 17535 and 20466 triples respectively. For extracting entity co-occurrences, we use the textual relations used in [64]. It contains around 3.7 millions textual triples. Each textual triple contains a subject entity mention, a textual relation, and a tail entity mention. It also contains the frequency of the triple in the corpus. The entity mentions in these textual triples are linked to the entities the KG. We use it for calculating entity co-occurences counts and PMI for entity pairs.

4.4.2 Experimental Setup

We use the method proposed in [54] as the baseline. Please refer to Section 4.3.2 for more details. For evaluating the learned embeddings, we test them on different tasks. All the hyper-parameters are tuned using performance (MRR) on validation data. We use 100 dimensions after cross validating among 50, 100 and 200 dimensions. For regularization, we use $\lambda_r = 0.01$ (from $10, 1, 0.1, 0.01$) and $\lambda_c = 0.01$ (from $10, 1, 0.1, 0.01$) for $L2$ and coherence regularization respectively. We use multiple random initializations sampled from a Gaussian distribution. For optimization, we use gradient descent and stop optimization when gradient becomes 0 upto 3 decimal places. The final performance measures are reported for test data.

Method	**Link Prediction**		
	MRR↑	MR↓	Hits@10(%)↑
Baseline	$\mathbf{31.6 \pm 0.08}$	121.9 ± 1.80	$\mathbf{48.3 \pm 0.39}$
Proposed	30.4 ± 0.08	$\mathbf{111.9 \pm 1.12}$	46.8 ± 0.08
	Triple Classification		
	AUC(%)↑	Accuracy(%)↑	
Baseline	72.9 ± 0.16	63.2 ± 0.50	
Proposed	$\mathbf{73.2 \pm 0.28}$	$\mathbf{67.6 \pm 0.17}$	
	Interpretability		
	AutoWI@5(%)↑	Coherence@5↑	Manual WI(%)↑
Baseline	6 ± 4.14	-47.4 ± 4.68	12
Proposed	$\mathbf{66 \pm 5.89}$	$\mathbf{-12.5 \pm 4.48}$	**84**

Table 4.1: Results of various tasks on FB15k-237 dataset. Here ↑ indicates higher values are better while ↓ indicates lower values are better. The proposed method significantly improves interpretability while maintaining comparable performance on KG tasks (4.4.3).

4.4.3 Results

In following sections, we compare the performance of the proposed method with the baseline method in different tasks. Please refer to Table 4.1 for results.

4.4.3.1 Interpretability

For evaluating the interpretability, we use *Coherence*@k (4.3), automated and manual word intrusion tests. In word intrusion test [16], top $k(=5)$ entities along a dimension are mixed with the bottom most entity (the intruder) in that dimension and shuffled. Then multiple (3 in our case) human annotators are asked to find out the intruder. We use majority voting to finalize one intruder. Amazon Mechanical Turk was used for crowdsourcing the annotation task and we used 25 randomly selected dimensions for evaluation. Thus, each of the three annotators evaluates 25 examples. Following previous works [38, 47, 25], we use $k = 5$ for each dimension. Using more dimensions may not be suitable for manual evaluation. For automated word intrusion [38], we calculate following score for all $k + 1$ entities

$$\text{AutoWI}(e_i) = \sum_{j=1, j\neq i}^{k+1} p_{ij} \tag{4.8}$$

where p_{ij} are the PMI scores. The entity having least score is identified as the intruder. We report the fraction of dimensions for which we were able to identify the intruder correctly.

As we can see in Table 4.1, the proposed method achieves better values for *Coherence*@5 as a direct consequence of the regularization term, thereby maximizing coherence between appropriate entities. Performance on the word intrusion task also improves drastically as the intruder along each dimension is a lot easier to identify owing to the fact that the top entities for each dimension group together more conspicuously. For the manual word intrusion task, we also estimate the inter-annotator agreement scores. The agreement scores for the proposed method (average Cohen's Kappa = 0.85 and Fleiss' Kappa = 0.85) was higher than for the baseline (average Cohen's Kappa = 0.34 and Fleiss' Kappa = 0.34). Further, all the annotators agreed on the same option for 84% of the examples for the proposed method as compared to 28% for the baseline. Please refer to the Table 4.2 for the pairwise agreement scores.

Annotator Pair	Baseline	Proposed	Average
(A1, A2)	0.32	0.80	0.56
(A1, A3)	0.27	0.85	0.56
(A2, A3)	0.42	0.90	0.66
Average	0.34	0.85	0.59

Table 4.2: The pairwise inter-annotator agreement scores (Cohen's Kappa) for three annotators A1, A2, and A3. The annotators have a better agreement on the examples from the proposed method compared to the baseline (4.4.3.1).

4.4.3.2 Link Prediction

In this experiment, we test the model's ability to predict the best object entity for a given subject entity and relation. For each of the triples, we fix the subject and the relation and rank all entities (within same category as true object entity) based on their score according to (4.5). We report Mean Rank (MR) and Mean Reciprocal rank (MRR) of the true object entity and Hits@10 (the number of times true object entity is ranked in top 10) as percentage. A good model should have higher values for MRR and Hits@10, and lower value for MR.

The coherence regularization term's objective, being tangential to that of the original loss function, is not expected to affect the link prediction task's performance. However, the results show a trivial drop of 1.2 in MRR. Upon further inspection, we found that the coherence term gives credibility to certain triples otherwise deemed incorrect by the closed world assumption. These triples appear in the text corpus and contain entity pairs with high PMI values.

[2]We have used abbreviations for BS (Bachelor of Science), MS (Master of Science), UK (United Kingdom) and USA (United States of America). They appear as full form in the data.

Top 5
Baseline
-**Jurist**, **Pipe organ**, USA, **Lions Gate Entertainment**, UK -Guitar, **71st Academy Awards**, **Jurist**, Piano, Bass guitar -Actor, **Official Website**, Screenwriter, Film Producer, **USA** -**Jurist**, USA, **Marriage**, **Male**, UK -**Pipe organ**, **Official Website**, Actor, Film Producer, Screenwriter
Proposed Method
-Juris Doctor, Business Administration, Biology, Psychology, BS -Bachelor of Arts, PhD, Bachelor's degree, BS, MS -European Union, Europe, Netherlands, Portugal, **Government** -UK, Hollywood, **DVD**, London, Europe -Hollywood, Academy Awards, **USA**, DVD, **Los Angeles**

Table 4.3: Top 5 entities for randomly selected dimensions. As we see, the proposed method produces more coherent entities compared to the baseline. Incoherent entities are marked in bold face. [2]

4.4.3.3 Triple Classification

In this experiment, we test the model on classifying correct and incorrect triples. For finding incorrect triples, we corrupt the object entity with a randomly selected entity within the same category. For classification, we use validation data to find the best threshold for each relation by training an SVM classifier and later use this threshold for classifying test triples. We report the mean accuracy and mean AUC over all relations.

We observe that the proposed method achieves slightly better performance for triple classification improving the accuracy by 4.4. The PMI information adds more evidence to the correct triples which are related in text data, generating a better threshold that more accurately distinguishes correct and incorrect triples.

4.4.4 Qualitative Analysis of Results

Since our aim is to induce interpretability in representations, in this section, we evaluate the embeddings learned by the baseline as well as the proposed method. For both methods, we select some dimensions randomly and present top 5 entities along those dimensions. As we can see from the results in Table 4.3, the proposed method produces more coherent entities than the baseline method.

4.5 Conclusion

In this chapter, we proposed a method for inducing interpretability in KG embeddings using a coherence regularization term. We evaluated the proposed and the baseline method on the interpretability of the learned embeddings. We also evaluated the methods on different KG tasks and compared their performance. We found that the proposed method achieves better interpretability while maintaining comparable performance on KG tasks.

Chapter 5

OKGIT: Open Knowledge Graph Link Prediction with Implicit Types

Previous chapters presented various analysis of existing KG Embedding methods. In this chapter, we address a different limitation of these methods. Specifically, we focus on a special type of KGs, called Open Knowledge Graph (OpenKG), where existing KG Embedding methods fall short. OpenKGs refer to a set of (head noun phrase, relation phrase, tail noun phrase) triples such as (tesla, return to, new york) extracted from a corpus using OpenIE tools. While OpenKGs are easy to bootstrap for a domain, they are very sparse and far from being directly usable in an end task. Therefore, the task of predicting new facts, i.e., link prediction, becomes an important step while using these graphs in downstream tasks such as text comprehension, question answering, and web search query recommendation. Learning embeddings for OpenKGs is one approach for link prediction that has received some attention lately. However, on careful examination, we found that current OpenKG link prediction algorithms often predict noun phrases (NPs) with incompatible types for given noun and relation phrases. We address this problem in this chapter and propose OKGIT that improves OpenKG link prediction using novel type compatibility score and type regularization. With extensive experiments on multiple datasets, we show that the proposed method achieves state-of-the-art performance while producing type compatible NPs in the link prediction task.

5.1 Introduction

An Open Knowledge Graph (OpenKG) is a set of factual triples extracted from a text corpus using Open Information Extraction (OpenIE) tools such as TextRunner [4] and ReVerb [24]. These triples are of the form (noun phrase, relation phrase, noun phrase), e.g., (*tesla*, *return*

Triple	(*tesla, return to, ?*)				
CaRE	*polytechnic institute*	*2009*	*1986*	*jp morgan*	*patent*
BERT	*chicago*	*earth*	*england*	*america*	*detroit*
OKGIT	*new york*	*america*	*paris*	*california*	*london*

Table 5.1: Some sample tail NP predictions by CaRE, BERT, and OKGIT. The true tail NP is underlined. As we can see, both CaRE and BERT fail to predict the correct tail NP. However, BERT predictions are type compatible with the query. OKGIT predicts the correct NP while improving the type compatibility with the query.

to, *new york*). An OpenKG can be viewed as a multi-relational graph where the noun phrases (NPs) are the nodes, and the relation phrases (RPs) are the labeled edges between pairs of nodes. It is easy to bootstrap OpenKGs from a domain-specific corpus, making them suitable for newer domains. However, they are extremely sparse and may not be directly usable for an end task. Therefore, tasks such as NP canonicalization (merging mentions of the same entity) and link prediction (predicting new facts) become an important step in downstream applications. Some example applications are text comprehension [42], relation schema induction [51], canonicalization [66], question answering [77], and web search query recommendation [30]. In this work, we focus on improving OpenKG link prediction.

Although OpenKGs are structurally similar to Ontological KGs, they come with a different set of challenges. They are extremely sparse, NPs and RPs are not canonicalized, and no type information is present for NPs. There has been much work on learning embeddings for Ontological KGs in the past years. However, this task has not received much attention in the context of OpenKGs. CaRE [27] is a recent method which addresses this problem. It learns embeddings for NPs and RPs in an OpenKG while incorporating NP canonicalization information. However, even after incorporating canonicalization, we find that CaRE struggles to predict NPs whose types are compatible with given head NP and RP.

As observed by Petroni et al. [52], modern pre-trained language representation models like BERT can store factual knowledge and can be used to perform link prediction in KGs. However, in our explorations with OpenKGs, we found that even though BERT may not predict the correct NP on the top, it predicts type compatible NPs (Table 5.1). A similar observation was also made in the context of entity linking [17]. As OpenKGs do not have any underlying ontology and obtaining type information can be expensive, BERT predictions can help improve OpenKG link prediction.

Motivated by this, we employ BERT for improving OpenKG link prediction, using novel type compatibility score (Section 5.4.2) and type regularizer term (Section 5.4.4). We propose

OKGIT, a method for OpenKG link prediction with improved type compatibility. We test our model on multiple datasets and show that it achieves state-of-the-art performance on all of these datasets.

We make the following contributions:

- We address the problem of OpenKG link prediction, focusing on improving type compatibility of predictions. To the best of our knowledge, this is the first work that addresses this problem.
- We propose OKGIT, a method for OpenKG link prediction with novel type compatibility score and type regularization. OKGIT can utilize NP canonicalization information while improving the type compatibility of predictions.
- We evaluate OKGIT on the link prediction across multiple datasets and observe that it outperforms the baseline methods. We also demonstrate that the learned model generates more type compatible predictions.

Source code for the proposed model and the experiments from this chapter is available at https://github.com/Chandrahasd/OKGIT.

5.2 Related Work

OpenKG Embeddings: Learning embeddings for OpenKGs has been a relatively under-explored area of research. Previous work using OpenKG embeddings has primarily focused on canonicalization. CESI [66] uses KG embedding models for the canonicalization of noun phrases in OpenKGs. The problem of incorporating canonicalization information into OpenKG embeddings was addressed by Gupta et al. [27]. Their method for OpenKG embeddings (i.e., CaRE) performs better than Ontological KG embedding baselines in terms of link prediction performance. The challenges in the link prediction for OpenKGs were discussed in Broscheit et al. [9], and methods similar to CaRE were proposed. In spirit, CaRE [27] comes closest to our model; however, they do not address the problem of type compatibility in the link prediction task.

Entity Type: Entity typing is a popular problem where given a sentence and an entity mention, the goal is to predict *explicit* types of the entity. It has been an active area of research, and many models and datasets, such as [41], [29], and [18], have been proposed. However, unlike this task, we aim to incorporate unsupervised **implicit** type information present in the pre-trained BERT model into OpenKG embeddings, rather than predicting *explicit* entity types present in ontologies or corpora.

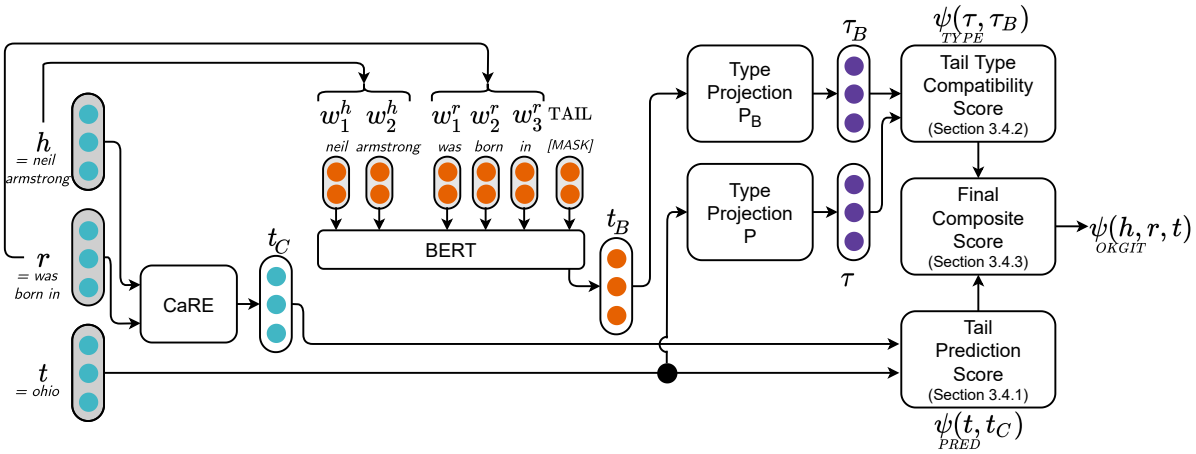


Figure 5.1: OKGIT Architecture. OKGIT learns embeddings for Noun Phrases (NP) and Relation Phrases (RP) present in an OpenKG by augmenting a standard tail prediction loss with type compatibility loss. Guidance for the tail type is obtained through type projection out of BERT's tail embedding prediction. In the figure, h, r, and t are the head NP, relation (RP), and tail NP. $h = w_1^h \dots w_{k_h}^h$ and $r = w_1^r \dots w_{k_r}^r$ are tokens in the head NP and relation, respectively. t_C and t_B are the tail NP vectors predicted by CaRE and BERT models (Please see Section 5.3 for background on these two models). Vectors τ_B and τ are the type vectors obtained using type projections P_B and P, respectively. ψ_{PRED} represents tail prediction score (Section 5.4.1) while ψ_{TYPE} represents type compatibility score (Section 5.4.2). ψ_{OKGIT} is the combined score generated by OKGIT for the input triple (h, r, t) (Section 5.4.3). Please refer to Section 5.4 for more details.

For unsupervised cases, the problem of type compatibility in link prediction was addressed in [31]. They employ a type compatibility score by learning a type vector for each NP and two type vectors (head and tail) for each relation. This score is multiplied with the triple score function, and the type vectors are trained jointly with embedding vectors. Although their method addresses the type compatibility issue, it is based on Ontological KG embedding models and shares the same limitations. In another work [74], hierarchical type information available in the dataset is incorporated while learning embeddings. However, their model is suitable only for Ontological KGs where the type information is readily available.

BERT in KG Embedding: BERT architecture has been used for scoring KG triples [76, 69]. However, their methods work on Ontological KGs without any explicit attention to NP types. In other work [52], pre-trained BERT models are used for predicting links in KG. However, their focus was to evaluate knowledge present in the pre-trained BERT models instead of improving the existing link prediction model. BERT embeddings were also used for extracting entity type information [17]. However, it was used for Entity Linking compared to OpenKG link prediction

in our case.

5.3 Background

We first introduce the notation used in this chapter, followed by brief descriptions of BERT and CaRE.

Notation: We follow similar notations as in Chapter 2 with minor differences. An Open Knowledge Graph OpenKG = ($\mathcal{N}$, $\mathcal{R}$, $\mathcal{T}$) contains a set of noun phrases (NPs) $\mathcal{N}$, a set of relation phrases (RPs) $\mathcal{R}$ and a set of triples $(h, r, t) \in \mathcal{T}$ where $h, t \in \mathcal{N}$ and $r \in \mathcal{R}$. Here, h and t are called the head and tail NPs, and r is the RP between them. Each of them contains tokens from a vocabulary $\mathcal{V}$, specifically, $h = (w_1^h, w_2^h, \ldots, w_{k_h}^h)$, $t = (w_1^t, w_2^t, \ldots, w_{k_t}^t)$ and $r = (w_1^r, w_2^r, \ldots, w_{k_r}^r)$. Here, k_h, k_r, and k_t are the numbers of tokens in the head NP, the relation, and the tail NP. OpenKG embedding methods learn vector representations for NPs and RPs. Specifically, vectors for an NP $e \in \mathcal{N}$ and an RP $r \in \mathcal{R}$ are represented by boldface letters $\mathbf{e} \in \mathbb{R}^{d_e}$ and $\mathbf{r} \in \mathbb{R}^{d_r}$. Here, d_e and d_r are dimensions of NP and RP vectors. Usually, $d_e = d_r$. A score function $\psi(h, r, t)$ represents the plausibility of a triple. Similarly, BERT represents tokens by d_B-dimensional vectors. A type projection matrix P takes the vectors to a common d_τ-dimensional type space $\mathbb{R}^{d_\tau}$. The vectors in the type space are denoted by τ.

BERT [20]: BERT is a bi-directional language representation model based on the transformer architecture [68], which has shown performance improvements across multiple NLP tasks. It is pre-trained on two tasks, (1) Masked Language Modeling (MLM), where the model is trained to predict randomly masked tokens from the input sentences, and (2) Next Sentence Prediction (NSP), where the model is trained to predict whether an input pair of sentences occurs in a sequence or not. In our case, we use a pre-trained BERT model (without fine-tuning) for predicting a masked tail NP in a triple.

CaRE [27]: CaRE is an OpenKG embedding method that can incorporate NP canonicalization information while learning the embeddings. NP canonicalization is the problem of grouping all surface forms of a given entity in one cluster, e.g., inferring that *Barack Obama*, *Barack H. Obama*, and *President Obama* all refer to the same underlying entity. CaRE consists of three components: (1) a canonicalization cluster encoder (CN), which generates NP embeddings by aggregating embeddings of canonical NPs from the corresponding cluster, (2) a bi-directional GRU based phrase encoder (PN), which encodes the tokens in RPs to generate RP embeddings, and (3) a base model, which is an Ontological KG embedding method like ConvE [19]. It uses NP and RP embeddings for scoring triples. These triple scores are then fed to a loss function (e.g., pairwise ranking loss with negative sampling [8] or binary cross-entropy loss (BCE) [19]). In this work, we use CaRE with ConvE as the base model. This model generates a candidate

tail NP vector for a given NP h and RP r, denoted by CaRE($\mathbf{h}$, $\mathbf{r}$).

5.4 OKGIT: Our Proposed Method

Motivation: As illustrated in Table 5.1, top NPs predicted by CaRE may not always be type compatible with the input query. On the other hand, BERT's top predictions are usually type compatible [17], although they may not be factually correct. Thus, we hypothesize that a combination of these two models can produce correct as well as type compatible predictions. Motivated by this, we develop OKGIT, which combines the best of both of these models. The complete architecture of the proposed model can be found in Figure 5.1. In the following section, we present various components of the proposed model.

5.4.1 ψ_{PRED}: Tail Prediction Score

The correctness of tail prediction in a triple is measured by the triple score function ψ_{PRED}. Given a triple (h, r, t), it uses the corresponding vectors ($\mathbf{h}$, $\mathbf{r}$, $\mathbf{t}$) and assigns high scores to correct triples and low scores to incorrect triples. We follow CaRE [27] for scoring triples, which internally uses ConvE [19] as the base model. For a given triple (h, r, t), the CaRE model first predicts a tail NP vector $\mathbf{t_C}$ as

$$\mathbf{t_C} = \text{CaRE}(\mathbf{h}, \mathbf{r}) \tag{5.1}$$

The predicted tail NP vector $\mathbf{t_C}$ is then matched against the given tail NP vector $\mathbf{t}$ using dot product to generate the triple score ψ_{PRED}.

$$\psi_{\text{PRED}}(\mathbf{t}, \mathbf{t_C}) = \mathbf{t_C}^\top \mathbf{t}. \tag{5.2}$$

The score ψ_{PRED} represents tail prediction correctness, and CaRE model uses only this score.

5.4.2 ψ_{TYPE}: Tail Type Compatibility Score

The type compatibility between a given (head NP, RP) pair and a tail NP is measured by the type compatibility score function ψ_{TYPE}. It assigns a high score when an NP t has suitable types as candidate tail NP for given head NP h and RP r. We employ a Masked Language Model (MLM) for measuring type compatibility, specifically BERT [20]. Following [52], we can generate a candidate tail NP vector using BERT. Specifically, given a triple (h, r, t), we replace the head NP h and RP r with their tokens and tail NP t with a special MASK token. The resulting sentence $(w_1^h, \ldots, w_{k_h}^h, w_1^r, \ldots, w_{k_r}^r, \text{MASK})$ is sent as input to the BERT model [1]. We

[1] A triple is analogous to a sentence, and link prediction is achieved by predicting a masked part of the sentence. Therefore, we use BERT for MLM task and not for next sentence prediction task.

denote the output vector from BERT corresponding to the MASK tail token as $\mathbf{t_B}$.

$$\mathbf{t_B} = \text{BERT}(h, r, \text{MASK}) \tag{5.3}$$

We can predict tail NPs for a given (h, r) by finding the nearest neighbors of $\mathbf{t_B}$ from the BERT vocabulary (Section 5.7.4). These predicted NPs may not be the correct tail NP present in KG; however, they tend to be type compatible with the given (h, r) pair.

Motivated by this, we extract the implicit NP type information from this vector using a type projector $P_B \in \mathbb{R}^{d_\tau \times d_B}$. The output vector from BERT t_B is high-dimensional and can be used as a proxy for NP's type [17]. Therefore, P_B projects the $\mathbf{t_B}$ vector to a lower dimensional space such that only relevant information is retained. We do a similar operation on tail NP embedding $\mathbf{t}$ and use a type projector $P \in \mathbb{R}^{d_\tau \times d_e}$ to extract type information. Both P_B and P are trained jointly with the model. Thus, the type vectors are given by

$$\boldsymbol{\tau_B} = P_B \mathbf{t_B} \quad \text{and} \quad \boldsymbol{\tau} = P\mathbf{t} \tag{5.4}$$

for BERT and CaRE, respectively. Here, both $\boldsymbol{\tau_B}, \boldsymbol{\tau} \in \mathbb{R}^{d_\tau}$. Then, the type compatibility score between these can be measured by negative of Euclidean distance, i.e.,

$$\psi_{\text{TYPE}}(\boldsymbol{\tau}, \boldsymbol{\tau_B}) = -||\boldsymbol{\tau_B} - \boldsymbol{\tau}||_2^2.$$

We also experimented with a dot product version of the type score, $\psi^{\text{Dot}}_{\text{TYPE}}(\boldsymbol{\tau}, \boldsymbol{\tau_B}) = \boldsymbol{\tau_B}^\top \boldsymbol{\tau}$, and found its performance to be comparable to the Euclidean distance version. Therefore, we use the Euclidean distance version for all our experiments.

5.4.3 ψ_{OKGIT}: Final Composite Score

The score functions ψ_{PRED} and ψ_{TYPE} may contain complementary information. Therefore, we use a combination of triple and type compatibility scores as final score for a given triple.

$$\psi_{\text{OKGIT}}(h, r, t) = \psi_{\text{PRED}}(\mathbf{t}, \mathbf{t_C}) + \gamma \times \psi_{\text{TYPE}}(\boldsymbol{\tau}, \boldsymbol{\tau_B}). \tag{5.5}$$

Please recall that $\mathbf{t_C}$ and $\boldsymbol{\tau_B}$ are in turn dependent on h and r ((5.1) and (5.3)), while $\boldsymbol{\tau}$ is dependent on t (5.4). Here, γ controls the relative weights given to individual scores. This final score takes care of both, i.e., triple correctness as well as type compatibility. For training, we feed the sigmoid of this score function to the Binary Cross Entropy (BCE) loss function following [19].

Dataset	#NPs	#RPs	#Gold Clusters	#Average NPs Per Cluster	#Train	#Validation	#Test
ReVerb20K	11,064	11,057	10,897	1.02	15,498	1,549	2,324
ReVerb45K	27,007	21,622	18,626	1.45	35,969	3,597	5,394
ReVerb20KF	3,524	6,076	3,406	1.03	6,685	1,015	1,517
ReVerb45KF	9,400	11,249	6,749	1.39	14,775	1,781	2,650

Table 5.2: Dataset Statistics. Please refer to Section 5.5 for more details.

5.4.4 Learning with Type Regularization

Let $\mathcal{X} = \{(h_i, r_i) | (h_i, r_i, t_i) \in \mathcal{T} \text{for some } t_i \in \mathcal{N}\}$ be the set of all head NPs and RPs which appear in the OpenKG. Let y_i be the label for the triple (h_i, r_i, t_i) which is 1 if $(h_i, r_i, t_i) \in \mathcal{T}$ and 0 otherwise. We apply the logistic sigmoid function σ on score ψ_{OKGIT} to get the predicted label

$$\hat{y}_i = \sigma(\psi_{\text{OKGIT}}(h_i, r_i, t_i))$$

Finally, we use the following binary cross-entropy (BCE) loss for triple correctness.

$$\text{TripleLoss}(h_i, r_i, t_i) = y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)$$

To further reinforce the type compatibility in the model, we include an additional loss term which forces the type vectors of correct triples to be closer in the type space. Similar to TripleLoss, we use the binary cross-entropy loss for type regularization as well. The type regularization term is shown below.

$$\text{TypeLoss}(h_i, r_i, t_i) = y_i \cdot \log(\hat{p}_i) + (1 - y_i) \cdot \log(1 - \hat{p}_i)$$

where $\hat{p} = \sigma(\psi_{\text{TYPE}}(\boldsymbol{\tau}, \boldsymbol{\tau_B}))$. The cumulative loss function is then given as below.

$$\sum_{i=1}^{n} \text{TripleLoss}(h_i, r_i, t_i) + \lambda \times \text{TypeLoss}(h_i, r_i, t_i) \tag{5.6}$$

where n is the number of training instances. We consider $\mathcal{X} \times \mathcal{N}$ as our training data where triples present in $\mathcal{T}$ have label 1 and rest have label 0.

Dataset	$d_e = d_r$	d_τ	λ	γ	BERT model
ReVerb20K	300	300	0.01	5.0	large
ReVerb45K	300	100	0.0	2.0	large
ReVerb20KF	300	300	0.001	5.0	base
ReVerb45KF	300	300	0.001	0.25	base

Table 5.3: Optimal Hyperparameter values. Please refer to Section 5.5 for more details.

5.5 Experiments

Datasets: Following [27], we use two subsets of English OpenKGs created using ReVerb [24], namely ReVerb20K and ReVerb45K. The full ReVerb Open KB [24] was created by running ReVerb OpenIE system on the Clueweb09 corpus[1]. Two datasets, i.e., Base and Ambiguous, were created from the full dataset. Firstly, 150 Freebase entities that appear with at least two names were sampled. The Base dataset was created by collecting triples mentioning these entities. The Ambiguous dataset was created by enriching the Base dataset with all mentions of homonym entities. These two datasets were combined to form the ReVerb20K dataset. The ReVerb45K is an extended version of Ambiguous dataset proposed in Vashishth et al. [66]. It was created by intersecting information from ReVerb Open KB [24], Freebase entity linking information [26], and Clueweb09 corpus [11]. We use the ReVerb20K and ReVerb45K datasets in our work and follow the same train-validation-test splits as used in Gupta et al. [27]. As noted in [52], predicting multi-token NPs using BERT could be challenging and it might require special pre-training [34]. To understand this difference, we create filtered subsets of these datasets such that they contain only single token NPs [2]. Specifically, we create ReVerb20KF (ReVerb20K-Filtered) and ReVerb45KF (ReVerb45K-Filtered) which contain only single token NPs. More details about these datasets can be found in Table 5.2.

Setup and hyperparameters: We use $d_e = d_r = 300$ for NP and RP vectors. For other hyper-parameters, we use grid-search and select the model based on MRR on validation split. For type vectors, we select d_τ from $\{100, 300, 500\}$. The weight for type regularization term λ is selected from the range $\{10^{-3}, 10^{-2} \ldots, 10^{1}\} \cup \{0\}$. Type composition weight γ is selected from $\{0.25, 0.5, 1.0, 2.0, 5.0\}$. For the language model, we try both BERT-base as well as BERT-large. The optimal values for hyperparameters are shown in Table 5.3. The experiments run for 1.5 hours (for filtered subsets) and 9 hours (for full datasets) on GeForce GTX 1080 Ti GPU.

[1] http://www.lemurproject.org/clueweb09.php/

[2] Please note that the single-token limitation is only valid for BERT, not for OKGIT (Section 5.7.4).

5.6 Results

We evaluate the proposed model on the link prediction task. We follow the same evaluation process as in [27]. From our experiments, we try to answer the following questions:

1. Is OKGIT effective in the link prediction task? (Section 5.6.1)
2. Does OKGIT generate more type compatible NPs in link prediction? (Section 5.6.2)
3. Is the Type Projector effective in extracting type vectors from embeddings? (Section 5.6.3)

5.6.1 Effectiveness of OKGIT Embeddings in Link Prediction

We evaluate our model on the link prediction task. Given a held-out triple (h_i, r_i, t_i), all the NPs $e \in \mathcal{N}$ in the KG are ranked as candidate tail NP based on their score $\psi_{\text{OKGIT}}(h_i, r_i, e)$. Let the rank of the correct tail NP t be denoted by $rank_i^t$. Similarly, ranks are also calculated for predicting head NPs instead of tail NPs using inverse relations [19, 27]; let it be denoted by $rank_i^h$. These ranks are then used to find Mean Reciprocal Rank (MRR), Mean Rank (MR) and Hit@k (k=1,3,10) as follows.

$$\text{MRR} = \frac{1}{2 \times n_{test}} \sum_{i=1}^{n_{test}} \left(\frac{1}{rank_i^h} + \frac{1}{rank_i^t} \right),$$

$$\text{MR} = \frac{1}{2 \times n_{test}} \sum_{i=1}^{n_{test}} \left(rank_i^h + rank_i^t \right), \text{and}$$

$$\text{Hits@k} = \sum_{i=1}^{n_{test}} \frac{\mathbb{1}(rank_i^h \leq k) + \mathbb{1}(rank_i^t \leq k)}{2 \times n_{test}}.$$

Here, n_{test} is the number of test triples and $\mathbb{1}$ is the indicator function. As noted in [27], ranking individual NPs is not suitable for OpenKGs due to the lack of canonicalization. Hence, following their approach, we rank gold canonicalization clusters instead of individual NPs. The gold canonicalization partitions the NPs into clusters such that NPs mentioning the same entity belong to the same cluster. For ranking these clusters, we first find ranks of all NPs $e \in \mathcal{N}$. Then for each cluster, we keep the NP with minimum rank as representative and discard others. The representative NPs are then ranked again and the new ranks are assigned to the corresponding clusters. The rank of the cluster containing the true NP is then used for evaluating the performance. Thus, for a given example, the inference requires one pass of the

full model and $\mathcal{O}(|\mathcal{N}| \log |\mathcal{N}|)$ time for ranking [1]. For better readability, the MRR and Hits@k metrics have been multiplied by 100.

We compare OKGIT with BERT (MLM), ConvE (Ontological KGE) and CaRE (OpenKGE). We also compare against a version of CaRE where phrase embeddings have been initialized with BERT (CaRE [BERT initialization]). As we can see from the results in Table 5.4, the proposed model OKGIT outperforms baseline methods in link prediction task across all datasets. This suggests that the implicit type scores from BERT help in improving ranks of correct NPs. Moreover, OKGIT outperforms CaRE with BERT initialization, suggesting the importance of type projectors [2].

Model	ReVerb20K					ReVerb45K				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
ConvE [19]	26.2	2177.0	20.2	29.1	36.3	18.4	6625.0	13.3	20.6	28.3
CaRE [27]	30.6	851.1	24.4	33.1	41.7	32.0	1276.8	25.3	35.0	44.6
CaRE [BERT initialization]	31.6	837.0	24.8	35.0	44.2	31.2	925.5	24.2	34.4	44.3
OKGIT [Our model]	**35.9**	**527.1**	**28.2**	**39.4**	**49.9**	**33.2**	**773.9**	**26.1**	**36.3**	**46.4**
Model	**ReVerb20KF**					**ReVerb45KF**				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
BERT [20]	4.9	1116.5	2.2	5.0	9.7	18.9	536.5	12.3	20.8	32.5
ConvE [19]	22.3	836.6	16.1	25.5	33.4	16.5	2398.1	10.9	18.9	27.6
CaRE [27]	29.3	308.3	22.1	31.6	43.2	26.6	692.7	20.1	28.8	39.1
CaRE [BERT initialization]	31.8	**207.6**	24.2	34.8	46.2	24.9	557.3	17.8	27.6	38.3
OKGIT [Our model]	**34.6**	214.7	**26.5**	**38.0**	**50.2**	**29.7**	**500.2**	**22.5**	**32.4**	**43.3**

Table 5.4: Results of link prediction task. Here ↑ indicates higher values are better while ↓ indicates lower values are better. We can see that the OKGIT model outperforms the baseline models on all the datasets (Section 5.6.1).

The performance gain is higher for ReVerb20K and ReVerb20KF (+5.3 MRR) than ReVerb45K and ReVerb45KF (+1.2 and +3.1 MRR) datasets. As we can see from Table 5.2, the number of NPs are very close to the number of gold clusters in the 20K datasets. Thus, the canonicalization information is slightly weaker in the 20K datasets than the 45K datasets. Due to this, CaRE achieved better gains in the ReVerb45K dataset as noted in [27]. This leaves more scope of improvements in the 20K datasets. By including the type information from BERT, OKGIT is able to fill this gap. It achieves better gains in the 20K datasets and is able to alleviate the lack of canonicalization information. Moreover, OKGIT is able to improve ranks of correct NPs ranked lower by CaRE. This can be seen by significant improvements in the MR.

[1] Please refer to the Section 5.5 for more details about the training and evaluation runtime.
[2] Please refer to Section 5.7.1 for a detailed comparison.

Other Language Models: Using RoBERTa instead of BERT results in similar performance improvements (Section 5.7.2). However, our primary focus is to understand the impact of *implicit type information* present in pre-trained MLMs, such as BERT, and *not* to compare multiple MLMs themselves.

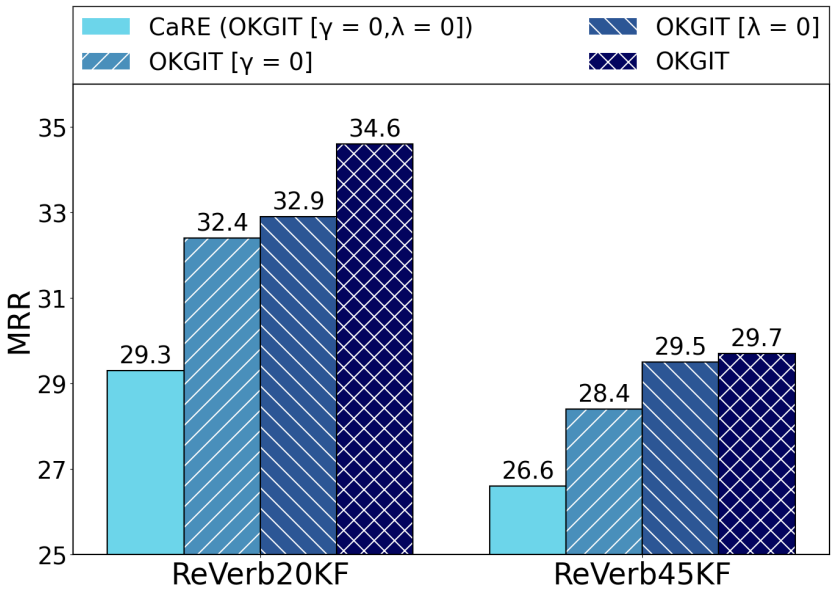


Figure 5.2: Effect of type compatibility score and type regularization on link prediction performance. While the type compatibility score with $\lambda = 0$ gives better gains in MRR (11%-12%) than type regularization term with $\gamma = 0$ (7%-11%), the combined model performs the best, achieving 12%-18% gains in MRR (Section 5.6.1).

Ablations: We perform ablation experiments to compare the relative importance of type compatibility score ψ_{TYPE} and type regularization term. We evaluate OKGIT with disabled type compatibility score (i.e., $\gamma = 0$ in Equation (5.5)) and disabled type regularization term (i.e., $\lambda = 0$ in Equation (5.6)) separately. Please note that CaRE model is equivalent to OKGIT with $\gamma = 0$ and $\lambda = 0$. The results of this experiment are shown in Figure 5.2. We find that while type compatibility score gives more performance gain (11%-12% gain in MRR) than type regularization (7%-11% gain in MRR), the combined model achieves the best performance (12%-18% gain in MRR). It suggests that both the components are important. Please refer to the Sections 5.7.1, 5.7.2, 5.7.3 for more ablation experiments.

Dataset	CaRE	OKGIT
ReVerb20KF	0.23	**0.30**
ReVerb20K	0.35	**0.36**
ReVerb45KF	0.22	**0.31**
ReVerb45K	0.34	**0.35**

Table 5.5: Results of type evaluation in CaRE and OKGIT predictions. We find that OKGIT performs better than CaRE in all datasets in terms of F1-score. Also, the results are statistically significant for all the datasets (Section 5.6.2).

5.6.2 Type Compatibility in Predicted NPs

As noted in [17], BERT vectors contain NP type information[1]. OKGIT utilizes this type information for improving OpenKG link prediction. In this section, we evaluate whether OKGIT improves upon CaRE in predicting type compatible NPs. For such an evaluation, we require type annotations for the NPs in the OpenKGs. However, OpenKGs do not have an underlying ontology or explicit gold NP type annotations, making a direct evaluation impossible. Therefore, we employ a pre-trained entity typing model UFET [18]. Given a sentence and an entity mention, the entity typing model predicts the mentioned entity's types. Using this model, we obtain types for true as well as predicted NPs by CaRE and OKGIT and use it for the evaluation. Please note that this evaluation is limited to the coverage and quality of the UFET model.

Evaluation Protocol: The type vocabulary in UFET model contains $10,331$ types including 9 general, 121 fine-grained, and $10,201$ ultra-fine types. The model takes a sentence $(w_1^h, \ldots, w_{k_h}^h, w_1^r, \ldots, w_{k_r}^r, w_1^t, \ldots, w_{k_t}^t)$ formed from a triple (h, r, t) along with an entity mention (either t or h) as inputs and outputs a distribution over types. We use the top five predicted types for our experiments[2]. For a triple (h, r, t), we consider the types predicted for the true tail NP t as true types $\Gamma(t)$. Let $\hat{t}_{CaRE}$ and $\hat{t}_{OKGIT}$ be the top predicted tail NP by CaRE and OKGIT for the (h, r) pair. Then the types $\Gamma(\hat{t}_{CaRE})$ predicted for $\hat{t}_{CaRE}$ in the triple $(h, r, \hat{t}_{CaRE})$ is used as predicted types for CaRE. Similarly the types $\Gamma(\hat{t}_{OKGIT})$ predicted for $\hat{t}_{OKGIT}$ in the triple $(h, r, \hat{t}_{OKGIT})$ are used as predicted types for OKGIT. For evaluation, we calculate the mean F1-score as follows [3]

$$\text{F1} = \frac{2}{n_{test}} \sum_{i=1}^{n_{test}} \frac{|\Gamma(t_i) \cap \Gamma(\hat{t}_i)|}{|\Gamma(\hat{t}_i)| + |\Gamma(t_i)|}.$$

[1]We also verify this using Freebase, an ontological KG. Please refer to the Section 5.7.6 for more details.

[2]We observe similar behaviour with top one and three types.

[3]Since we use a fixed number of types for ground truth and predictions, precision, recall, and F1-score have the same values. Therefore, we only report the F1-score.

Here, $|\Gamma(t)|$ denotes the number of types present in $\Gamma(t)$ and $\hat{t}$ represents $\hat{t}_{CaRE}$ or $\hat{t}_{OKGIT}$. We can obtain the F1-scores for head NP similarly. We evaluate the mean F1-scores across head and tail NP prediction tasks on the test data and compare CaRE with OKGIT.

As we can see from the results in Table 5.5, OKGIT performs better than CaRE, suggesting that OKGIT generates more type compatible NPs than CaRE in the link prediction task. OKGIT achieves higher gains in the single-token datasets (i.e., ReVerb20KF and ReVerb45KF) than multi-token dataset (i.e., ReVerb20K and ReVerb45K). Upon investigation, we found that the types obtained using the entity typing model (true as well as predicted) for the multi-tokens datasets often contain common noisy types, leading to the small difference between CaRE and OKGIT. Following Dror et al. [22], we also check the results for statistical significance using Permutation, Wilcoxon, and t-test with $\alpha = 0.05$, and found it to be significant for all the datasets.

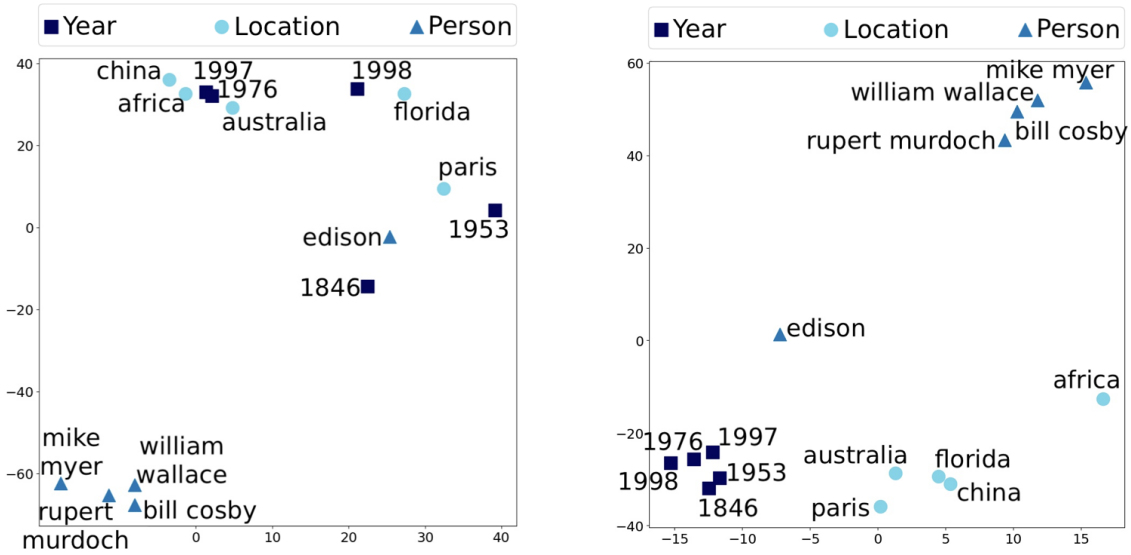


Figure 5.3: t-SNE projections of tail NP embeddings (left) and type vectors (right) extracted by the Type Projector from tail NP embeddings (Section 5.4.2) in the ReVerb20K dataset. We find that the Type Projector is able to extract informative type vectors from the tail embeddings. This is evident from the fact that the tail embeddings corresponding to person, location, and dates were inter-mixed in the left plot, while they have been separated into type specific clusters in the right plot. Please see Section 5.6.3 for details.

5.6.3 Effectiveness of Type Projector

To better understand the effect of type projection, we visualize the vectors in NP-space from CaRE and Type-space (i.e., after type projection) from OKGIT. For this experiment, we

Triple-1	CaRE	OKGIT	Triple-2	CaRE	OKGIT
(*bach*, *moved to*, *?*)	*leipzig* *mobile* *vladimir h. horowitz* *yo yo*	*leipzig* *vienna* *stockholm* *sweden* *turin*	(*clinton*, *lead by*, *?*)	*purchase* *sale* *video* *movie* *discount*	*1500* *260* *80* *99* *hire*

Table 5.6: Few example predictions made by CaRE and OKGIT models. We observe that the OKGIT predictions are more type compatible with the query. Please refer to Section 5.6.4 for more details.

randomly select 5 NPs from 3 categories, namely Person, Location and Year. As OpenKGs do not have type annotations for the NPs, we manually annotated a set of NPs and visualized a random subset. For this process, we first list all the NPs and shuffle them. Then we scan this list and note the first fifteen person names, locations, and years. Later, we select five NPs from each of these categories randomly and use them for the evaluation. We project the NP vectors (i.e., $\mathbf{t}$) corresponding to these NPs to a 2-dimensional NP-space using t-SNE [40][1]. Similarly, we also project the corresponding type vectors (i.e., τ) to 2-dimensional Type-space. We plot the resulting vectors, color and shape coded by their respective categories, in Figure 5.3.

We can see that the vectors from different categories in the NP-space are mixed. However, after the type projection, the vectors in the Type-space are clustered together based on their categories.

5.6.4 Qualitative Evaluations

In this section, we present some examples of predictions made by CaRE and OKGIT methods. The result is shown in Table 5.6. As we see in Triple-1, both CaRE and OKGIT predict the correct NP (i.e., *leipzig*) on top. However, more predictions from OKGIT are type compatible (i.e., all are locations) to the input query. On the other hand, CaRE predictions have mixed types (i.e., location, person, etc.). Also, CaRE makes an incorrect prediction, *vladimir horowitz*, possibly due to the presence of a training triple (*vladimir horowitz*, *had a great affinity for*, *bach*).

We see similar patterns in Triple-2, where the correct tail NP should be of type *number* indicating the count of votes. OKGIT is able to predict numbers in top predictions for Triple-2, while CaRE has mixed types in top predictions.

[1]We run t-SNE for 2000 iterations with 15 perplexity.

5.7 Further Experiments and Results

In this section, we present additional results that demonstrate the importance of various components of OKGIT.

5.7.1 BERT Initialization vs Type Projectors

Here, we demonstrate the importance of type projectors by comparing OKGIT with multiple BERT-augmented versions of CaRE. Specifically, we initialize the phrase and word embeddings in CaRE with a pre-trained BERT model. The phrase (word) is passed as input to BERT and the output corresponding to the $[CLS]$ token is then used for initializing phrase (word) embedding for CaRE model. This modified CaRE model is trained similar to the base CaRE model. Based on different initialization methods, we experiment with following baselines.

Model	ReVerb20K					ReVerb45K				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
CaRE	30.6	851.1	24.4	33.1	41.7	32.0	1276.8	25.3	35.0	44.6
CaRE [BERT-L NP]	31.6	837.0	24.8	35.0	44.2	31.2	925.5	24.2	34.4	44.3
CaRE [BERT NP+PROJ][§]	27.4	950.2	21.9	29.2	38.0	30.7	952.8	23.0	34.4	45.5
CaRE [BERT-L NP+RP]	30.9	862.4	24.6	33.5	42.6	32.8	1015.6	25.9	35.9	45.6
OKGIT [Our model]	**35.9**	**527.1**	**28.2**	**39.4**	**49.9**	**33.2**	**773.9**	**26.1**	**36.3**	**46.4**
Model	**ReVerb20KF**					**ReVerb45KF**				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
CaRE	29.3	308.3	22.1	31.6	43.2	26.6	692.7	20.1	28.8	39.1
CaRE [BERT-B NP]	31.8	207.6	24.2	34.8	46.2	24.9	557.3	17.8	27.6	38.3
CaRE [BERT-L NP+PROJ]	27.6	258.6	21.0	29.1	40.7	24.7	600.5	17.4	27.4	39.2
CaRE [BERT-L NP+RP]	30.1	289.3	22.7	32.8	43.3	26.8	562.5	19.8	29.7	39.8
OKGIT [Our model]	**34.6**	**214.7**	**26.5**	**38.0**	**50.2**	**29.7**	**500.2**	**22.5**	**32.4**	**43.3**

Table 5.7: Results of the link prediction task. Here ↑ indicates higher values are better while ↓ indicates lower values are better. We can see that the OKGIT model outperforms the baseline models on all the datasets (Section 5.7.1). Here, BERT-B and BERT-L denote BERT-base and BERT-large respectively. [§]For NP+PROJ models, BERT-large performs best for ReVerb20K, while BERT-base performs best for ReVerb45K.

CaRE [BERT NP]: NP embeddings are initialized using BERT and the rest of the model is same as CaRE. This model uses 768 (for BERT-base) or 1024 (for BERT-large) dimensional vectors.

CaRE [BERT NP+PROJ]: Since CaRE [BERT NP] uses higher dimensional vectors (768 or 1024) compared to other methods (300), the comparison may not be fair. To address this issue, we project BERT embeddings to 300 dimension. The projection is trained with the rest

of the model.

CaRE [BERT NP+RP]: We initialize NP embedings as well as the word embeddings in RP encoder using BERT embeddings. This method also uses 768 or 1024 dimensional vectors[1].

In all the methods, including OKGIT, we never fine-tune BERT, as our goal is to evaluate the type information already present in pre-trained BERT model. We experiment with both, BERT-base and BERT-large, and report the best performing model.

As we can see from the results in Table 5.7, OKGIT outperforms these baselines. Although BERT initialization improves the performance of CaRE model, the usage of explicit type-score and type regularization leads to significant performance improvements, suggesting their importance.

Model	ReVerb20KF					ReVerb45KF				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
CaRE [27]	29.3	308.3	22.1	31.6	43.2	26.6	692.7	20.1	28.8	39.1
OKGIT-C [$\mathbf{t_B} = [\mathbf{h}; \mathbf{r}]$]	30.0	309.3	22.9	32.4	43.8	27.1	666.5	20.2	29.8	39.9
OKGIT-A [$\mathbf{t_B} = \mathbf{h} + \mathbf{r}$]	30.4	331.7	23.5	32.9	43.6	27.1	660.5	19.9	30.6	40.2
OKGIT-R [RoBERTa]	32.7	221.0	25.3	35.1	46.5	29.0	596.7	21.8	32.0	43.0
OKGIT [Our model]	**34.6**	**214.7**	**26.5**	**38.0**	**50.2**	**29.7**	**500.2**	**22.5**	**32.4**	**43.3**

Table 5.8: Results of the ablation experiments. We replace the BERT module from OKGIT with simple operations such as vector addition (OKGIT-A) and vector concatenation (OKGIT-C). We also use RoBERTa in place of BERT(OKGIT-R). As we can see, replacing BERT with simple operations result in performance similar to CaRE. However, we do see better gains with RoBERTa, which performs better than CaRE and similar to OKGIT for ReVerb45KF. For all datasets, the OKGIT model outperforms other variants (Section 5.7.2).

5.7.2 Replacing BERT with other operations

In this section, we evaluate whether BERT module in OKGIT can be replaced by simple operations such as vector addition and concatenation. Specifically, we modify $\mathbf{t_B}$ in Equation (3) by replacing BERT with these operations leading to the following variants of OKGIT.

OKGIT-C: BERT is replaced by concatenation of head NP vector $\mathbf{h}$ and relation phrase vector $\mathbf{r}$

$$\mathbf{t_B} = [\mathbf{h}; \mathbf{r}].$$

OKGIT-A: BERT is replaced by vector addition

$$\mathbf{t_B} = \mathbf{h} + \mathbf{r}.$$

[1] we also tried using pre-trained BERT as RP encoder in CaRE, however, it performed poorly due to fixed RP encoder.

OKGIT-R: We also experiment with another masked language model RoBERTa in place of BERT.

$$\mathbf{t_B} = \text{RoBERTa}(h, r, \text{MASK}).$$

For this experiment, we use the ReVerb20KF and ReVerb45KF datasets as representatives. We perform grid-search with similar hyper-parameters as in Section 5.5 and select the best model based on the MRR on the validation split. The results are reported in Table 5.8.

Model	ReVerb20KF					ReVerb45KF				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
CaRE	29.3	308.3	22.1	31.6	43.2	26.6	692.7	20.1	28.8	39.1
OKGIT (UFET)	8.8	1208.2	6.9	9.6	11.0	4.9	1156.8	1.5	4.0	11.5
OKGIT [Our model]	**34.6**	**214.7**	**26.5**	**38.0**	**50.2**	**29.7**	**500.2**	**22.5**	**32.4**	**43.3**

Table 5.9: Comparison of OKGIT with OKGIT(UFET). We can see that including UFET model in the OKGIT hurts the performance of the model (Section 5.7.3).

As we can see from the results, the OKGIT-C and OKGIT-A perform very similar to CaRE on both datasets. This suggests that the performance gains for OKGIT come from the BERT module. This observation is further reinforced because OKGIT-R results in similar improvements compared to CaRE as OKGIT. However, in all cases, we find that OKGIT with BERT outperforms other model variants.

5.7.3 CaRE with Entity Typing

Entity typing is the task of predicting explicit types of an entity given a sentence and its mention. As we are interested in improving type compatibility of predictions in the link prediction task, we can also incorporate the output from an entity typing model. In this section, we explore this setting by replacing the BERT module in OKGIT with an entity typing model UFET from [18]. Specifically, we replace the vector t_B in Equation (3) with the output of UFET representing the predicted probability distribution over types.

OKGIT(UFET) Model: The UFET model takes a sentence and an entity mention as input and produces a distribution over explicit set of types. In our case, the sentence is formed by concatenating the subject NP, relation phrase, and object NP, while the object NP is used as mention. The output distribution from UFET is used as t_B in our model. We call this version of the model as OKGIT(UFET) and compare it CaRE and OKGIT.

We run a grid-search for finding the best hyper-parameter similar to Section 5 and report the results on ReVerb20KF and ReVerb45KF datasets. The results are presented in the Table 5.9. As we can see from the results, OKGIT(UFET) performs poorly, even when compared to CaRE.

It suggests that explicit type vectors from UFET model does not help in the link prediction task.

5.7.4 BERT as Link Prediction Model

As mentioned in Section 5.4.2, t_B from Equation (5.3) can be used for predicting tail NPs by finding nearest neighbors in BERT vocabulary. However, this approach has a limitation. This model can only predict NPs that are single token and present in BERT vocabulary, restricting its applicability.

Model	ReVerb20K					ReVerb45K				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
CaRE [27]	30.7	879.2	24.4	33.5	41.7	32.9	1325.1	26.1	36.2	45.4
OKGIT [Our model]	**34.3**	**609.4**	**27.0**	**37.1**	**47.4**	**34.1**	**820.2**	**26.7**	**37.5**	**47.5**
Model	**ReVerb20KF**					**ReVerb45KF**				
	MRR(%)↑	MR↓	Hits(%)↑			MRR(%)↑	MR↓	Hits(%)↑		
			@1	@3	@10			@1	@3	@10
CaRE [27]	28.0	326.0	21.2	30.5	41.3	28.0	683.8	21.0	31.4	41.3
OKGIT [Our model]	**31.7**	**258.1**	**24.2**	**34.0**	**46.3**	**31.2**	**483.3**	**23.8**	**34.4**	**45.4**

Table 5.10: Results of link prediction task on the validation split. We can see that the OKGIT model outperforms the baseline models on all the datasets (Section 5.7.5).

This limitation, however, is not valid for OKGIT. In OKGIT, the vector t_B is used for computing tail type compatibility score, instead of predicting tail NPs. Therefore, it is not restricted to BERT vocabulary or single-token NPs. As shown in Table 5.4, OKGIT is equally effective for single-token datasets (e.g., ReVerb20KF and ReVerb45KF) and multi-token datasets (e.g., ReVerb20K and ReVerb45K).

5.7.5 Link Prediction Performance on Validation Split

The performance of CaRE and OKGIT on validation data on the link prediction task can be found in Table 5.10. These performance corresponds to the respective models which were used to report results in Table 5.4.

5.7.6 Type Information in BERT Predictions

Our proposed OKGIT model is based on the hypothesis that BERT vectors (i.e., $\mathbf{t_B}$ in Equation (3) in Section 4.2) contain implicit type information. In this section, we evaluate this hypothesis that BERT vectors contain type information. It should be noted that evaluating OKGIT model for predicting NP types is not the goal here. We are interested in understanding whether

Model	Precision	Recall	F1
Random	0.13	0.10	0.10
MFT	0.45	0.30	0.31
BERT	0.44	**0.40**	**0.36**
Human	**0.87**	0.18	0.27

Table 5.11: Results of the experiment to test whether BERT embeddings are rich with type information. As we can see, BERT outperforms other methods in terms of F1 score, suggesting that it contains relevant type information. Please refer to Section 5.7.6 for more details.

pre-trained BERT vectors have sufficient type information, measured with respect to some existing anchors.

Evaluation Method: For this experiment, we use Freebase [6] which contains explicit gold type information for entities. Specifically, we use FB15K dataset [8]. We use the data from [76] for converting symbolic names in FB15k to textual descriptions. We only consider the subset of triples in FB15k which has single token in the tail node as BERT can only predict single token NPs.[1] This results in $n_T = 95,782$ triples. For type information, we use the data from [74]. It contains 61 primary types (e.g., */award*). Please note that each node in FB15k can have multiple types. For a triple (h, r, t), we consider the types associated with the true tail NP t as true types $\Gamma(t)$. We then pass tokenized head NP and RP to BERT and find the top prediction $\hat{t} = \text{BERT}(h, r, \text{MASK})$ for tail position. The set of types associated with the predicted NP $\hat{t}$, denoted by $\Gamma(\hat{t})$, is then used as the predicted types. For evaluation, we calculate the following metrics

$$\text{Precision} = \frac{1}{n_T}\sum_{i=1}^{n_T}\frac{|\Gamma(t_i) \cap \Gamma(\hat{t_i})|}{|\Gamma(\hat{t_i})|},$$

$$\text{Recall} = \frac{1}{n_T}\sum_{i=1}^{n_T}\frac{|\Gamma(t_i) \cap \Gamma(\hat{t_i})|}{|\Gamma(t_i)|}, \textit{and}$$

$$\text{F1} = \frac{2}{n_T}\sum_{i=1}^{n_T}\frac{|\Gamma(t_i) \cap \Gamma(\hat{t_i})|}{|\Gamma(\hat{t_i})| + |\Gamma(t_i)|}.$$

Here, $|\Gamma(t)|$ and $|\Gamma(\hat{t})|$ denotes the number of types present in $\Gamma(t)$ and $\Gamma(\hat{t})$ respectively. [2] For comparison, we use the following baseline methods to assign types to a given (h, r, t).

[1] Please note that this limitation is only valid for BERT, not for OKGIT.

[2] Please note that, since we have gold type annotations available for Freebase, the number of true and predicted types need not be the same. Therefore, we evaluate precision and recall along with F1-scores.

Random: assign $|\Gamma(\hat{t})|$ randomly selected types.
Most Frequent Types (MFT): assign $|\Gamma(\hat{t})|$ most frequent types.
Human: We also evaluate the type annotations provided by human annotators on randomly selected 100 triples. Each triple is exposed to three annotators and they are asked to provide types to the tail NP. Since most of the annotations contain one type for each triple, we take the union of the types provided by different annotators to compensate for Recall. For 69% of the triples, the annotators agreed on the same type.

To be fair with the automated baselines, we use the same number of predicted types as BERT (i.e., $|\Gamma(\hat{t})|$). A comparison with a pre-trained explicit entity typing methods, such as [18], is not applicable here as their type vocabulary is different. As we can see from the results in Table 5.11, BERT achieves best F1 score, suggesting that it contains type information. The Recall for Human is low since most of the annotations contained only one type, resulting in lower F1 score.

5.8 Conclusion

The task of link prediction for Open Knowledge Graphs (OpenKG) has been a relatively under-explored research area. Previous work on OpenKG embeddings has primarily focussed on improving or incorporating NP canonicalization information. While there are few methods for OpenKG link prediction, they often predict noun phrases with types incompatible with the query noun and relation phrases. Therefore, we use implicit type information from BERT to improve OpenKG link prediction and propose OKGIT. With the help of novel type compatibility score and type regularization term, OKGIT achieves significant performance improvement on the link prediction task across multiple datasets. We also find that OKGIT produces more type compatible predictions than CaRE, evaluated using an external entity typing model.

Chapter 6

Learning to Interact: An Adaptive Interaction Framework for Knowledge Graph Embeddings

In the last chapter, we addressed the problem of link prediction in Open Knowledge Graphs. In this chapter, we address the same problem, but in the context of Ontological Knowledge Graphs. Specifically, we focus on improving KG Embeddings by learning adaptive interactions. KGE methods represent entities and relations in the KG as vectors in a vector space, trained to distinguish correct edges from incorrect ones. For this distinction, simple functions of vectors' dimensions, called interactions, are used. These interactions are used to calculate the candidate tail entity vector, which is matched against all entities in the KG. However, for most of the existing methods, these interactions are fixed and manually specified. In this chapter, we propose an automated framework for discovering the interactions while training the KG Embeddings. The proposed method learns relevant interactions along with other parameters during training, allowing it to adapt to different datasets. Many of the existing methods can be seen as special cases of the proposed framework. We demonstrate the effectiveness of the proposed method on the link prediction task by extensive experiments on multiple benchmark datasets.

6.1 Introduction

Knowledge Graph Embedding (KGE) methods have been a popular approach for the link prediction task. Most of these methods learn vectorial representations for entities and relations in the KG. A score function is then used to distinguish correct triples from the incorrect ones. Given a triple of the form (h, r, t) where h, r and t are the head entity, relation, and the tail

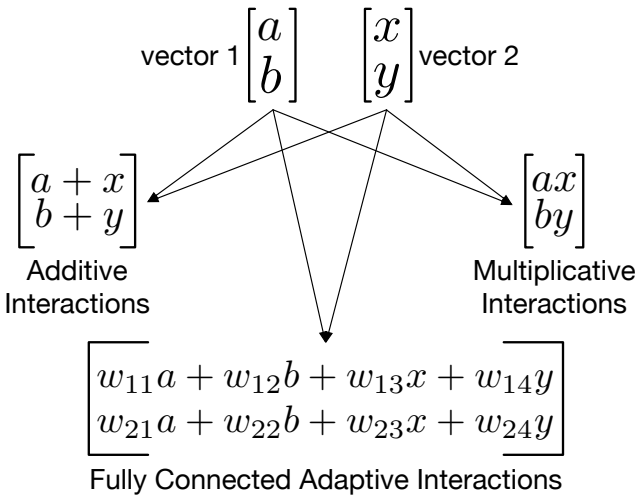


Figure 6.1: Some examples of interactions between two vectors. While additive and multiplicative interactions depend only on the input vectors, the fully connected (FC) interactions have trainable weights, allowing it to adapt to different datasets. In this example, additive interaction $a + x$ can be achived by FC interaction by setting $w_{11} = w_{13} = 1$ and $w_{12} = w_{14} = 0$. Similarly, the multiplicative interaction ax can be achieved by setting $w_{11} = x$ (vector specific weights) and other weights as zero. Please refer to Section 6.3.2 for more details.

entity, a score function assigns a real-valued score to the triple. These score functions depend upon the interactions of dimensions of vectors for h and r. Some examples of interactions are given in Figure 6.1. TransE [8] uses the additive interactions while DistMult [75] uses the multiplicative interactions. However, these interactions are fixed for a given method and are not learnable. This restricts models' capability to weigh entities and relations differently, and hence, from adapting to different datasets. For instance, relations in Freebase like *place_of_birth* give much more information about head and tail entities compared to relations like *_similar_to*, *_hypernym* in WordNet. Thus, learning these interactions while training can enable the model to adapt to different datasets.

We address this issue in this chapter and propose a novel adaptive framework that allows learning these interactions directly from the data during training. The proposed framework is capable of weighing entities and relations differently by using a fully connected interaction layer. It allows the proposed method to adapt to different datasets by learning dataset-specific interactions. By extensive experiments on multiple benchmark datasets, we show the effectiveness of the proposed method on the link prediction task. We also demonstrate that the proposed method assigns different weights to entities and relations by learning dataset-specific interactions.

In summary, we make the following contributions:

- We propose an adaptive interaction framework that can discover relevant interactions of

embeddings from data. We show that many of the existing methods can be seen as special cases of the proposed framework.

- Based on this framework, we propose two new models FCE and FCConvE which outperform the baseline models on link prediction task across commonly used benchmark datasets.
- We also present a method to analyse the fully connected interactions and use it to compare the interactions learned by FCConvE for different datasets.

Notations: We use similar notations as in Chapter 2. A Knowledge Graph is represented by $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$ where $\mathcal{E}$ is the set of entities, $\mathcal{R}$ is the set of relations and $\mathcal{T} \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ is the set of triples stored in the graph. Most of the KG embedding methods learn vectors $\mathbf{e} \in \mathbb{R}^{d_e}$ for $e \in \mathcal{E}$, and $\mathbf{r} \in \mathbb{R}^{d_r}$ for $r \in \mathcal{R}$. Some methods also learn projection matrices $M_r \in \mathbb{R}^{d_r \times d_e}$ for relations. The correctness of a triple is evaluated using a model specific score function $\psi : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \to \mathbb{R}$. For learning the embeddings, a loss function $\mathcal{L}(\mathcal{T}, \mathcal{T}'; \theta)$, defined over a set of positive triples $\mathcal{T}$, a set of (sampled) negative triples $\mathcal{T}'$, and the parameters θ is optimized. I_d denotes the $d \times d$ identity matrix while 0_d denotes the $d \times d$ zero matrix. $diag(v)$ denotes a diagonal matrix created from vector v. All vectors are assumed to be column vectors including the concatenation of vectors $[v_1; v_2; \dots v_k]$. In case of matrix, $[M_1; M_2]$ denotes the block matrix consisting of blocks M_1 and M_2.

6.2 Related Work

The problem of learning KG Embeddings for link prediction has been very popular in the last few years. Based on the score function, these methods can be broadly grouped into three categories, namely Additive, Multiplicative and Neural models.

6.2.1 Additive Models

This is the class of methods where the vectors interact via additive operations after an optional projection operation. One of the simple and popular additive models is TransE [8] where the entity and relation vectors lie in the same vector space. The relation vector acts as a translation from the head entity vector to the tail entity vector. SE [7] is another model that uses relation specific similarity between head and tail entity vectors. Following the ideas of translation and projection, many different methods have been developed. These include TransH [70], TransR [39], STransE [48], ITransF [73], etc. These methods can only extract a restricted set of interactions from the vectors.

Type	Model	Score Function $\psi(h,r,t)$	Interactions
Additive	SE [7]	$-\left\|M_r^1\mathbf{h} - M_r^2\mathbf{t}\right\|_p$	Manual
	TransE [8]	$-\left\|\mathbf{h}+\mathbf{r}-\mathbf{t}\right\|_p$	Manual
	TransR [39]	$-\left\|M_r\mathbf{h}+\mathbf{r}-M_r\mathbf{t}\right\|_p$	Manual
	STransE [48]	$-\left\|M_r^1\mathbf{h}+\mathbf{r}-M_r^2\mathbf{t}\right\|_p$	Manual
Multiplicative	DistMult [75]	$\langle\mathbf{r},(\mathbf{h}\odot\mathbf{t})\rangle$	Manual
	HolE [50]	$\langle\mathbf{r},(\mathbf{h}\star\mathbf{t})\rangle$	Manual
	ComplEx [65]	$\text{Real}(\langle\mathbf{r},(\mathbf{h}\odot\bar{\mathbf{t}})\rangle)$	Manual
	RotatE [62]	$-\left\|\mathbf{h}\odot\mathbf{r}-\mathbf{t}\right\|_p$	Manual
Neural	ER-MLP [21]	$\sigma(\langle\beta,\sigma(A\times[\mathbf{h};\mathbf{r};\mathbf{t}])\rangle)$	Automatic
	ConvE [19]	$\langle\sigma(\text{vec}(\sigma(\rho(\mathbf{v_{hr}})*\Omega))U),\mathbf{t}\rangle$	Manual
Adaptive Interactions Models	FCE (This work)	$\langle\sigma(W\times[\mathbf{h};\mathbf{r}]),\mathbf{t}\rangle$	Automatic
	FCConvE (This work)	$\langle\sigma(\text{vec}(\sigma(\rho(\sigma(W\times[\mathbf{h};\mathbf{r}]))*\Omega))U),\mathbf{t}\rangle$	Automatic

Table 6.1: Summary of various Knowledge Graph (KG) embedding methods mentioned in the chapter. Here $\mathbf{h},\mathbf{r},\mathbf{t}$ denote the head entity, relation and tail entity vectors respectively. M_r, M_r^1, M_r^2 represent the relation specific projection matrices. $\left\|\cdot\right\|_p$ denotes the L1-norm ($p=1$) or L2-norm ($p=2$). $\langle\cdot,\cdot\rangle$, $\odot$, $\star$ and $*$ represent the inner product, the Hadamard product, circular correlation and convolution operations respectively. A and β represent the first and second layer weight matrices in ER-MLP respectively. Ω represents the convolution filters while U represents the final projection matrix in ConvE. $\rho(\mathbf{v_{hr}})$ is a reshaped permutation of $[\mathbf{h};\mathbf{r}]$ with optional repetition. vec denotes vectorization step followed by an activation function σ. Please refer to Section 6.2 for more details.

6.2.2 Multiplicative Models

In this class of methods, the vectors interact via a multiplicative operation. One of the initial models in this category is RESCAL [49], which is based on tensor factorization. It models entities as vectors while relations as matrices. DistMult [75] is a special case of RESCAL where the relations matrices are restricted to be diagonal. However, DistMult score function is symmetric and hence it can not handle asymmetric relations. To alleviate this issue, HolE [50] was proposed which uses circular correlation operation between head and tail entity vectors. ComplEx [65] addresses the same issue by modelling vectors in the complex domain. The asymmetry of complex dot product allows it to handle symmetric, asymmetric as well as anti-symmetric relations with the same score function. SimplE [36] is a more recent model based on tensor factorization which can express all types of relations. RotatE [62] is another model which uses complex vectors for representation. It models relation vectors as rotations from head to tail entity vector and then uses L1-norm based distance as score function. Similar to additive models, multiplicative models are also restricted in terms of the interactions they can extract.

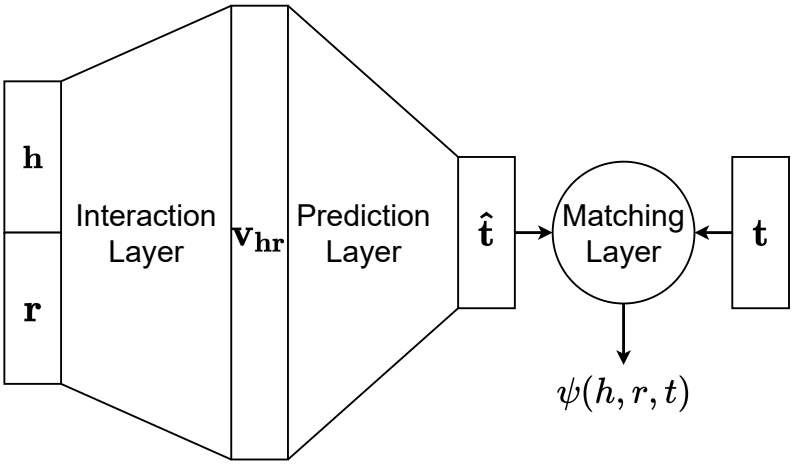
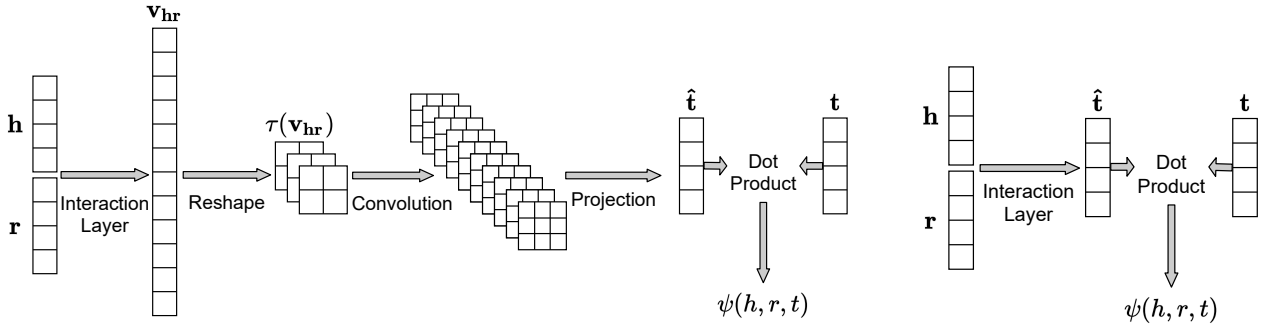


Figure 6.2: The block diagram of the Adaptive Interaction Framework. The interaction layer extracts interactions $\mathbf{v_{hr}}$ from head entity $\mathbf{h}$ and relation vector $\mathbf{r}$. The prediction layer calculates a candidate tail entity vector $\hat{\mathbf{t}}$ which is then matched with existing tail entity vector $\mathbf{t}$ using the matching layer to produce a real valued score.

6.2.3 Neural Models

There are many models that use various neural network architectures for learning KG Embeddings. Some of these models are NTN [59], ER-MLP [21], CONV [64], ProjE [58], R-GCN [56], ConvE [19], ER-MLP-2n [53], KG-BERT [76], InteractE [67], etc. Unlike other methods, KG-BERT uses word embeddings for encoding entities and relations. Therefore, the interactions of entity and relation vectors are not directly clear. Among the rest of the models, ProjE can extract interactions only from entity or relation vectors but not both. ConvE can extract interactions from both, entity as well as relation vectors, but they are extracted using predefined permutations. InteractE exploits more sophisticated interactions using feature permutation, checkered reshaping, and circular convolution resulting in improved performance. However, these methods depend on a fixed set of interactions defined by the model and can not adaptively learn these interactions during training.

As described in the next section, the models proposed in this chapter are neural models that can adaptively learn the interactions from the entity as well well as relation vectors using a fully connected interaction layer. This choice is also motivated by ER-MLP-2n [53] which demonstrated generalization capabilities across datasets.



(a) FCConvE (b) FCE

Figure 6.3: Architecture diagrams for FCConvE (left) and FCE (right). Please refer to Section 6.3.3 and Section 6.3.4 for more details.

6.3 Proposed Method

6.3.1 The Adaptive Interaction Framework

As shown in Figure 6.2, it consists of the following three components

6.3.1.1 Interaction Layer

This layer extracts the interactions between the head entity and relation vectors which are later used for predicting tail entities. The interactions are extracted using a fully-connected (FC) layer:

$$\mathbf{v_{hr}} = \sigma(W_r \times [\mathbf{h}; \mathbf{r}]), \tag{6.1}$$

where σ is an activation function. In the general case, W_r is $\mathbb{R}^{d_i \times (d_e + d_r)}$ where d_i is the dimension of the interaction layer. A special case of interaction layer is when $W_r = W, \forall\ r \in \mathcal{R}$ which avoids over-parameterization. We use this special case for our experimentations.

6.3.1.2 Prediction Layer

This layer predicts a vector $\hat{\mathbf{t}}$ for the candidate tail entity as

$$\hat{\mathbf{t}} = g(\mathbf{v_{hr}}), \tag{6.2}$$

where the function g depends on the method. Many of the methods like TransE and DistMult use identity function. For our experiments, we use ConvE's [19] architecture for this step.

6.3.1.3 Matching Layer

This is the final layer where the predicted tail entity is matched against a given tail entity producing the final score for the triple. The score function is given as

$$\psi(h, r, t) = f(\hat{\mathbf{t}}, \mathbf{t}). \tag{6.3}$$

Again, the matching function f is method dependent. We use the vector dot product for matching.

6.3.2 Existing models as special case

In this section, we demonstrate how the proposed framework generalizes many of the existing models. The score functions for these models can be found in Table 6.1.

TransE: In TransE, the entities and relations lie in the same $d_e = d_r = d$ dimensional space $\mathbb{R}^d$. The interaction matrix is $W_r = [I_d; I_d]$ with identity activation while the tail prediction function $\hat{\mathbf{t}} = g(\mathbf{v_{hr}}) = \mathbf{v_{hr}}$ is the identity function. The matching function takes the form of a vector norm $-\left\|\hat{\mathbf{t}} - \mathbf{t}\right\|_p$.

STransE: STransE [48] generalizes many translation-based models like TransH [70] and TransR [39] by using two relation specific projection matrices, M_r^1 for head and M_r^2 for tail entities. The interaction matrix is $W_r = [M_r^1; I_{d_r}]$ while the activation and prediction functions are identity functions. The matching function takes the form of $-\left\|\hat{\mathbf{t}} - M_r^2\mathbf{t}\right\|_p$.

DistMult: For DistMult [75], the interaction matrix is $W_r = [diag(\mathbf{r}), 0_d]$ while the activation and prediction functions are identity functions. Vector dot product between $\hat{\mathbf{t}}$ and $\mathbf{t}$ is used as matching function.

ConvE: ConvE [19] is a recent model which uses a convolution network for extracting features from the interactions and then predicting the tail entity. Here, the interaction matrix is a fixed and manually specified permutation matrix, used for dimension shuffling in ConvE. Please note that the interaction matrix is shared among all relations. The prediction function is the 2D convolution network applied on the interactions vector $\mathbf{v_{hr}}$ while dot product between $\hat{\mathbf{t}}$ and $\mathbf{t}$ is used for matching.

Based on the proposed framework, we present two new models FCConvE and FCE in the following sections. Their architecture can be found in Figure 6.3.

6.3.3 FCConvE

One issue with the existing methods is that the interactions are either fixed or it requires to be specified manually. For example, in the case of ConvE the permutations are specified by the

user. This leaves the choice of interactions to the user and also does not specify which ones are better than others. Also, the performance on the link prediction task varies with different choices of permutations in ConvE. Hence it will be useful if the optimal interactions can be learned directly from the dataset.

FCConvE addresses this exact issue and allows the model to learn the appropriate interactions while training. It achieves this by using interaction matrix $W_r = W, \forall r \in \mathcal{R}$ as model parameter. Since many of the existing interactions can be achieved using different W matrices, the proposed model is capable of choosing the optimal interactions by itself. The score function for FCConvE can be given as follows:

$$\psi(h, r, t) = \langle \sigma(\text{vec}(\sigma(\rho(\mathbf{v_{hr}}) * \Omega))U), \mathbf{t} \rangle, \tag{6.4}$$

where $\mathbf{v_{hr}}$ is given by (6.1). As compared to ConvE, FCConvE contains approximately $\mathcal{O}(d_i \times (2d_e + d_r))$ more parameters due to the interaction and projection layers weights.

6.3.4 FCE

We experiment with a modification of DistMult named DistMult-BCE which uses Binary Cross Entropy (BCE) loss instead of margin-based ranking loss used in [75]. We also introduce its adaptive interactions variant FCE (Fully Connected Embedding) which uses an FC layer as in (6.1) for interactions instead of Hadamard product between $\mathbf{h}$ and $\mathbf{r}$. The score function for FCE can be given as follows:

$$\psi(h, r, t) = \langle \sigma(W \times [\mathbf{h}; \mathbf{r}]), \mathbf{t} \rangle. \tag{6.5}$$

We use ReLU for the activation function σ. Please note that, similar to DistMult, entity and relation vectors lie in same $d_e = d_r = d$ dimensional space $\mathbb{R}^d$. The interaction vector $\mathbf{v_{hr}}$ also lies in the same space $\mathbb{R}^d$ (i.e., $d_i = d$). Unlike DistMult, the interaction matrix $W \in \mathbb{R}^{d \times d}$ is shared across relations. In terms of model size, FCE contains $d_i \times (d_e + d_r) = 2d^2$ more parameters as compared to DistMult.

6.3.5 Analysing Interactions using NAIW

In this section, we introduce a novel method to analyse fully connected interactions. As mentioned in previous sections, the interaction weight matrix W in FCConvE as well as FCE is shared among all relations i.e., $W_r = W, \forall r \in \mathcal{R}$. This interaction weight matrix can be split into two parts, $W = [W^{\mathcal{E}}; W^{\mathcal{R}}], W^{\mathcal{E}} \in \mathbb{R}^{d_i \times d_e}, W^{\mathcal{R}} \in \mathbb{R}^{d_i \times d_r}$ corresponding to the head entity and the relation vectors respectively. The equation for the interaction layer can be re-written as

	FB15K	FB15K-237	WN18	WN18RR
#Entities	14,941	14,541	40,943	40,943
#Relations	1,345	237	18	11
#Train	483,142	272,115	141,441	86,835
#Validation	50,000	17,535	5,000	3,034
#Test	59,071	20,466	5,000	3,134

Table 6.2: Details of the datasets used in our experiments

follows:

$$\mathbf{v_{hr}} = \sigma(W^{\mathcal{E}}\mathbf{h} + W^{\mathcal{R}}\mathbf{r}). \tag{6.6}$$

The values in $W^{\mathcal{E}}$ and $W^{\mathcal{R}}$ represent the importance of various dimensions of the head entity and relation vectors. We use the absolute values of these weights for comparing the importance of entity and relation. Specifically, we use the *Normalized Absolute Interaction Weights (NAIW)* as defined below for comparing the weights corresponding to entities and relations.

$$\begin{aligned} V^{\mathcal{E}} &= \sum_{j=1}^{d_e} \text{AbsoluteValue}(W^{\mathcal{E}}[:, j]), \\ \text{NAIW}^{\mathcal{E}} &= \frac{V^{\mathcal{E}}}{\max(V^{\mathcal{E}})}, \end{aligned} \tag{6.7}$$

where $V^{\mathcal{E}} \in \mathbb{R}^{d_i}$ is a vector containing sum of absolute interaction weights across dimensions of head entity. It denotes the importance of the head entity for each unit in the interaction layer. We normalize this vector such that the values lie in $[0, 1]$ range. This allows us to compare this value across datasets. Similarly, we can calculate $V^{\mathcal{R}}$ and $\text{NAIW}^{\mathcal{R}}$ which denote the importance of relation for units in the interaction layer.

Each value in the NAIW vector represents the importance of entities (for $\text{NAIW}^{\mathcal{E}}$) or relations (for $\text{NAIW}^{\mathcal{R}}$) for the link prediction task. Thus, comparing these values helps us understand the relative importance of entities and relations for different datasets. Since a comparison of individual interaction units may be inconclusive, we compare their distributions. We estimate the distributions[1] of $\text{NAIW}^{\mathcal{E}}$ and $\text{NAIW}^{\mathcal{R}}$ and compare them across multiple datasets. These distributions allow us to compare the importance of the entity and relation for the link prediction task.

[1]We use *gaussian_kde* function from SciPy library for estimating distributions.

6.4 Experiment Results

In this section, we evaluate the proposed method on the link prediction task and compare it against the baselines. The details of the datasets used for evaluation are given in Table 6.2. We provide the implementation details followed by the results in the following sections.

6.4.1 Implementation details

In our experiments, we use 200-dimension embeddings for both entity as well as relations. For selecting other hyper-parameters, we use cross-validation using the MRR on validation split of the data. Similar to ConvE, we use dropouts and batch normalization at input, convolution and final projection layer. The corresponding dropout probabilities are selected from [0.1, 0.2, 0.3], [0.2, 0.3, 0.4] and [0.4, 0.5, 0.6] respectively. For the interaction layer, we use 5000 dimensions reshaped to $25 \times 10 \times 20$ before applying convolution. Unlike ConvE, FCConvE uses depthwise group convolution. The filter size for convolution is selected from $[3 \times 3, 5 \times 5]$. For optimization, we use Adam optimizer with an initial learning rate of 0.001. We use early stopping using MRR on validation split with the maximum number of epochs set to 100. The best model is then run for 1000 epochs[1] and final performance is reported. We use 500 negative samples per correct triple. For head entity prediction, we follow ConvE and use reversed relations during training as well as evaluation. The embeddings are trained using binary cross-entropy loss. The baseline method ConvE uses 4M parameters for FB15K and FB15K-237, and 9M for WN18 and WN18RR datasets. The proposed approach FCE uses 3M parameters for FB15K and FB15K-237, and 8M for WN18 and WN18RR datasets. FCConvE uses 28M parameters for FB15K, 20M for FB15K-237, 26M for WN18, and 33M for WN18RR. Thus, FCConvE uses much higher number of parameters as compared to the other two methods. For a given triple, it requires one forward pass of the model followed by $\mathcal{O}(|\mathcal{E}| \log |\mathcal{E}|)$ time for ranking entities. The training takes 2-3 minutes per epoch while the evaluation takes 9-15 seconds on GeForce GTX 1080 Ti GPU.

6.4.2 Link Prediction

Given a test or validation triple, we score the head entity and relation against all entities and report the rank of the correct tail entity. A similar strategy is used for head entity prediction except that we use reverse relation vectors. The model's performance is evaluated on both head as well as tail entity prediction and the average performance is reported. Similar to previous work, we use filtered setting, i.e., we exclude all triples appearing in train, test or validation

[1] We pick the model from the epoch with the best validation split MRR, and it need not be the final epoch.

Model	FB15K-237					WN18RR				
	MR↓	MRR↑	Hits↑			MR↓	MRR↑	Hits↑		
			@1	@3	@10			@1	@3	@10
DistMult [75]	254	24.1	15.5	26.3	41.9	5110	43.0	39.0	44.0	49.0
ComplEx [65]	339	24.7	15.8	27.5	42.8	5261	44.0	41.0	46.0	51.0
R-GCN [56]	-	24.8	15.3	25.8	41.7	-	-	-	-	-
ConvE [19]	**244**	32.5	23.7	35.6	50.1	4187	43.0	40.0	44.0	52.0
DistMult-BCE	318	30.1	21.6	33.0	46.9	7037	41.0	38.6	41.6	46.0
FCE	331	30.6	21.7	33.7	48.6	4732	41.3	38.2	42.2	47.5
FCConvE	255	**35.5**	**26.4**	**39.1**	**54.0**	**4103**	**46.1**	**42.8**	**47.7**	**52.7**

Model	FB15K					WN18				
	MR↓	MRR↑	Hits↑			MR↓	MRR↑	Hits↑		
			@1	@3	@10			@1	@3	@10
TransE [8]	-	46.3	29.7	57.8	74.9	-	49.5	11.3	88.8	94.3
DistMult [75]	97	65.4	54.6	73.3	82.4	902	82.2	72.8	91.4	93.6
ComplEx [65]	-	69.2	59.9	75.9	84.0	-	94.1	93.6	93.6	94.7
R-GCN [56]	-	69.6	60.1	76.0	84.2	-	81.4	69.7	92.9	**96.4**
ConvE [19]	**51**	65.7	55.8	72.3	83.1	**374**	94.3	93.5	94.6	95.6
DistMult-BCE	115	73.3	66.8	77.8	84.9	671	83.9	74.8	92.6	94.7
FCE	108	**74.6**	**67.8**	**79.5**	**86.1**	516	94.2	93.6	94.5	95.2
FCConvE	67	71.7	63.4	77.3	85.6	440	**94.8**	**94.3**	**95.1**	95.5

Table 6.3: Link prediction results on benchmark datasets. Here ↑ indicates higher values are better while ↓ indicates lower values are better. The adaptive interaction versions of the models FCConvE and FCE outperform the corresponding baseline models ConvE and DistMult-BCE in all the datasets. They also outperform other methods across all datasets. Results for the baseline models except TransE were taken from [19]. For TransE, we have taken the results from [50]. We have also included results for a modification of DistMult called DistMult-BCE. Please refer to Section 6.4.2 for more details.

split while ranking. We report Mean Reciprocal Rank (MRR), Mean Rank (MR) and Hits@k for k=1, 3, 10 on test split. The results can be found in Table 6.3.

For comparison, we use a few representative baselines from each category of the models. Specifically, we use TransE for additive models, DistMult and ComplEx for multiplicative models, and R-GCN and ConvE for neural models. Please note that our goal is to compare a model with its adaptive interaction version, instead of comparing all available models.

We observe from the results that the proposed adaptive interaction versions of the models outperform the corresponding baseline models in all the datasets. FCConvE significantly outperforms ConvE in all the datasets except WN18 where the performance is comparable. Similarly, FCE significantly outperforms DistMult-BCE in WN18 while showing marginal

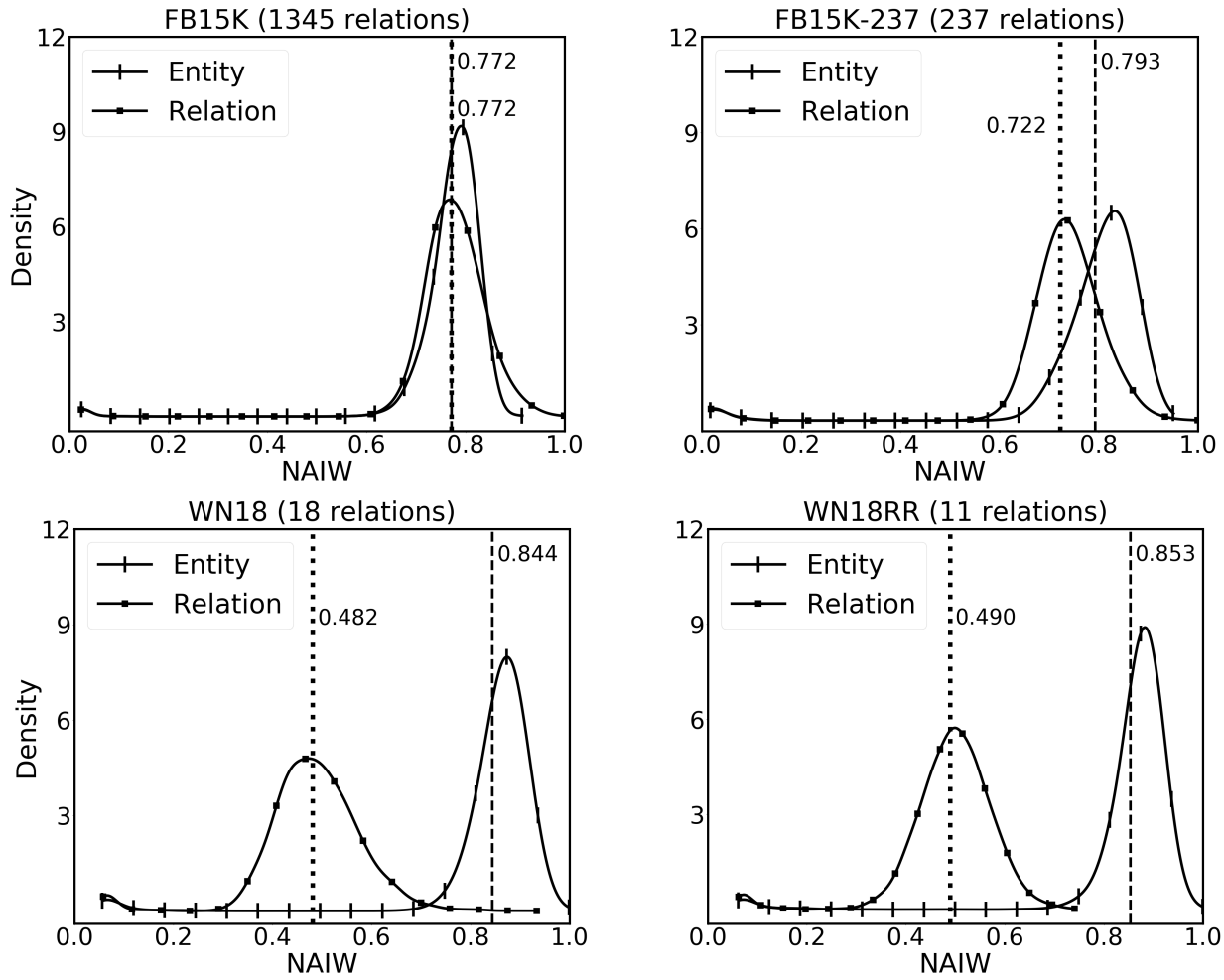


Figure 6.4: Distributions of the Normalized Absolute Interaction Weights for entities and relations learned by FCConvE on different datasets. The means of these distributions are shown as a dashed (for Entity) or dotted (for Relation) vertical lines along with their values. The datasets are arranged according to decreasing order of number of relations from left to right. As we can see, for lower number of relations, the difference in weights for entities and relations are much higher. Please refer to Section 6.4.3 for more details.

improvements is other datasets. Also, they outperform other methods on the link prediction task across all the datasets suggesting that the proposed approach is able to adapt to different datasets resulting in performance improvements.

It should be noted that ConvE would struggle to differentially weigh entity and relation vector dimensions due to the sharing of convolution filters. Adding an FC interaction layer allows it to prioritize between entity and relation vectors resulting in better performance of FCConvE. We also observe that the DistMult-BCE model significantly outperforms the DistMult model in FB15K and FB15K-237 which suggests the BCE loss with multiple negative samples

improves performance.

We find that a simple method FCE performs much better on FB15K and WN18 datasets compared to the other two datasets. As noted in the past works [64, 19], FB15K and WN18 datasets suffer from test leakage due to presence of inverse relations in the evaluation splits. FCE exploits this test leakage leading to comparable or better performance on these datasets. Also, FCE outperforms FCConvE on FB15K dataset while using much less number of parameters. FCConvE is possibly too complex and over-parameterized for FB15K, leading to poor generalization performance. On all other datasets, model with the most number of parameters (i.e., FCConvE) performs the best (avoiding over-fitting).

Among the metrics used in Table 6.3, MR is more sensitive to outliers (i.e., large values of ranks) than others. We observe that the proposed approaches achieve improvements in MRR and Hits@k, but not MR for many datasets. It suggests that the proposed methods are effective in bringing more cases into the high-rank region, which could be a desirable property in many applications.

6.4.3 Interactions Analysis

In this section, we analyse the interactions learned by FCConvE on different datasets. We use the distributions of $\text{NAIW}^{\mathcal{E}}$ and $\text{NAIW}^{\mathcal{R}}$, as defined in Section 6.3.5 and compare them. The results are shown in Figure 6.4.

As we can see from Figure 6.4, the distributions of $\text{NAIW}^{\mathcal{E}}$ and $\text{NAIW}^{\mathcal{R}}$ varies across different datasets. Furthermore, we make the following observations.

- The difference between the means of $\text{NAIW}^{\mathcal{E}}$ and $\text{NAIW}^{\mathcal{R}}$ increases with decreasing number of relations. Among the datasets used, FB15K has the most number of relations (i.e., 1,345) while WN18RR has the least number of relations (i.e., 11). This number is correlated with the difference of the means of $\text{NAIW}^{\mathcal{E}}$ and $\text{NAIW}^{\mathcal{R}}$ with WN18RR having the highest difference, while FB15K having the lowest difference. This suggests that when a dataset has a small number of relations, entities have more distinguishing capability than the relations.

- The relations in Freebase datasets (i.e., FB15K and FB15K-237) have more distinguishing capability than relations in WordNet datasets (i.e., WN18 and WN18RR). For example, relations like *place_of_birth* in Freebase restricts candidate entity types for head and tail entities. On the other hand, relations in Wordnet (e.g., *_similar_to*, *_hypernym*) are not very specific to some type of entities. This behavior is reflected in the distributions of $\text{NAIW}^{\mathcal{E}}$ and $\text{NAIW}^{\mathcal{R}}$ with relations getting more weights in Freebase datasets as compared

Permutation	FB15K	FB15K-237	WN18	WN18RR
Plain	63.2	32.7	94.3	43.2
Alternate Rows	63.6	33.3	94.8	44.3
Alternate	63.9	33.3	94.8	44.4
FCConvE	71.7	35.5	94.8	46.1

Table 6.4: The effect of various permutation schemes on the performance of ConvE. We report the MRR in link prediction task across various datasets. As we can see, the performance of ConvE is dependent on the choice of permutation scheme and using Alternate or Alternate Rows permutation improves the performance of ConvE. Please refer to Section 6.4.4 for more details.

to WordNet datasets.

6.4.4 Effect of various interactions on ConvE

To further understand the advantages of adaptive interactions, we compare its performance with various fixed interactions (as used in [67]). As mentioned in Section 6.3.2, ConvE can use a permutation matrix for generating interactions. For demonstration, let's assume a head entity vector $\mathbf{h} = [\mathbf{h}^1, \mathbf{h}^2, \mathbf{h}^3, \mathbf{h}^4, \mathbf{h}^5, \mathbf{h}^6]$ and a relation vector $\mathbf{r} = [\mathbf{r}^1, \mathbf{r}^2, \mathbf{r}^3, \mathbf{r}^4, \mathbf{r}^5, \mathbf{r}^6]$. These vectors are concatenated, permuted and then reshaped into a 4×3 matrix before passing it to the convolution layer. As shown in Figure 6.5, the following are some of the candidate permutations.
Plain: A plain concatenation and reshaping of the vectors.
Alternate Rows: Alternate rows of head entity and relation vectors dimensions.
Alternate: Strictly alternating dimensions of head entity and relation vectors.

$$\begin{bmatrix} \mathbf{h}^1 & \mathbf{h}^2 & \mathbf{h}^3 \\ \mathbf{h}^4 & \mathbf{h}^5 & \mathbf{h}^6 \\ \mathbf{r}^1 & \mathbf{r}^2 & \mathbf{r}^3 \\ \mathbf{r}^4 & \mathbf{r}^5 & \mathbf{r}^6 \end{bmatrix}$$

(a) Plain

$$\begin{bmatrix} \mathbf{h}^1 & \mathbf{h}^2 & \mathbf{h}^3 \\ \mathbf{r}^1 & \mathbf{r}^2 & \mathbf{r}^3 \\ \mathbf{h}^4 & \mathbf{h}^5 & \mathbf{h}^6 \\ \mathbf{r}^4 & \mathbf{r}^5 & \mathbf{r}^6 \end{bmatrix}$$

(b) Alternate Rows

$$\begin{bmatrix} \mathbf{h}^1 & \mathbf{r}^1 & \mathbf{h}^2 \\ \mathbf{r}^2 & \mathbf{h}^3 & \mathbf{r}^3 \\ \mathbf{h}^4 & \mathbf{r}^4 & \mathbf{h}^5 \\ \mathbf{r}^5 & \mathbf{h}^6 & \mathbf{r}^6 \end{bmatrix}$$

(c) Alternate

Figure 6.5: Some example permutations of two 6-dimensional vectors $\mathbf{h}$ and $\mathbf{r}$.

As we can see, *Plain* method allows entity-relation interactions only at the boundary region while *Alternate Rows* and *Alternate* allow deeper interactions.

We run the best hyper-parameters settings of ConvE with these three permutations on all the datasets and compare the MRR of the link prediction task. As seen from the results in Table 6.4, the *Alternate Rows* and *Alternate* permutation schemes achieve better results

compared to *Plain* permutation scheme. However, since FCConvE can learn the interactions while training, it achieves better or comparable MRR on all the datasets.

6.5 Conclusion

We presented an adaptive interaction framework for learning KG Embeddings and proposed two new models based on the framework. We demonstrated that the proposed models are capable of learning relevant interactions across different datasets. We also demonstrated how some of the existing KG Embedding models can be seen as special cases of the proposed framework.

Chapter 7

Conclusion and Future Work

This thesis presented an extensive analysis of Knowledge Graph Embeddings and addressed some limitations of existing methods. In the first part, we presented a macro and a micro-analysis of KG Embeddings. We started with analyzing the geometrical behavior of KG Embeddings. We presented a set of simple metrics for the geometrical analysis of a group of vectors. Using these metrics, we discovered significant differences between additive and multiplicative models. We also correlated the geometry with various hyper-parameters and performance in the link prediction task. To the best of our knowledge, this is the first study of its kind for KG Embeddings. We continued the analysis, focusing on understanding the semantics of KG Embeddings' dimensions. We presented a method for inducing interpretability in KG Embeddings using a coherence regularizer. The proposed method exhibits significant improvements in interpretability while maintaining comparable performance in link prediction and triple classification tasks. Further, a qualitative evaluation revealed that the top entities along embeddings' dimensions are more coherent for the proposed method than the baseline.

In the second part of the thesis, we focused on a special type of KGs called Open Knowledge Graphs. OpenKG link prediction has been a relatively under-explored area of research. While there are a few methods, we found that they struggle to maintain type compatibility in link prediction. We addressed this problem and proposed OKGIT for improving type compatibility in OpenKG link prediction. Using the novel type compatibility score and type regularization term, OKGIT achieved significant performance improvement in the link prediction task. With further evaluation, we demonstrated that OKGIT produces more type compatible predictions that CaRE.

Lastly, in the third part of the thesis, we focused on improving KG Embeddings by learning adaptive interactions. We presented an adaptive interaction framework for learning KG Embeddings. Based on this framework, we also proposed two new models which performed better than

the corresponding baselines. With experiments on multiple datasets, we demonstrated that the proposed method is capable of learning relevant interactions and can adapt to the datasets. We also demonstrated how some of the existing models can be seen as special cases of the proposed framework.

Future Works: There are multiple future directions one can explore based on this thesis. The analysis in Chapter 3 can be used for characterizing models and model selection. For example, Conicity was used for mimicking a complex model's behavior in a simple model for improving diversity in a conversation model [1]. Similarly, one can explore the relationship between diversity and Conicity in other tasks. Understanding the mapping between embeddings' dimensions and entity categories could be another possible direction of work. While we have presented a method to induce implicit semantic categories in entity embeddings, mapping them with explicit categories is still an unexplored problem. For OpenKGs, one can compare the impact of other pre-trained masked language models (e.g., ALBERT [37]) as well as autoregressive language models (e.g., GPT-3 [10]). Understanding the impact of multi-token masking with ERNIE [78] and SpanBERT [34] in OpenKG link prediction can be another future direction of exploration. Finally, one can explore the effect of more complex interactions in KG Embeddings using Kernel methods and its correlation with dataset properties.

Bibliography

[1] Siddhartha Arora, Mitesh M. Khapra, and Harish G. Ramaswamy. On knowledge distillation from complex networks for response prediction. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3813–3822, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1382. URL https://www.aclweb.org/anthology/N19-1382. 79

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *Proceedings of the 6th International The Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, page 722–735, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 3540762973. 1

[3] Ivana Balazevic, Carl Allen, and Timothy Hospedales. TuckER: Tensor factorization for knowledge graph completion. In *EMNLP-IJCNLP*, pages 5184–5193, Hong Kong, China, November 2019. ACL. doi: 10.18653/v1/D19-1522. URL https://www.aclweb.org/anthology/D19-1522. 2, 33, 34

[4] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. Open information extraction from the web. In *IJCAI*, IJCAI'07, page 2670–2676, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. 42

[5] Federico Bianchi, Gaetano Rossiello, Luca Costabello, Matteo Palmonari, and Pasquale Minervini. Knowledge graph embeddings and explainable ai. *arXiv preprint arXiv:2004.14843*, 2020. 35

[6] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of*

the 2008 ACM SIGMOD international conference on Management of data, pages 1247–1250. ACM, 2008. 1, 15, 61

[7] Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011. 65, 66

[8] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems (NeurIPS)*, pages 1–9, 2013. 1, 3, 7, 9, 11, 12, 15, 21, 33, 34, 46, 61, 64, 65, 66, 73

[9] Samuel Broscheit, Kiril Gashteovski, Yanjie Wang, and Rainer Gemulla. Can we predict new facts with open knowledge graph embeddings? a benchmark for open link prediction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2296–2308, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.209. URL https://www.aclweb.org/anthology/2020.acl-main.209. 44

[10] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. 2020. 79

[11] Jamie Callan, Mark Hoy, Changkuk Yoo, and Le Zhao. Clueweb09 data set, 2009. 50

[12] Chandrahas and Partha Pratim Talukdar. OKGIT: Open knowledge graph link prediction with implicit types. In *Findings of the ACL: ACL-IJCNLP*. Association for Computational Linguistics, 2021. 4

[13] Chandrahas, Aditya Sharma, and Partha Talukdar. Towards understanding the geometry of knowledge graph embeddings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 122–131. Association for Computational Linguistics, 2018. 4

[14] Chandrahas, Nilesh Agrawal, and Partha Pratim Talukdar. Learning to Interact: An Adaptive Interaction Framework for Knowledge Graph Embeddings. In *International Conference on Natural Language Processing (ICON)*, pages 60–69, 2020. 4, 23

[15] Chandrahas, Tathagata Sengupta, Cibi Pragadeesh, and Partha Pratim Talukdar. Inducing Interpretability in Knowledge Graph Embeddings. In *International Conference on Natural Language Processing (ICON)*, pages 70–75, 2020. 4

[16] Jonathan Chang, Sean Gerrish, Chong Wang, Jordan Boyd-graber, and David Blei. Reading tea leaves: How humans interpret topic models. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009. URL https://proceedings.neurips.cc/paper/2009/file/f92586a25bb3145facd64ab20fd554ff-Paper.pdf. 38

[17] Shuang Chen, Jinpeng Wang, Feng Jiang, and Chin-Yew Lin. Improving entity linking by modeling latent entity type information. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7529–7537, 2020. 43, 45, 47, 48, 54

[18] Eunsol Choi, Omer Levy, Yejin Choi, and Luke Zettlemoyer. Ultra-fine entity typing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 87–96, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1009. URL https://www.aclweb.org/anthology/P18-1009. 44, 54, 59, 62

[19] Tim Dettmers, Minervini Pasquale, Stenetorp Pontus, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the 32th AAAI Conference on Artificial Intelligence*, pages 1811–1818, February 2018. URL https://arxiv.org/abs/1707.01476. xvii, 2, 7, 9, 11, 23, 33, 34, 46, 47, 48, 51, 52, 66, 67, 68, 69, 73, 75

[20] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. ACL. doi: 10.18653/v1/N19-1423. URL https://www.aclweb.org/anthology/N19-1423. 46, 47, 52

[21] Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610. ACM, 2014. 1, 2, 11, 34, 66, 67

[22] Rotem Dror, Gili Baumer, Segev Shlomov, and Roi Reichart. The hitchhiker's guide to testing statistical significance in natural language processing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1383–1392. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/P18-1128. 55

[23] Mengnan Du, Fan Yang, Na Zou, and Xia Hu. Fairness in deep learning: A computational perspective. *IEEE Intelligent Systems*, 2020. 34

[24] Anthony Fader, Stephen Soderland, and Oren Etzioni. Identifying relations for open information extraction. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545. Association for Computational Linguistics, 2011. 2, 3, 42, 50

[25] Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah A. Smith. Sparse overcomplete word vector representations. 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 1491–1500, Beijing, China, July 2015. Association for Computational Linguistics. doi: 10.3115/v1/P15-1144. URL https://www.aclweb.org/anthology/P15-1144. 34, 38

[26] Evgeniy Gabrilovich, Michael Ringgaard, and Amarnag Subramanya. Facc1: Freebase annotation of clueweb corpora, version 1 (release date 2013-06-26, format version 1, correction level 0). 06 2013. 50

[27] Swapnil Gupta, Sreyash Kenkre, and Partha Talukdar. CaRe: Open knowledge graph embeddings. In *EMNLP-IJCNLP*, pages 378–388. ACL, 2019. 3, 43, 44, 46, 47, 50, 51, 52, 58, 60

[28] Arthur Colombini Gusmao, Alvaro Henrique Chaim Correia, Glauber De Bona, and Fabio Gagliardi Cozman. Interpreting embedding models of knowledge bases: a pedagogical approach. 2018. 35

[29] Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. OntoNotes: The 90% solution. In *NAACL-HLT, Companion Volume: Short Papers*, pages 57–60, New York City, USA, June 2006. ACL. URL https://www.aclweb.org/anthology/N06-2015. 44

[30] Zhipeng Huang, Bogdan Cautis, Reynold Cheng, and Yudian Zheng. Kb-enabled query recommendation for long-tail queries. In *CIKM*, CIKM '16, pages 2107–2112, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4073-1. doi: 10.1145/2983323.2983650. URL http://doi.acm.org/10.1145/2983323.2983650. 1, 43

[31] Prachi Jain, Pankaj Kumar, Mausam, and Soumen Chakrabarti. Type-sensitive knowledge base inference without explicit type supervision. In *ACL (Volume 2: Short Papers)*, pages 75–80, Melbourne, Australia, July 2018. ACL. doi: 10.18653/v1/P18-2013. URL https://www.aclweb.org/anthology/P18-2013. 45

[32] Prachi Jain, Sushant Rathi, Soumen Chakrabarti, et al. Knowledge base completion: Baseline strikes back (again). *arXiv preprint arXiv:2005.00804*, 2020. 36

[33] Shoaib Jameel, Zied Bouraoui, and Steven Schockaert. Member: Max-margin based embeddings for entity retrieval. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 783–792, 2017. 34

[34] Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020. doi: 10.1162/tacl_a_00300. URL https://www.aclweb.org/anthology/2020.tacl-1.5. 50, 79

[35] Rudolf Kadlec, Ondrej Bajgar, and Jan Kleindienst. Knowledge base completion: Baselines strike back. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 69–74, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-2609. URL https://www.aclweb.org/anthology/W17-2609. 36

[36] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *Advances in Neural Information Processing Systems*, pages 4284–4295, 2018. 66

[37] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *ICLR*, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS. 79

[38] Jey Han Lau, David Newman, and Timothy Baldwin. Machine reading tea leaves: Automatically evaluating topic coherence and topic model quality. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages

530–539, Gothenburg, Sweden, April 2014. Association for Computational Linguistics. doi: 10.3115/v1/E14-1056. URL https://www.aclweb.org/anthology/E14-1056. 34, 35, 36, 38

[39] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, pages 2181–2187, 2015. 1, 9, 11, 12, 34, 65, 66, 69

[40] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *JMLR*, 9(Nov): 2579–2605, 2008. 56

[41] Khai Mai, Thai-Hoang Pham, Minh Trung Nguyen, Tuan Duc Nguyen, Danushka Bollegala, Ryohei Sasano, and Satoshi Sekine. An empirical study on fine-grained named entity recognition. In *COLING*, pages 711–722, Santa Fe, New Mexico, USA, August 2018. ACL. URL https://www.aclweb.org/anthology/C18-1060. 44

[42] Mausam Mausam. Open information extraction systems and downstream applications. In *Proceedings of the twenty-fifth international joint conference on artificial intelligence*, pages 4074–4077, 2016. 43

[43] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NeurIPS*, pages 3111–3119, 2013. 20

[44] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38 (11):39–41, 1995. 15

[45] David Mimno and Laure Thompson. The strange geometry of skip-gram with negative sampling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2863–2868, 2017. 10, 15, 20

[46] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *AAAI*, 2015. 1

[47] Brian Murphy, Partha Pratim Talukdar, and Tom Mitchell. Learning effective and interpretable semantic models using non-negative sparse embedding. In *International Conference on Computational Linguistics (COLING 2012), Mumbai, India*. http://aclweb.org/anthology/C/C12/C12-1118.pdf, 2012. 34, 38

[48] Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. STransE: a novel embedding model of entities and relationships in knowledge bases. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 460–466, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1054. URL https://www.aclweb.org/anthology/N16-1054. 1, 9, 11, 12, 65, 66, 69

[49] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, volume 11, pages 809–816, 2011. 2, 34, 66

[50] Maximilian Nickel, Lorenzo Rosasco, and Tomaso A. Poggio. Holographic embeddings of knowledge graphs. In *AAAI*, 2016. xvii, 2, 9, 11, 13, 19, 66, 73

[51] Madhav Nimishakavi, Uday Singh Saini, and Partha Talukdar. Relation schema induction using tensor factorization with side information. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 414–423, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1040. URL https://www.aclweb.org/anthology/D16-1040. 43

[52] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *EMNLP-IJCNLP*, pages 2463–2473, Hong Kong, China, November 2019. ACL. doi: 10.18653/v1/D19-1250. URL https://www.aclweb.org/anthology/D19-1250. 43, 45, 47, 50

[53] Srinivas Ravishankar, Chandrahas, and Partha Pratim Talukdar. Revisiting simple neural networks for learning representations of knowledge graphs. *6th Workshop on Automated Knowledge Base Construction (AKBC) at NIPS 2017*, 2017. 11, 67

[54] Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. Relation extraction with matrix factorization and universal schemas. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 74–84, Atlanta, Georgia, June 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/N13-1008. 33, 34, 36, 37

[55] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You can teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BkxSmlBFvr. 36

[56] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. Modeling Relational Data with Graph Convolutional Networks. *ArXiv e-prints*, March 2017. 11, 33, 67, 73

[57] Sanket Shah, Anand Mishra, Naganand Yadati, and Partha Pratim Talukdar. Kvqa: Knowledge-aware visual question answering. In *AAAI*, 2019. 1

[58] Baoxu Shi and Tim Weninger. Proje: Embedding projection for knowledge graph completion. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 67

[59] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems*, pages 926–934, 2013. 11, 67

[60] Anant Subramanian, Danish Pruthi, Harsh Jhamtani, Taylor Berg-Kirkpatrick, and Eduard Hovy. Spine: Sparse interpretable neural embeddings. *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, 2018. 34

[61] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *WWW*, 2007. 1

[62] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *ICLR*, 2019. URL https://openreview.net/forum?id=HkgEQnRqYQ. 2, 66

[63] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *3rd Workshop on Continuous Vector Space Models and Their Compositionality*. ACL – Association for Computational Linguistics, July 2015. 37

[64] Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing Text for Joint Embedding of Text and Knowledge Bases. In *EMNLP*. ACL, sep 2015. 11, 33, 34, 37, 67, 75

[65] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *ICML*, 2016. 2, 9, 11, 13, 33, 34, 66, 73

[66] Shikhar Vashishth, Prince Jain, and Partha P. Talukdar. CESI: canonicalizing open knowledge bases using embeddings and side information. In *WWW 2018, Lyon, France, April 23-27, 2018*, pages 1317–1327, 2018. doi: 10.1145/3178876.3186030. URL https://doi.org/10.1145/3178876.3186030. 43, 44, 50

[67] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, Nilesh Agrawal, and Partha Talukdar. InteractE: Improving Convolution-based Knowledge Graph Embeddings by Increasing Feature Interactions. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, pages 3009–3016. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/5694. 2, 67, 76

[68] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *NeurIPS*, pages 5998–6008. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf. 46

[69] Quan Wang, Pingping Huang, Haifeng Wang, Songtai Dai, Wenbin Jiang, Jing Liu, Yajuan Lyu, Yong Zhu, and Hua Wu. COKE: Contextualized Knowledge Graph Embedding. *arXiv preprint arXiv:1911.02168*, 2019. 2, 45

[70] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, pages 1112–1119. Citeseer, 2014. 1, 9, 65, 69

[71] Han Xiao, Minlie Huang, and Xiaoyan Zhu. Knowledge semantic representation: A generative model for interpretable knowledge graph embedding. *arXiv preprint arXiv:1608.07685*, 2016. 34, 35

[72] Han Xiao, Minlie Huang, and Xiaoyan Zhu. TransG : A generative model for knowledge graph embedding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2316–2325, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1219. URL https://www.aclweb.org/anthology/P16-1219. 34

[73] Qizhe Xie, Xuezhe Ma, Zihang Dai, and Eduard Hovy. An interpretable knowledge transfer model for knowledge base completion. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 950–962, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1088. URL https://www.aclweb.org/anthology/P17-1088. 1, 34, 35, 65

[74] Ruobing Xie, Zhiyuan Liu, and Maosong Sun. Representation learning of knowledge graphs with hierarchical types. In *IJCAI*, IJCAI'16, page 2965–2971. AAAI Press, 2016. ISBN 9781577357704. 45, 61

[75] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014. 2, 3, 7, 9, 11, 13, 33, 34, 64, 66, 69, 70, 73

[76] Liang Yao, Chengsheng Mao, and Yuan Luo. KG-BERT: BERT for Knowledge Graph Completion. *ArXiv*, abs/1909.03193, 2019. 2, 45, 61, 67

[77] Xuchen Yao and Benjamin Van Durme. Information extraction over structured data: Question answering with Freebase. In *ACL (Volume 1: Long Papers)*, pages 956–966, Baltimore, Maryland, June 2014. ACL. doi: 10.3115/v1/P14-1090. URL https://www.aclweb.org/anthology/P14-1090. 1, 43

[78] Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. ERNIE: Enhanced language representation with informative entities. In *ACL*, pages 1441–1451, Florence, Italy, July 2019. ACL. doi: 10.18653/v1/P19-1139. URL https://www.aclweb.org/anthology/P19-1139. 79