Knowledge Graph Embeddings: Link Prediction and Beyond

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von
Daniel Ruffinelli
aus Asuncion, Paraguay

Mannheim, 2023

Dekan	Prof. Dr. Claus Hertling, Universität Mannheim
Referent	Prof. Dr. Rainer Gemulla, Universität Mannheim
Korreferent	Prof. Dr. Simone Paolo Ponzetto, Universität Mannheim
Korreferent	Prof. Dr. Fabian Suchanek, Institut Polytechnique de Paris

Tag der mündlichen Prüfung: 22.11.2023

ABSTRACT

Knowledge graph embeddings, or KGEs, are models that learn vector representations of knowledge graphs. These representations have been used for tasks such as predicting missing links in the graph, or as pre-trained representations that encode structured data for downstream applications, such as question answering or recommender systems. Despite the large amount of models developed for this purpose, the variety in experimental settings has made it difficult to compare results across different studies. Models are often learned using different training and hyperparameter optimization strategies. In addition, most of the literature has focused on a specific form of predicting missing links, known as *link prediction*. Almost no attention was given to predicting other types of structures in a knowledge graph, and despite their use in downstream applications, there are virtually no studies on the usability of KGE models as pre-trained representations of knowledge graphs.

In this thesis, we propose new training and evaluation methods and conduct several large scale empirical studies, all aimed at studying KGE models as a form of knowledge representation. First, we compare model performance in a fair experimental setting that allows us to separate between contributions from new models and those from new training strategies. We find that differences in training approaches, and not necessarily in model architectures, may account for much of the previously reported progress in link prediction. Second, we study some potential limitations that may result from focusing almost exclusively on the link prediction task for KGE research. We find that good link prediction models are not necessarily able to successfully predict missing links in a knowledge graph, and that link prediction performance is not an indication that models generally capture information in the graph. This contradicts the common argument that KGE models are able to generally preserve the structure in a knowledge graph.

Finally, we look beyond the link prediction task and study different training objectives aimed at capturing more information in the graph, and the impact that the resulting representations have on downstream applications. We find that models trained with the standard approach based on link prediction do not capture as much information about the graph as possible, and that link prediction performance is also not a good indicator for good downstream performance. These results suggest that the relation between pre-training objectives and downstream performance is not as clear as suggested in the literature, and that more research is needed to better understand how to learn generally useful representations of knowledge graphs.

ZUSAMMENFASSUNG

Wissensgrapheneinbettungen (Knowledge Graph Embeddings, KGEs) sind Modelle, die Vektordarstellungen von Wissensgraphen lernen. Diese Repräsentationen wurden für Aufgaben wie die Vorhersage fehlender Links im Graphen oder als vortrainierte Repräsentationen verwendet, die strukturierte Daten für nachgelagerte Anwendungen wie Fragebeantwortungs- oder Empfehlungssysteme kodieren. Trotz der großen Anzahl von Modellen, die für diesen Zweck entwickelt wurden, ist es aufgrund der Vielfalt der Versuchsbedingungen schwierig, die Ergebnisse verschiedener Studien zu vergleichen. Die Modelle werden oft mit unterschiedlichen Trainings- und Hyperparameter-Optimierungsstrategien erlernt. Darüber hinaus konzentrierte sich der Großteil der Literatur auf eine spezielle Form der Vorhersage fehlender Verbindungen, die so genannte Link-Prediction. Der Vorhersage anderer Arten von Strukturen in einem Wissensgraphen wurde fast keine Aufmerksamkeit geschenkt, und trotz ihrer Verwendung in nachgelagerten Anwendungen gibt es praktisch keine Studien über die Verwendbarkeit von KGE-Modellen als vortrainierte Repräsentationen von Wissensgraphen.

In dieser Arbeit schlagen wir neue Trainings- und Bewertungsmethoden vor und führen mehrere groß angelegte empirische Studien durch, die alle darauf abzielen, KGE-Modelle als eine Form der Wissensrepräsentation zu untersuchen. Zunächst vergleichen wir die Leistung der Modelle in einem fairen experimentellen Rahmen, der es uns ermöglicht, zwischen den Beiträgen der neuen Modelle und denen der neuen Trainingsstrategien zu unterscheiden. Wir stellen fest, dass Unterschiede in den Trainingsansätzen und nicht notwendigerweise in den Modellarchitekturen für einen Großteil der zuvor berichteten Fortschritte in der Linkvorhersage verantwortlich sein könnten. Zweitens untersuchen wir einige potenzielle Einschränkungen, die sich aus der fast ausschließlichen Konzentration auf die Aufgabe der

Link-Vorhersage in der KGE-Forschung ergeben können. Wir stellen fest, dass gute Linkvorhersagemodelle nicht unbedingt in der Lage sind fehlende Links in einem Wissensgraphen erfolgreich vorhersagen können, und dass die Leistung der Linkvorhersage kein Hinweis darauf ist, dass die Modelle generell Informationen im Graphen erfassen. Dies widerspricht dem gängigen Argument, dass KGE-Modelle in der Lage sind, die Struktur eines Wissensgraphen im Allgemeinen zu erhalten.

Schließlich gehen wir über die Aufgabe der Linkvorhersage hinaus und untersuchen verschiedene Trainingsziele, die darauf abzielen, mehr Informationen im Graphen zu erfassen, sowie die Auswirkungen, die die resultierenden Darstellungen auf nachgelagerte Anwendungen haben. Wir stellen fest, dass Modelle, die mit dem Standardansatz der Linkvorhersage trainiert werden, nicht so viele Informationen über den Graphen erfassen wie möglich, und dass die Vorhersageleistung auch kein guter Indikator für eine gute nachgelagerte Leistung ist. Diese Ergebnisse deuten darauf hin, dass die Beziehung zwischen den vor dem Training gesetzten Zielen und der nachgelagerten Leistung nicht so klar ist, wie in der Literatur angenommen wird, und dass weitere Forschung erforderlich ist, um besser zu verstehen, wie man allgemein nützliche Repräsentationen von Wissensgraphen erlernen kann.

ACKNOWLEDGEMENTS

In the early years of studying for my engineering degree, I was introduced to, and subsequently became fascinated with, the history and philosophy of science. Through the works of Karl Popper and Thomas Kuhn, I learned about the scientific method which, based on observation, rigor and reproducible evidence, brought about long term improvement to our lives. It is my belief that the general focus on obtaining a high number of (often speedy) publications has resulted in a process that often neglects those core principles that made the scientific method the revolution that it was. I say this, so the reader may appreciate how much I am thankful for having had Rainer Gemulla as my adviser throughout my PhD. Instead of focusing on publications, Rainer's primary interest has always been in deriving meaningful insights that build understanding on top of the latest related work. This process was often tough, but it is exactly what I was looking for in a PhD. Far from a stricter alternative, I belief this is a better reflection of the scientific method, and should thus be the norm. Thank you, Rainer!

In addition to my adviser, there are many people to whom I owe my gratitude. I was fortunate enough to have had colleagues that acted as mentors to me early in my PhD. When I most needed it, Christian Meilicke, Yanjie Wang and Samuel Broscheit patiently shared their wisdom with me, and I am happy to have had the opportunity to collaborate with them. Through one way or another, I have also worked with many professors at the Data and Web Science Group: Heiner (without whom I would not be here), Simone, Heiko, Chris and Goran. I am thankful for all of their useful advice. I have also learned a lot from other PhD students throughout these years, many of whom I am happy to call friends today. Thank you all for the very important debates about food, music, and other ways to enjoy life. *Jungbusch* would not be the same without you!

I am fortunate enough to have too many friends to name, but I do want to mention one person by name. My best friend and wife Helga, the best partner I could have ever hoped for. I am incredibly grateful for her constant support throughout these years, be it through listening, giving advice, or simply talking for hours about anything and everything while listening to music over a weekend breakfast. I love you! Thank you for coming with me on this adventure!

Finally, I would like to thank Hidetaka Miyazaki for the amazing experiences I have been able to enjoy through his creations.

CONTENTS

CHAPTER

ONE

INTRODUCTION

"Reality is frequently inaccurate."

Douglas Adams

Knowledge graphs encode real-world information in the form of directed labeled multigraphs. Their use is prevalent in many application scenarios where there is a need to integrate and extract value from data at large scale (Hogan et al., 2021). Some examples of such applications are web search (Shrivastava, 2017; Singhal, 2012), social networks (Noy et al., 2019; He et al., 2016) and recommender systems (Chang, 2018; Hamad et al., 2018). In such scenarios, knowledge graphs can grow to the size of billions of edges (Pellissier Tanon et al., 2016), presenting challenges to their construction, maintenance and use (Noy et al., 2019).

Over the past decade, the study and development of methods for learning vector representations, or *embeddings*, of knowledge graphs has been a very active research area (Nickel et al., 2015; Wang et al., 2017; Ji et al., 2021). This follows from the success of using learned representations in areas such as natural language processing (Kamath et al., 2019) or computer vision (Szeliski, 2022). A *knowledge graph embedding* model, or KGE, learns vector representations of the entities and relations in a knowledge graph. These representations capture the general patterns and structure in the knowledge graph (Trouillon et al., 2016; Bordes et al., 2013b), which makes them useful

for various purposes, such as predicting missing facts in the knowledge graph (Balazevic et al., 2019; Sun et al., 2019), or to enhance performance in knowledge-intensive applications such as question answering (Ilyas et al., 2022) or recommender systems (El-Kishky et al., 2022).

Despite the large amount of works that have proposed new KGE models (Ji et al., 2021), it is often difficult to compare model performance across different studies due to the variety in experimental settings. Models are often learned using different training and hyperparameter optimization strategies, and performance is often compared across different implementations of these training scenarios. In addition, most of the literature on KGE models has centered around a specific form of predicting missing links in a knowledge graph, known as *link prediction*. Almost no attention was given to predicting other types of structures in a knowledge graph, such as predicting the neighborhood of an entity or the domain of a relation, and despite their use in downstream applications, there are virtually no studies on embedding quality. That is, on the usability of the representations from KGE models as pre-trained representations of knowledge graphs.

In this thesis, we propose new training and evaluation methods and conduct several large scale empirical studies, all aimed at studying KGE models as a form of knowledge representation. First, we compare model performance in a fair experimental setting that allows us to separate between contributions from new models and those from new training strategies. We find that differences in training approaches, and not necessarily in model architectures, may account for much of the previously reported progress in link prediction. Second, we study some potential limitations that may result from focusing almost exclusively on the link prediction task for KGE research. We find that good link prediction models are not necessarily able to successfully predict missing links in a knowledge graph, and that link prediction performance is not an indication that models generally capture information in the graph. This contradicts the common argument that KGE models generally preserve the structure in a knowledge graph.

Finally, we look beyond the link prediction task and study different training objectives aimed at capturing more information in the graph, and the impact that the resulting representations have on downstream applications. We find that models do not capture as much information about the graph as possible with the standard training approach, and that link prediction performance is also not a good indicator for good downstream performance.

These results suggest that the relation between pre-training objectives and downstream performance is not as clear as suggested in the literature, and that more research is needed to better understand how to learn generally useful representations of knowledge graphs.

Contributions

The main contributions in this thesis are:

1. We conduct a large experimental study to compare the link prediction performance of several popular KGE models all under the same training conditions. We find that KGE models are very sensitive to training settings as well as to hyperparameter optimization, and that given more recent training methods, models that underperform in prior work become competitive with, or even outperform, state-of-the-art models. This suggests that observations made in studies that compare published results across different experimental settings may need to be revised, and that future work should provide the same resources to all models to ensure a fair assessment.

2. We design a task we call knowledge base completion, which is a generalization of the link prediction task, and propose a new evaluation method based on this new task to assess whether KGE models can more generally predict missing links in a knowledge graph. With this approach, we illustrate that the standard form of evaluating link prediction performance is limited in that models that fail to capture large areas of a knowledge graph still perform well in standard link prediction. In contrast, our experimental result show that our proposed evaluation method makes a clear distinction between more and less expressive KGE models.

3. We propose a generalization of the standard evaluation method for link prediction, so that it may be used to evaluate model performance on any number of prediction tasks. Using this evaluation approach in combination with a new set of tasks for predicting different structures in a graph, we assess the extent to which KGE models can make different types of predictions about the graph they encode. We find that models with strong link prediction performance are often not those that are best

at making predictions about the graph more generally. These results challenge the intuition that KGE models preserve the general structure of a knowledge graph.

4. We propose a generalization of the standard training method based on link prediction to an approach for simultaneously training models on multiple tasks beyond link prediction. Further, we extend the ability of KGE models to efficiently answer new types of queries. We compare the performance of models that are trained with the standard link prediction approach and our proposed multi-task training approach. We find that KGE models can indeed learn to simultaneously capture more information about a graph when trained on multiple tasks.

5. We conduct a large experimental study to assess the impact that different pre-training methods have on the resulting embeddings when used in downstream applications. We collect and create 35 different datasets for downstream tasks and compare performance of several downstream models that use knowledge graph embeddings trained with the standard approach and our proposed multi-task approach. We find that link prediction performance is not a good indicator for good downstream performance, and that multi-task pre-training provides benefits in downstream tasks most of the time. However, including more tasks during training does not often lead to improved downstream tasks, suggesting that more research is needed to better understand the relation between pre-training KGE models and their usability in downstream applications. To assist with future research in this direction, we provide our collection of downstream datasets to assess embedding quality, as well as code for all of our proposed methods and empirical studies, implemented as part of the open source framework LibKGE.

Publications

The work presented in this thesis is based on the following publications:

- Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, Samuel Broscheit, and Christian Meilicke. On Evaluating Embedding Models for Knowledge Base Completion. In *4th Workshop on Representation Learning for NLP (Rep4NLP@ACL)*, 2019. Received Outstanding Paper Award

- 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 (ICLR)*, 2020
- Daniel Ruffinelli and Rainer Gemulla. Beyond Link Prediction: On Pre-Training Knowledge Graph Embeddings. 2023. *Under Submission*

We also contributed to, and draw insights from, the following works:

- Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. Fine-Grained Evaluation of Rule- and Embedding-based Systems for Knowledge Graph Completion. In *International Semantic Web Conference (ISWC)*, 2018
- Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019
- Haris Widjaja, Kiril Gashteovski, Wiem Ben Rim, Pengfei Liu, Christopher Malon, Daniel Ruffinelli, Carolin Lawrence, and Graham Neubig. KGxBoard: Explainable and Interactive Leaderboard for Evaluation of Knowledge Graph Completion Models. In *Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP)*, 2022

Outline

We introduce relevant background concepts and methods in Chapter 2. In Chapter 3 we discuss the large-scale study where we compare the link prediction performance of various models all under the same training conditions. We introduce our proposed knowledge base completion task and the evaluation method we designed based on this task in Chapter 4. We propose a generalization of the standard training and evaluation approaches in Chapter 5 and present the corresponding experimental results. In Chapter 6 we discuss our experimental study comparing the effect that different pre-training approaches have when KGE models are used in downstream applications. We conclude our thesis in Chapter 7, where we also briefly discuss opportunities for future work.

CHAPTER

TWO

BACKGROUND

In this chapter, we introduce the fundamental concepts and methods that underpin the work in this thesis. We start with knowledge graphs, and then introduce knowledge graph embeddings, which are the main focus of this work. We follow with a brief discussion on other families of related models, and finish with a description of the relevant datasets used in this work.

2.1 Knowledge Graphs

Many of the properties of knowledge graph embeddings, both observed and desired, come from the objects these models aim to represent: knowledge graphs. Consequently, discussing relevant concepts about knowledge graphs is important to understand the motivation behind most of our proposed methods and experimental studies. In this section, we give a brief introduction to knowledge graphs as they are used in this work. For a comprehensive introduction, see Hogan et al. (2021).

A knowledge graph (KG) encodes real-world information in the form of a directed labeled multigraph, where nodes correspond to any type of real-world entities, such as cities or famous persons, and edges represent relations between two entities. Figure 2.1 shows an example of a small knowledge graph that represents basic geographical facts about a few cities and states in the United States of America (USA).

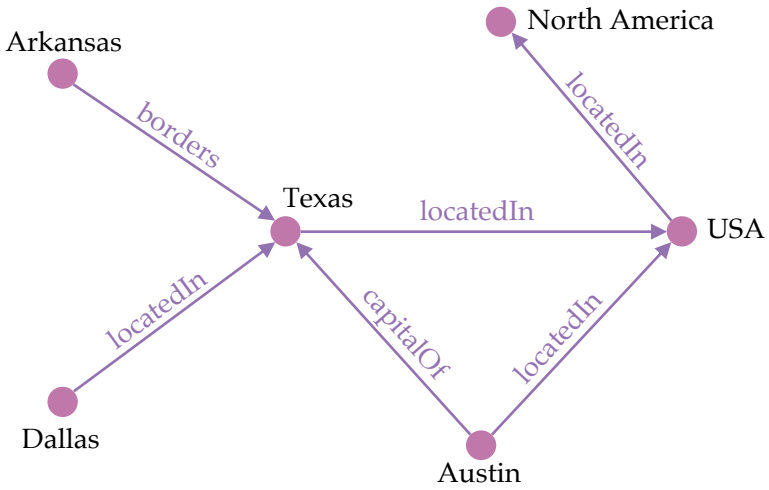


Figure 2.1: Example of a small knowledge graph

Following the standard data model for KGs, the Resource Description Framework (RDF) (W3C, 2014), we represent a knowledge graph as a set of *(subject, predicate, object)* or *(s,p,o)* triples. For example, the triple *(Austin, capitalOf, Texas)* represents the fact that the city of Austin is the capital of the state of Texas in the USA. Table 2.1 shows the set of triples that correspond to the KG in Figure 2.1. We formally define a KG as follows.

Definition 2.1.1 (Knowledge Graph)**.** Given a finite set $\mathcal{E}$ of entities and a finite set $\mathcal{R}$ of relations, a knowledge graph $\mathcal{K} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ is a set of *(subject, predicate, object)* triples, each representing a fact in $\mathcal{K}$.

Although generally not required, some KGs follow to a pre-defined schema, i.e. a set of rules that describe the constraints that the entities and relations in a KG must adhere to. For example, the KG in Figure 2.1 may describe that each entity is an instantiation of a more general class, such as *(Texas, instanceOf, city)*, where *city* is a node that represents the concept of a city. Similarly, relations in the KG may have constrained domain and range, e.g. *(capitalOf, hasDomain, city)* and *(capitalOf, hasRange, state)*. It is worth noting that, even in the absence of an explicit schema, the triples that make up a KG typically follow some set of implicit rules, e.g. relations have a specific domain and range, or some relations have certain properties, such as transitivity or symmetry.

Subject	Predicate	Object
Dallas	locatedIn	Texas
Austin	capitalOf	Texas
Austin	locatedIn	Texas
Arkansas	borders	Texas
USA	locatedIn	North America
Austin	locatedIn	USA

Table 2.1: Knowledge graph from Figure 2.1 as a set of triples

In this work follow, we follow the *open world assumption* (OWA) when working with KGs. That is, we assume that any fact missing from the KG is unknown, i.e. either true or false. In contrast, the *closed world assumption* (CWA) states that missing facts are assumed to be false, such as in traditional relational database systems.

Knowledge graphs may also contain *literals*, i.e. nodes that do not represent a real-world entity, but instead hold information about an entity node in the form of strings, integers or dates. For example, the triple *(USA, founded, 04/07/2023)* may be used to represent the date when the USA was founded. In this work, we focus on KGs that do not contain literals.

2.1.1 Properties

The quality of the data in a KG may be described by several properties (Hogan et al., 2021). For example, *timeliness* refers to the relevance of the data across time, and *provenance* refers to the process by which the data was obtained. Two arguably more fundamental properties of KGs, and the two that are relevant in this work, are *accuracy* and *completeness*.

Definition 2.1.2 (KG Syntactic Accuracy)**.** A KG is more syntactically accurate the more its facts are accurate w.r.t. the rules established by the data model, e.g. domain and range restrictions in relations.

Definition 2.1.3 (KG Semantic Accuracy)**.** A KG is more semantically accurate the more its edges correctly represent real-world facts.

Definition 2.1.4 (KG Completeness)**.** A KG is more complete the higher the percentage of relevant real-world facts are represented within it.

Note that completeness implies an ideal set of facts that a KG should have. In practice, completeness can be (at least partially assessed) by using a larger set of known facts (Darari et al., 2018).

2.1.2 Construction Methods

Knowledge graphs can be constructed in various ways, each of which has an impact on its completeness and accuracy. Methods for constructing KGs can be divided into *automated* or *non-automated* and *structured* or *unstructured* (Nickel et al., 2015).

An automated approach is one that automatically extracts facts from various sources, whereas a non-automated approach typically means having a group of experts manually create a set of triples. While the latter implies higher quality, its cost may be prohibitively large depending on the size of the KG and the field of knowledge it represents. Further, there may not be agreement among experts w.r.t. what some of the facts are. Automated approaches are more cost effective, but the quality of the resulting KG may suffer, both in terms of accuracy and completeness, depending on the extraction method and sources of information.

Structured approaches are those whose source of information is structured data, e.g. web tables (Luzuriaga et al., 2021) or Wikipedia infoboxes (Peng et al., 2019). Conversely, unstructured methods extract information from unstructured sources, such as publicly available text (Gashteovski et al., 2017). We discuss how the KGs that are relevant for this thesis were constructed in Section 2.4.

2.1.3 Applications

The main purpose of KGs is to provide structured data that can be interpreted by automated systems in a semantically meaningful way across various application scenarios, such as social network analysis or recommender systems (Nickel et al., 2015). A common mechanism by which users can extract information from KGs is logical reasoning. For example, given the KG in Figure 2.1, the facts *(Texas, locatedIn, USA)* and *(USA, locatedIn, North America)* could lead us to infer the fact *(Texas, locatedIn, North America)*. KGs are usually consulted via a given query language, such as SPARQL, which was designed for RDF graphs (W3C, 2013). The cost of extracting information from a KG

typically depends on the size of the KG and the expressivity of the query language. We discuss in Section 2.1.5) how this can become a challenge when using KGs in real-world scenarios.

Depending on their intended purpose, KGs may vary in size and scope. Some KGs are limited in scope, such as the Gene Ontology, which provides information about the relations between genes and the biochemical materials that result from the expression of these genes (GOC, 2021). The Gene Ontology contains data about 27 942 biological processes, 11 263 molecular functions and 4043 cellular components (GOC, 2023). Conversely, the Google Knowledge Graph (Singhal, 2012) is a general purpose graph that contains billions of facts about the world. It was designed to enhance the company's search engine with semantically meaningful results. The actual size of this KG is unknown, as this is not an open graph, but the private property of a company. Other companies in the industry have also created knowledge graphs for internal use, such as Amazon (Krishnan, 2018), Facebook (Noy et al., 2019) or Microsoft (Shrivastava, 2017).

2.1.4 Open Knowledge Graphs

An open KG is one that is publicly available to query, download and use for any purpose (Hogan et al., 2021). In Section 2.4, we discuss the benchmark datasets used in this work, all of which are subsets of the following KGs.

Freebase (Bollacker et al., 2007) was a KG with the purpose of encoding general human knowledge. At launch, it was made up of more than 125 000 000 triples (Bollacker et al., 2008) and it was designed to provide integrated support for the semantic web (Hitzler, 2021). Its construction was based on collaborative contributions from human editors and it was possible to extend it via a web user interface. Freebase was deprecated when it was acquired by Google and became an important part in the creation of the Google Knowledge Graph (Singhal, 2012).

WordNet (Miller, 1995) is a domain-specific KG. It was designed as a lexical database that could be easily processed with machines and thus provide support for automated natural language tasks. WordNet was manually constructed by experts and contains more than 118 000 word forms and more than 90 000 different word meanings.

YAGO (Suchanek et al., 2007) is another general-purpose KG, constructed

by automatically extracting information from structured data in Wikipedia and using WordNet to integrate and disambiguate facts and concepts. At launch, it contained about 5 million facts. YAGO was designed to provide support for various tasks, such as machine translation or document classification, among others.

Wikidata (Vrandečić and Krötzsch, 2014) was designed as a centralized database that stores all of the structured information contained in Wikipedia, with the aim of providing easier access to such structured data. At launch, it contained over 43 million statements. As with Wikipedia, collaborative editing is allowed and providing references to claims is encouraged.

2.1.5 Challenges

Despite their wide applicability, KGs have known limitations. Those most relevant for this work are their incompleteness and their cost of inference.

Due to their construction methods, KGs are often incomplete. For example, KGs that are constructed based on information in Wikipedia would only include facts contained in Wikipedia, such as the high amount of data about American actors and actresses and the lack of corresponding information about Indian or Nigerian actors or actresses, despite the latter two being larger film industries (Nickel et al., 2015). Similarly, the place of birth of 71% of the people included in Freebase was reportedly missing (West et al., 2014).

As mentioned in Section 2.1.3, the cost of deductive reasoning used to infer missing or new data in a KG depends on the language bias of the query language, i.e. what types of queries are supported, and on the size of the KG. In general, query answering can be undecidable given languages that are expressive enough (Hitzler et al., 2010). For this reason, a multitude of less expressive languages, i.e. languages that only support restricted types of queries, have been developed to allow for efficient reasoning. These range from IF-THEN rules, such as Datalog for databases (Ceri et al., 1989), to the various forms of Description Logics designed for the semantic web (Baader et al., 2017).

Challenges such as the ones described above have been important motivations for the design of knowledge graph embedding models, which we introduce in the next section.

Numbers, Arrays, and Sets	
a	A scalar
$\boldsymbol{a}$	A column vector
$\boldsymbol{A}$	A matrix
a_{ij}	Element at position i, j of matrix $\boldsymbol{A}$
$\boldsymbol{a}_i$ or $\boldsymbol{a}_{i:}$	Column vector of row i of matrix $\boldsymbol{A}$
$\boldsymbol{a}_{:j}$	Column vector of column j of matrix $\boldsymbol{A}$
$\mathcal{A}$	A set
$\mathbb{R}$	The set of real numbers
$\mathbb{C}$	The set of complex numbers
$a \in \mathcal{A}$	a is an element of set $\mathcal{A}$
$\mathcal{A} \subset \mathcal{B}$	Set $\mathcal{A}$ is a subset of set $\mathcal{B}$
$\mathcal{A} \cup \mathcal{B}$	Union of sets $\mathcal{A}$ and $\mathcal{B}$
$\mathcal{A} \cap \mathcal{B}$	Intersection of sets $\mathcal{A}$ and $\mathcal{B}$
Linear Algebra Operations	
$\boldsymbol{a}^T$	Transpose of vector $\boldsymbol{a}$
$\boldsymbol{A}^T$	Transpose of matrix $\boldsymbol{A}$
$\boldsymbol{a}^T\boldsymbol{b}$	Dot product between $\boldsymbol{a}$ and $\boldsymbol{b}$
$\boldsymbol{a} \circ \boldsymbol{b}$	Element-wise (Hadamard) product between $\boldsymbol{a}$ and $\boldsymbol{b}$
$\boldsymbol{a}\boldsymbol{A}$	Matrix-vector product between $\boldsymbol{a}$ and $\boldsymbol{A}$
$\text{diag}(\boldsymbol{a})$	Diagonal matrix with vector $\boldsymbol{a}$ in main diagonal

Table 2.2: Mathematical notation used throughout this thesis.

2.2 Knowledge Graph Embeddings

In this section, we introduce the main focus of this thesis: knowledge graph embedding (KGE) models. KGEs are dense vector representations of the entities and relations in a KG. Therefore, we give a brief introduction into dense representations before discussing KGEs in general.

2.2.1 Mathematical Notation

Table 2.2 introduces the mathematical notation used throughout this thesis.

2.2.2 Distributed Representations

Most of the recent progress in some areas of machine learning, such as natural language processing and computer vision, is based on models that rely on dense representations instead of local, or sparse, representations (Kamath et al., 2019; Szeliski, 2022). Local representations refer to the mapping of a single representational element to a single concept from a set of concepts to be represented. For example, a sequence of text could be represented using *one-hot* encoding vectors, i.e. vectors with binary indicator features, where each feature indicates whether a specific word is present in the sequence. Thus, each feature is mapped to a single word and a sequence of text is represented by a sparse vector with dimension equal to the size of the available vocabulary.

In contrast, dense (or distributed) representations rely on a many-to-many relation between representational components and the concepts or entities that are to be represented (Hinton, 1984). That is, a single element may be used to represent multiple concepts, and multiple concepts may be represented by multiple elements. Note that representational elements need not be binary as in a *one-hot* encoding vector. This results in dense, i.e. non-sparse, vector representations, which allow for the size of the representation vector to be much smaller than when relying on sparse representations. Much of the success in machine learning in recent years has been driven by models that learn such dense representations, which are in turn useful as input representations for downstream tasks (Bengio et al., 2000; Bengio, 2008; LeCun et al., 2015; Zhuang et al., 2021). Such dense representations are often referred to as *embeddings*.

Dense representations are also related to latent feature models (Orbanz and Teh, 2010). These models represent concepts as a set of latent features, i.e. features not directly present in the data, but fundamentally underlying it. Such models were widely applied in recommender systems based on matrix factorization techniques (Koren et al., 2009) and have also been successfully applied to graphs (Orbanz and Roy, 2014).

2.2.3 Representations of Knowledge Graphs

Methods that provide vector representations of knowledge graphs have been proposed and developed with different purposes in mind. Paccanaro

and Hinton (2001) focused on the task of inferring new instances of binary relations between entities in a KG. Three components of their work are still commonly used today by models that focus on that same task. First, they modeled this task as predicting a missing element of a triple, e.g. *(Alberto, hasFather, ?)*. We call this task link prediction and discuss it more formally in Section 2.2.4. Second, inspired by Hinton (1984), their method was based on learning distributed vector representations of each entity by embedding them into a multidimensional Euclidean space. They modeled the relations between entities in the KG as relation-dependent transformations that, when applied to the embedding of the subject of a triple, should result in an approximation of the embedding of the object in that triple. Finally, they represented the plausibility that a triple is true with a real value given by a function that takes a triple as input. We call such functions *score functions*. This score function was parameterized by the entity and relation embeddings, and their training approach was based on increasing the scores of known edges in the graph w.r.t. to all other possible edges.

Inspired by the success of collaborative filtering in recommender systems, a different set of methods was proposed to solve the same task (but referred to as relational learning in this line of work) using methods based on matrix factorization. Singh and Gordon (2008) proposed collective matrix factorization, where they followed a collaborative filtering approach to simultaneously learn relations between different sets of entities, e.g. users and movies in a recommendation system. Similarly, Nickel et al. (2011) proposed a tensor factorization approach that, as in collaborative filtering, results in dense representations of the entities in the data. We discuss this last model further in Section 2.2.5.

With perhaps a more general purpose in mind, Bordes et al. (2011) proposed a neural network architecture designed to embed a knowledge graph into a continuous vector space in order to gain flexibility while preserving its knowledge. Such representations of a KG would allow the injection of the knowledge represented in the KG into machine learning methods that commonly rely on feature vectors. They learned vector representations for each entity and a pair of matrices that acted as linear transformations for each relation. During training, their model learned to rank the scores of known triples lower than that of negative triples. The latter were generated by replacing the subject or object of a triple by a randomly chosen entity from the set of entities, a process that is still used by models (see Section 2.2.7).

Thus, their approach was also based on the link prediction task as in the work of Paccanaro and Hinton (2001).

The works described above from Paccanaro and Hinton (2001), Bordes et al. (2011) and Nickel et al. (2011) were some of the first to introduce the motivation and methods that became the starting point of research into knowledge graph embeddings (KGE). With a clearer picture of such earlier works, we define KGE models as follows:

Definition 2.2.1 (Knowledge Graph Embedding Model)**.** Given a knowledge graph $\mathcal{K} \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, a KGE model associates each entity $i \in \mathcal{E}$ and relation $k \in \mathcal{R}$ with an embedding $\boldsymbol{e}_i \in \mathcal{R}^{de}$ and $\boldsymbol{r}_k \in \mathcal{R}^{dr \times dr}$ in a low-dimensional vector space, respectively. These embeddings act as parameters for a score function $s : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightarrow \mathbb{R}$ that represents the plausibility of any triple (i, k, j) with a real number. The score function takes the form $s(i, k, j) = f(\boldsymbol{e}_i, \boldsymbol{r}_k, \boldsymbol{e}_j)$, i.e., it depends on i, k, and j only through their respective embeddings.

Typically, models are trained to associate higher scores to positive triples, i.e. those known to be correct, and lower scores to negative or pseudo-negative triples, which are obtained via some protocol for generating negative examples. We describe the training process of KGEs in detail in Section 2.2.7.

Since the earliest proposals described above, dozens of KGE models have been proposed, most of which share the same motivation as the works described above. For a comprehensive list of these proposed models, see the surveys from Ji et al. (2021) and Wang et al. (2017). In the following sections, we first discuss the tasks that motivated the design of KGE models in general, and then formally introduce the specific KGE models that are relevant for this work.

2.2.4 Link Prediction and other Applications

As we will see in the following sections, link prediction has been a fundamental task for KGE models, not only as a goal, but also when designing approaches for training and evaluating models. This focus on link prediction has in fact motivated the work we present in Chapters 4 to 6. In this section, we formally introduce the link prediction task. We also briefly discuss other types of tasks that KGEs are useful for, such as predicting other types of graph structure and downstream applications.

Link prediction. Generally, link prediction relates to a model's ability to predict missing edges in a KG. More specifically, the link prediction task has been commonly defined in the literature as follows.

Definition 2.2.2 (Link Prediction)**.** Given a knowledge graph $\mathcal{K} \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, link prediction is the task of predicting a missing entity in a given triple $(i, k, j) \in \mathcal{K}$, i.e. answering queries *(i,k,?)* or subject prediction, and *(?,k,j)* or object prediction, with j and i, respectively.

The development of KGE models was partially motivated by some advantages that such models could have on the link prediction task compared to other types of approaches, e.g. rule-based models (Nickel et al., 2015). First, KGES are significantly more efficient at inference time, because compared to rule-based reasoning, models need to compute the score of a given triple, which is often a single operation using the corresponding embeddings. Second, they should in principle be more robust to noise, since noisy data should have less of an impact on a learned vector space than on the learning of a set of discrete rules. However, we are not aware of any formal study into KGE's ability to handle noise in comparison to rule-based models. We discuss rule-based models in more detail, including their advantages compared to KGE models, in Section 2.3.3, and compare their performance on link prediction and other tasks in Chapters 3 and 4.

In addition, distributed representations of entities and relations in a KG can be very useful for tasks like entity clustering, entity linking across different KGs, or even identifying errors in the KG by checking the scores of existing edges.

Other graph structure tasks. KGEs can also perform predictions about other types of structure in a KG. For example, some works also refer to the task of predicting the missing relation in a triple, i.e. *(s,?,o)* as link prediction. We follow Chang et al. (2020) and Chen et al. (2021c) and refer to this task as *relation prediction*.

In addition to relation prediction, some works study the triple classification task (Socher et al., 2013; Safavi and Koutra, 2020), i.e. given a triple t, predict whether $t \in \mathcal{K}$ or $t \notin \mathcal{K}$. Although we do not study the triple classification task in this thesis, we discuss it further in Section 2.2.6. For a discussion on other graph-structure tasks, see Wang et al. (2017).

Downstream applications. In addition to tasks that aim at predicting structure in the KG, the learned embeddings provided by KGE models may

be used as features in downstream applications that may benefit from the information in the corresponding KG. In this sense, it is crucial that models actually preserve the information in the KG, which is indeed the goal of many proposed KGE models (Bordes et al., 2011, 2013b; Trouillon et al., 2016). There are several different types of downstream models that make use of pre-trained KGE models, such as recommender systems (El-Kishky et al., 2022; Wang et al., 2018a), language models (He et al., 2020; Zhang et al., 2019b), visual models (Baier et al., 2017), and question answering systems (Ilyas et al., 2022). Different models use the learned representations from KGEs differently depending on the downstream application. We discuss downstream applications in more detail in Chapter 6. For more comprehensive discussions on the use of KGE models in downstream applications, see Wang et al. (2017) and Ji et al. (2021).

2.2.5 Models

KGE models are mostly differentiated by the assumptions they make, which are best summarized by their score functions. In this section, we introduce the models that are part of our experiments in later chapters, based on the type of score function they use. We focus on what makes models distinct from one another and leave the discussion of hyperparameters that apply to all models for Section 2.2.7. In addition, we comment on the more theoretical aspects of these models, such as model expressivity, at the end of this section. Later, in Section 2.3, we briefly discuss related families of models, including more recent link prediction models that do not necessarily learn vector representations of the components of a KG. For all models discussed below, let d be a hyperparameter, $N = |\mathcal{E}|$ and $K = |\mathcal{R}|$, where $\mathcal{E}, \mathcal{R}$ are the sets of entities and relations in some knowledge graph $\mathcal{K}$, respectively.

Bilinear Models

Often referred to as factorization-based models (Ji et al., 2021), bilinear models are those with the following score function:

$$s(i, k, j) = \boldsymbol{e}_i^T \boldsymbol{R}_k \boldsymbol{e}_j, \tag{2.1}$$

where $\boldsymbol{e}_i, \boldsymbol{e}_j$ and $\boldsymbol{R}_k$ are embeddings of entities $i, j \in \mathcal{E}$ and relation $k \in \mathcal{R}$, respectively.

RESCAL. Proposed by (Nickel et al., 2011), RESCAL is parameterized by an entity matrix $\boldsymbol{E} \in \mathbb{R}^{N \times d}$ and K relation matrices $\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_K \in \mathbb{R}^{d \times d}$. RESCAL uses the standard bilinear score function described by Equation 2.1.

Given that RESCAL learns a $d \times d$ matrix for each relation in a KG, it has a considerably higher number of parameters compared to other KGE models proposed in subsequent years, some of which we describe below. This makes the process of training RESCAL more expensive than other models, and some authors report that this model may be overparameterized due to its tendency to overfit (Kotnis and Nastase, 2017). Perhaps for these reasons, RESCAL fell out of use in KGE research compared to other models. But as we show empirically in Chapter 3, RESCAL is still a very competitive model for link prediction when trained appropriately, and can even outperform many models that were more recently proposed.

DistMult. This model can be seen as a constrained variant of RESCAL. DistMult (Yang et al., 2015) is parameterized by entity matrix $\boldsymbol{E} \in \mathbb{R}^{N \times d}$ and relation matrix $\boldsymbol{R} \in \mathbb{R}^{K \times d}$, and it uses the following score function:

$$s(i, k, j) = \boldsymbol{e}_i^T \operatorname{diag}(\boldsymbol{r}_k)\, \boldsymbol{e}_j. \tag{2.2}$$

Due to the use of a diagonal matrix as the transformation applied to entity embeddings, all symmetric pairs of triples (i, k, j) and (j, k, i) all assigned the same score. This means that DistMult essentially treats all relations in a KG as symmetric, a strong assumption that does not accurately describe real-world KGs. Despite this limitation, our empirical results in Chapter 3 show that DistMult is competitive with more expressive link prediction models and can even outperform them on some datasets. We discuss the implications of this observation further in Chapter 4.

ComplEx. Another constrained bilinear model, ComplEx (Trouillon et al., 2016) is parameterized by entity matrix $\boldsymbol{E} \in \mathbb{C}^{N \times d}$ and relation matrix $\boldsymbol{R} \in \mathbb{C}^{K \times d}$. Complex has the following score function:

$$s(i, k, j) = \operatorname{Re}(\boldsymbol{e}_i^T \operatorname{diag}(\boldsymbol{r}_k)\, \boldsymbol{e}_j), \tag{2.3}$$

where $\operatorname{Re}(x)$ is the real part of $x \in \mathbb{C}$.

ComplEx was designed to be more expressive than previous models like DistMult and TransE (discussed below), while retaining the same number of parameters as these models. Despite its simplicity, ComplEx has shown to be

competitive with more involved link prediction models when trained with considerably larger embedding sizes (Lacroix et al., 2018) or when trained with additional training objectives (Chen et al., 2021c).

Analogy. This model is similar to RESCAL but with additional constraints on the relation embeddings. Like RESCAL, Analogy (Liu et al., 2017) is parameterized by entity matrix $\boldsymbol{E} \in \mathbb{R}^{N \times d}$ and K relation matrices $\boldsymbol{R}_1, \boldsymbol{R}_2, \ldots, \boldsymbol{R}_K \in \mathbb{R}^{d \times d}$, and also uses the standard bilinear score function described by Equation 2.1. However, it introduces the constraints that relation matrices be normal, i.e. $\boldsymbol{R}_k \boldsymbol{R}_k^T = \boldsymbol{R}_k^T \boldsymbol{R}_k$, and commutative, i.e. $\boldsymbol{R}_k \boldsymbol{R}_{k'} = \boldsymbol{R}_{k'} \boldsymbol{R}_k$, for every $k, k' \in \mathcal{R}$.

Liu et al. (2017) show that despite using matrices for relation embeddings, the constraints imposed on the model result in block-diagonal relation matrices that are almost diagonal. Further, the authors show that ComplEx, DistMult and HolE (Nickel et al., 2016) can all be seen as restricted forms of the Analogy model.

Translational Models

Translational models are those that use distance-based score functions. That is, they assume that the embedding of the subject entity, after applying the corresponding relation-specific transformation, should be in close proximity to the object embedding according to some distance metric in the embedding space.

TransE. TransE (Bordes et al., 2013b) is parameterized by entity matrix $\boldsymbol{E} \in \mathbb{R}^{N \times d}$ and relation matrix $\boldsymbol{R} \in \mathbb{R}^{K \times d}$. Inspired by translational properties of word embeddings (Mikolov et al., 2013a), TransE uses the following score function:

$$s(i, k, j) = -\|\boldsymbol{e}_i + \boldsymbol{r}_k - \boldsymbol{e}_j\|_p, \tag{2.4}$$

where $p \in \{1, 2\}$ is the p-norm of the resulting vector.

TransE's score function suggests that, except for the case of 1-to-1 relations, the model might have a tendency to cluster many entities in embedding space despite them not necessarily being similar (Wang et al., 2014; Yang et al., 2015). For example, the relation *city_of* should map several different cities to the same country. To increase the plausibility of observed triples for that relation, TransE may give similar representations to each of those cities. Similarly, to

better represent symmetric relations, TransE may have a tendency to reduce the size of the embedding of a symmetric relation, so that the scores of triples with that relation are indeed symmetric. We observe this last phenomenon empirically in Chapter 4.

The limitations discussed above inspired several variations of TransE, such as TransH (Wang et al., 2014) and TransR (Lin et al., 2015). We refer to Wang et al. (2017) and Ji et al. (2021) for a more detailed discussion of variations of the TransE model.

Our experimental results from later chapters show that, perhaps due to its limitations, TransE underperforms in the link prediction task compared to other models. However, we show in Chapter 6 that despite its relatively low link prediction performance, the representations learned by TransE are often more useful in downstream applications than models that perform better on the link prediction task.

RotatE. RotatE (Sun et al., 2019) is parameterized by entity matrix $\boldsymbol{E} \in \mathbb{C}^{N \times d}$ and relation matrix $\boldsymbol{R} \in \mathbb{C}^{K \times d}$. RotatE's score function is the following:

$$s(i, k, j) = -\|\boldsymbol{e}_i \circ \boldsymbol{r}_k - \boldsymbol{e}_j\|_1, \tag{2.5}$$

where $\circ$ is the Hadamard product. This model defines each relation as a rotation in complex vector space. Note that, as with TransE, we negate the scores so that larger scores rank higher.

RotatE is a more recent translational model that showed state-of-the-art performance on some benchmark datasets when released. In addition, it is not constrained by the limitations in the TransE model, which make it a more capable representative of translation-based KGE models.

Neural Models

While bilinear models can be represented as neural network models, the design of several models has been based on neural networks more generally (Socher et al., 2013; Chen et al., 2021a; Zhu et al., 2021). In the following, we discuss the most relevant neural model in this thesis.

ConvE. Dettmers et al. (2018) proposed ConvE, a model that applies a two-dimensional convolutional layer to the concatenation of the subject and relation embeddings of a triple, after reshaping each into a two-dimensional embedding. It then projects the resulting feature vector into a desired dimen-

sionality using a linear layer. ConvE's score function is as follows:

$$s(i,k,j) = f(\text{vec}(f([\boldsymbol{E}_i; \boldsymbol{R}_k] * w))\boldsymbol{W})\boldsymbol{e}_j, \tag{2.6}$$

where f is a non-linear function (originally set to ReLU), $\text{vec}(\boldsymbol{x})$ transforms $\boldsymbol{x} \in \mathbb{R}^2$ into a uni-dimensional vector, $[\boldsymbol{x}; \boldsymbol{y}]$ is the horizontal concatenation of inputs $\boldsymbol{x}$ and $\boldsymbol{y}$, $\boldsymbol{E}_i$ and $\boldsymbol{R}_k$ are the embeddings of entity i and relation k reshaped into two dimensions, $*$ is the convolution operation, w is the filter used in the convolutional layer and $\boldsymbol{W}$ the weight matrix of the linear layer.

The parameters of the convolutional and linear layers in ConvE are relation-independent, meaning that the model learns a parameterized function that predicts the object of a triple given its subject and relation. This is different from the previously described models in that score functions are usually able to predict either the missing subject or object of a triple, given the other to components. To address this limitation, ConvE represents the *object prediction* task with the query $(j, k^{-1}, ?)$ and learns two different embeddings for each relation k, one for k and one for k^{-1}. Thus, the model is parameterized by the filter w of its two-dimensional convolutional layer, weight matrix $\boldsymbol{W}$ of the linear layer, entity matrix $\boldsymbol{E} \in \mathbb{R}^{N \times d}$ and relation matrix $\boldsymbol{R} \in \mathbb{R}^{K' \times d}$, where $K' = K \times 2$. Note that its dependency on the use of reciprocal relations means it is unclear how to use ConvE for tasks where the direction of the relation is not specified, e.g. triple classification. We discuss this limitation further in Chapter 3.

Theoretical Expressivity of KGE Models

Several studies have looked into the theoretical expressivity and limitations of different KGE models (Liu et al., 2017; Wang et al., 2018b; Abboud et al., 2020). In fact, the design of new KGE models is often inspired by the theoretical limitations of previous models (Wang et al., 2014; Lin et al., 2015; Liu et al., 2017; Sun et al., 2019; Abboud et al., 2020). However, we note that despite being designed to overcome the theoretical limitations of previous models, most such studies test their proposed models on generally the same experimental settings. That is, on the same link prediction task and using the same benchmark datasets. Thus, except for a few studies (Trouillon et al., 2016, 2019), the impact of the theoretical modeling abilities of most models is often not tested in practice.

In addition, as we see in later chapters, the theoretical differences between some models are often not apparent when looking at their empirical performance, especially when comparing different models on fair experimental settings. For example, we see in Chapters 3 and 4 that DistMult, a theoretically restricted model, is competitive with the more expressive ComplEx model on link prediction in most datasets, even outperforming it at times. Similarly, and as mentioned before, we see in Chapter 6 that the representations learned by TransE are often more useful in downstream applications compared to those of less restricted models, such as ComplEx or RotatE.

For the reasons stated above, in this thesis we focus on the empirical performance of models on benchmark datasets, and do not discuss the theoretical differences between KGE models further, except where relevant for discussing experimental results.

KGE Model Extensions

In this thesis, we propose new training and evaluation methods that generally apply to KGE models. Therefore, as a first step, the empirical studies presented in later chapters focus exclusively on *pure* KGE models, i.e. those that only make use of known facts in a KG to learn representations about its entities and relations. However, we note that in addition to pure KGE models, several models have been designed to use additional information about entities and relation during training and/or at inference time. While the results presented in later chapters are insightful for pure KGE models, future studies may extend the work in this thesis to include KGE models that use additional information. In principle, all of our proposed methods are model agnostic, but some models may require some adaptations for these methods to work. For reference, we cite a few of those models in the following.

Entity types. Some models make use of information about entity types, e.g. as constrains in the embedding space (Guo et al., 2015) or to generate negative samples during training (Kotnis and Nastase, 2018).

Entity descriptions. Other models make use of the textual descriptions often available for entities in a KG. Such information may be used, for example, by learning KG and word embeddings simultaneously and aligning both embedding spaces with corresponding anchors (Wang et al., 2014), or as additional training objectives (Xiao et al., 2017).

Schema restrictions. Several properties of relations in a KG can be expressed as logical rules, e.g. $\forall x, y : \text{hasWife}(x, y) \implies \text{hasSpouse}(x, y)$. Such information can be used, e.g. as additional constraints in the embedding space (Guo et al., 2016; Rocktäschel et al., 2015).

Temporal information. Several studies have focused on the temporal nature of facts in a KG, e.g. a country's president changes every few years. Such information has been used in previous studies to learn temporally-aware models (Dasgupta et al., 2018; García-Durán et al., 2018; Goel et al., 2020).

2.2.6 Evaluation

We introduce in this section the triple classification task and its corresponding form of evaluation. We then discuss in detail the most common form of KGE evaluation, called *entity ranking*. In Chapters 4 and 5, we propose two alternative methods of evaluation, including a generalization of the entity ranking protocol introduced below.

Triple classification (TC). The goal of triple classification is to test the model's ability to discriminate between true and false triples (Socher et al., 2013). We define the task as follows:

Definition 2.2.3 (Triple Classification)**.** Given a knowledge graph $\mathcal{K} \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, triple classification is the task of predicting whether a given triple (i, k, j) is true or false, i.e. answering the query (i, k, j)?.

To evaluate a model on this task, a held-out set of evaluation triples $\mathcal{T}$ is used. Since only true triples are available in practice, $|\mathcal{T}|$ negative triples are generated by randomly replacing either the subject or the object of each evaluation triple by a randomly sampled entity (we discuss the generation of negative examples in more detail below). All triples are then classified as positive or negative according to the KGE scores. Model performance is then typically assessed by classification accuracy. A triple (i, k, j) is classified as positive if its score $s(i, k, j)$ exceeds a relation-specific decision threshold τ_k that is learned by maximizing classification accuracy on a different held-out set of triples created with the same procedure.

Model performance on triple classification suggests that the task is generally easy, with results that can be overly optimistic and misleading (Safavi and Koutra, 2020). This is likely because for each evaluation triple, a single

negative example is created, and because uniform sampling is used to generate these negatives from a high number of possible candidates, it is more likely that we obtain a low scored triple, rendering most classification queries "easy". To address this issue, Safavi and Koutra (2020) proposed a dataset with manually labeled negatives constructed to be "hard" (we discuss the creation of this dataset in Section 2.4). They found that triple classification with hard negatives is indeed a relatively more difficult task. But despite these efforts, and perhaps due to the generally optimistic results of this evaluation, triple classification is not nearly as commonly used as the entity ranking evaluation protocol. We thus do not consider it in the studies in this thesis.

Entity Ranking. Most of the studies in the KGE literature have measured KGE performance using the entity ranking evaluation protocol (Ji et al., 2021; Wang et al., 2017), described by Algorithm 2.1. This protocol is based on the link prediction task, as it evaluates a model's ability to answer questions $(i, k, ?)$ and $(?, k, j)$ given a held-out set of (i, k, j) evaluation triples. Specifically, given query $(i, k, ?)$, models score all triples (i, k, j') where j' is every possible entity in $\mathcal{E}$. The resulting $|\mathcal{E}|$ candidate triples are ranked based on their score, with the expectation that the known correct answer to the query, i.e. triple (i, k, j), would rank high on the list of candidates. This process is performed for every evaluation triple to obtain as many rankings as there are evaluation triples. The same approach is followed for query $(?, k, j)$, after which all obtained rankings for both types of queries are aggregated via the micro-average of some metric, such as mean reciprocal rank or Hits@10, which we formally define below.

Local-closed world assumption. The evaluation process described above implies that there is a single correct answer to every evaluation query. While this is true for functional relations, such as *livesIn*, it is not true for relations with more than one possibly correct answer, such as *livedIn*. This is known as the local-closed world assumption (LCWA) (Galárraga et al., 2013), which in contrast to the closed world assumption (see Section 2.1), states that for every observed triple (i, k, j), every unobserved triple (i, k, j') is assumed to be false. This assumption is often violated in different ways. First, and as mentioned before, a known correct answer may be one of many correct answers, all of which a good model should rank high. Second, unless the KG is complete, there are unknown true triples that a model that generalized well should be able to rank high. This assumption penalizes such a model. Finally,

Algorithm 2.1: Entity Ranking (ER) Evaluation Protocol

Require: $\mathcal{T}$: set of evaluation triples,
$\mathcal{E}$: set of entities in knowledge graph $\mathcal{K}$
Ensure: Aggregated metric m_{all} over all evaluation triples

$R_{\text{all}} \leftarrow [\,]$ // ranks of target answers
foreach $(i,k,j) \in \mathcal{T}$ **do**
  // object prediction
  $\mathcal{N}_o \leftarrow$ construct $|\mathcal{E}|-1$ candidates (i,k,j') with every $j' \neq j \in \mathcal{E}$
  $S_o \leftarrow \textsc{Compute\_Scores}(\mathcal{N}_o \cup \{(i,k,j)\})$
  $r_o \leftarrow \textsc{Compute\_Ranks}(S_o, j)$
  $R_{\text{all}} \leftarrow \textsc{Append}(R_{\text{all}}, r_o)$ // add rank of j
  // subject prediction
  $\mathcal{N}_s \leftarrow$ construct $|\mathcal{E}|-1$ candidates (i',k,j) with every $i' \neq i \in \mathcal{E}$
  $S_s \leftarrow \textsc{Compute\_Scores}(\mathcal{N}_s \cup \{(i,k,j)\})$
  $r_s \leftarrow \textsc{Compute\_Ranks}(S_s, i)$
  $R_{\text{all}} \leftarrow \textsc{Append}(R_{\text{all}}, r_s)$ // add rank of i
$m_{\text{all}} \leftarrow \textsc{Compute\_Metrics}(R_{\text{all}})$

this process may generate triples that are known to be true, e.g. because they are in the training, validation or test sets. This last factor is partially addressed by the use of filtered metrics, described below. In addition, we show in Chapter 4 that it is possible to use known schema constraints to check whether a candidate triple is actually negative. In general, however, reliance on LCWA may result in negative examples that are unknown positive triples, For this reason, we often refer to such generated triples as *pseudo-negatives*.

Evaluation metrics. We define the evaluation metrics used throughout this thesis, which are the two most commonly used metrics in the literature: mean reciprocal rank (MRR) and Hits@K. Given a triple (i,k,j), denote by $\text{rank}(j|i,k)$ the rank of object j given subject and relation tuple (i,k), i.e., the rank of model score $s(i,k,j)$ among the scores of a set of pseudo-negative triples. The set of pseudo-negatives used to compute evaluation metrics determines whether the metrics are *raw* or *filtered*. For raw metrics, this set is defined as $\{s(i,k,j') : j' \in \mathcal{E} \wedge j' \neq j\}$, as in Algorithm 2.1. To avoid underestimation of a model that ranks training triples higher than the expected answer, filtered metrics further add the condition that all candidate triples do not occur in the training, validation or test splits. We follow common practice in the recent literature and only report filtered metrics in

this thesis.

If there are ties, we take the mean rank of all triples with score $s(i,k,j)$. Define $\text{rank}(i|k,j)$ likewise. Denote by $\mathcal{K}^{\text{eval}}$ the set of evaluation triples. Then

$$\text{MRR} = \frac{1}{2|\mathcal{K}^{\text{eval}}|} \sum_{(i,k,j)\in\mathcal{K}^{\text{eval}}} \left(\frac{1}{\text{rank}(i|k,j)} + \frac{1}{\text{rank}(j|i,k)} \right), \tag{2.7}$$

$$\text{Hits@}K = \frac{1}{2|\mathcal{K}^{\text{eval}}|} \sum_{(i,k,j)\in\mathcal{K}^{\text{eval}}} \Big(\mathbb{1}(\text{rank}(i|k,j) \leq K) + \mathbb{1}(\text{rank}(j|i,k) \leq K) \Big), \tag{2.8}$$

where indicator $\mathbb{1}(C)$ is 1 if condition C is true, else 0.

Mean rank (MR) is another metric that is sometimes used in the literature. It is defined as the average rank of the expected answer in all evaluation triples. This metric has seldom been used in recent work, as its sensitivity to outliers is what inspired the widespread use of MRR.

As part of our extensive study on link prediction performance in Chapter 3, we discuss and provide empirical results about variations of the entity ranking protocol, such as using other forms of handling ties and the impact that using entities not seen during training has on reported metrics. In addition, we discuss limitations of this protocol at length in Chapter 4.

2.2.7 Training

In this section, we introduce the most commonly used training method for KGE models in the literature, including the standard approach for generating negative examples. We discuss more details about this training method, such as commonly used loss functions, different regularization approaches and other related training approaches, in Chapter 3 as part of our large comparative study on link prediction performance. In addition, we introduce in Chapter 5 a generalization of this training method.

All methods for training KGE models require negative examples to avoid solutions that do not generalize well (Nickel et al., 2015). However, knowledge graphs are almost always constructed using only positive examples, i.e. observed triples. Thus, there is a need to generate negative examples during training. There are different types of training methods for KGE models,

Algorithm 2.2: Negative Sampling Training

Require: $\mathcal{T}$: set of training triples,
$\mathcal{E}$: set of entities in knowledge graph $\mathcal{K}$
θ: model parameters,
n: number of negatives per positive (hyperparameter)
Ensure: Updated model parameters θ

$\mathcal{T}_{\text{all}} \leftarrow \mathcal{T}$
foreach $(i, k, j) \in \mathcal{T}$ **do**
  // object prediction
  $C_o \leftarrow \textsc{Sample}(\mathcal{E}, n)$ // sample n entities
  $N_o \leftarrow$ construct negatives (i, k, j') with every $j' \in C_o$ // corrupt j
  // subject prediction
  $C_s \leftarrow \textsc{Sample}(\mathcal{E}, n)$
  $N_s \leftarrow$ construct negatives (i', k, j) with every $i' \in C_s$ // corrupt i
  $\mathcal{T}_{\text{all}} \leftarrow \mathcal{T}_{\text{all}} \cup N_o \cup N_s$
$S_{\text{all}} \leftarrow \textsc{Compute\_Scores}(\mathcal{T}_{\text{all}})$
$L_{\text{all}} \leftarrow \textsc{Compute\_Loss}(S_{\text{all}}, \mathcal{T}_{\text{all}})$
$\theta \leftarrow \textsc{Update\_Parameters}(\theta, L_{\text{all}})$

which mostly differ in their use of negative examples. We describe the most common training type in the following.

Negative Sampling. The most common approach for training KGE models is called negative sampling (*NegSamp*) (Bordes et al., 2013b) and it is described by Algorithm 2.2. For every training triple (i, k, j), n negative triples are generated by *corrupting* the object of that triple. That is, by replacing it with a randomly selected entity $j' \in \mathcal{E}$ n times to form n triples (i, k, j'). These generated negatives are used as additional training triples. Similarly, the subject of every training triple is corrupted n times to create n additional negative triples. These generated triples are labeled as negatives and added to the training set. In this type of training, the number n of negatives generated from each training triple is a hyperparameter.

In general, an approach that relies on corrupting either the subject or object of a given positive triple can generate at most $|\mathcal{E}|$ distinct negative examples per corrupted slot. This choice of the number of distinct negative examples that are generated for each positive triple is what partially distinguishes NegSamp from other *training types* that have been proposed to train

KGE models. We discuss those in detail in Chapter 3.

Note that the standard approach for generating negatives follows the LCWA, and consequently suffers from the same issues discussed in the context of the entity ranking protocol in Section 2.2.6. Namely, that a training triple may have more than one correct answer, and that this process may produce false negatives. The former motivated the design of the *KvsAll* training approach discussed in Section 3.1, while the effect of the latter can be mildened by checking whether the generated negatives are part of a set of known positives, e.g. the training set. This set membership operation has a significant cost in the training process, so it is often not used in practice. Another approach to minimizing the probability of generating false negatives relies on using additional information, such as schema constraints, to generate negatives that are less likely to be positives (Kotnis and Nastase, 2018). In general, the process of generating negatives for KGE training is known to have an impact on model performance, and thus has been the focus of previous studies, e.g. Kotnis and Nastase (2018). We also study this to an extent in Chapter 3, where we look at the impact on model performance that results from the choice of different training approaches, as well the choice of n when training with NegSamp.

Finally, note that the generation of negative examples by corrupting the subject of a triple is related to the query $(?, p, o)$, as we are training models to identify correct and incorrect answers to that question. Similarly, when corrupting the object of a triple, we train models to answer the question $(s, p, ?)$. Thus, the standard method for training KGE models is designed to train a model to perform link prediction. We discuss the implications of this in Chapters 5 and 6, where we propose a generalized form to train KGE models on multiple tasks simultaneously, and check the impact that using different training approaches have in downstream applications, respectively.

2.2.8 Limitations and Relevance of KGEs

KGE models are limited in many respects compared to other families of models. In this section, we briefly discuss some of these limitations. We then briefly argue why research into KGEs is nevertheless relevant.

Transductivity. One of the most important limitations of KGE models is their inability to perform inductive predictions, i.e. predictions about unseen KG components. Pure KGE models are transductive, meaning that they can

only make predictions about entities and relations they have seen during training. Whenever a KG is expanded with new entities and/or relations, one alternative for an existing KGE model that was trained on this KG is to be retrained on the entire graph. This is a severe limitation, as retraining a model from scratch is especially limiting when considering that KGEs are often intended for large-scale real-world knowledge graphs, where training costs are significantly high (Zheng et al., 2020; Lerer et al., 2019).

Another alternative is for either the model or the training process to incorporate a mechanism that allows for the addition of new components of the KG to an existing model. Indeed, some extensions of KGE models have such mechanisms. For example, Wang et al. (2021) developed a KGE model that uses a text encoder to embed entities based on textual descriptions. This model can thus make predictions about unseen entities given a textual description of them. This limitation of KGEs is well-known and has been the focus of several studies in the past, e.g. Jambor et al. (2021); Albooyeh et al. (2020); Zhao et al. (2017); Xie et al. (2016).

Complex query answering. Pure KGE models are designed to provide scores for given triples. Thus, they are commonly used for simple 1-hop predictions over a graph, e.g. the link prediction task. While we show in Chapter 5 that KGEs can be extended and trained to make a wider type of predictions, these too are not as complex compared to other models that can answer, e.g. conjunctive queries with missing entities (Kotnis et al., 2021). Rule-based models can also perform predictions about complex queries, so long as these are supported by their language bias (see Section 2.3.3).

Several studies in the past have focused on extending KGEs or using them in combination with other models to make complex predictions over KGs. For example, Li et al. (2022) focus on multi-hop question answering, and Jia et al. (2021) focus on complex predictions over temporal knowledge graphs.

Training runtime. KGE models have a considerably higher training cost compared to some types of models. For example, some rule-based models can provide competitive results in link prediction after only training for a few seconds (Meilicke et al., 2019) (more in Section 2.3.3). The high cost of training KGE models, in combination with the large scale of real-world KGs, has inspired several engineering efforts to make these costs manageable, e.g. the works of Mohoney et al. (2021) and Lerer et al. (2019).

The high cost of training is not a problem that is exclusive to KGEs, with

more recent link prediction models based on transformers (Chen et al., 2021a) or on graph convolutional neural networks (Zhu et al., 2021) being even more costly to train. While some of these models do provide some benefits compared to KGE models, they also come with some disadvantages. We discuss some of these models further in Section 2.3.1.

Interpretability. KGE models, like many models based on learning representations, are not interpretable in the sense that in the best case, there is no straightforward way to derive explanations for the predictions they make. It is often the case that nothing can be said about what determines the decisions made by a model. This is in contrast with rule-based models, which can provide explanations for their predictions (Galárraga et al., 2013; Meilicke et al., 2019).

A few studies in the past have focused on interpretability of KGE models, e.g. Zhang et al. (2019a) and Xie et al. (2017), but this research area is not as active as the ones described above, as there is less relevant work in recent years.

Relevance of KGE models. We argue that, despite the limitations described above, KGE models are a relevant family of models for the following reason. KGEs are learned representations of KGs, which in turn are a form of encoding *structured* data. As such, KGs have various uses both in industry and academia (see Section 2.1.3). In addition, and as discussed in Section 2.2.4, KGEs have potential advantages over KGs, such as a reduced cost of inference or the ability to be used as distributed representations of knowledge in downstream applications. Thus, so long as KGs are useful, KGEs can too be useful, as long as they are faithful representations of KGs (an important question in this thesis).

Recently, a long line of research into language modeling (Mikolov et al., 2013a; Devlin et al., 2019; Radford et al., 2018) has produced models referred to as large language models (LLMs) (Touvron et al., 2023; Ouyang et al., 2022). These models, which are based on distributed representations, have shown strong performance on several types of tasks that require various forms of reasoning (Wei et al., 2022). Thus, these models have challenged the intuition that structured data is needed to encode knowledge that can be reasoned over. We maintain that while this may be true in the long term, this is at the moment not the case. LLMs are currently very unreliable sources of knowledge (Sun et al., 2023). In addition, they are very expensive to train, so that incorporating new knowledge into these models remains a significant challenge (Bubeck

et al., 2023). Finally, it may be possible to combine the knowledge in KGs by either combining them, or their distributed representations, with LLMs. Previous works have already studied the incorporation of KGE models into language modeling (Wang et al., 2021; He et al., 2020; Zhang et al., 2019b).

2.3 Related Models

There are other families of models that relate to KGE models because they can perform link prediction, but either do not learn representations of the entities and relations in a KG, or because they learn such representations but are not designed to make use of a score function to make fact predictions. We use some of these models as baselines in later chapters, such as rule-based models or models based on graph neural networks, so we discuss these two families of models in detail in Sections 2.3.3 and 2.3.4, respectively. For reference, we discuss other related models in the following sections.

2.3.1 Link Prediction Models

Some models are designed exclusively for the link prediction task in KGs. Such models are often more involved than pure KGE models, which results in more costly training and/or inference times. In addition, these models often do not provide representations of the entities or relations in a KG, at least not in a straightforward manner.

An example of a link prediction model is the HittEr model (Chen et al., 2021a). It is based on the transformer architecture and is designed to use contextual representations of entities to perform link prediction. At the time of its publication, it achieved state-of-the-art results in some link prediction benchmarks, but it has since been shown that ComplEx achieves similar results with larger embedding sizes and additional training objectives (Chen et al., 2021c).

A more recent link prediction model is the NBFNet model (Zhu et al., 2021). This model is based on graph convolutional networks (GCN) and achieved state-of-the-art results in some benchmark datasets. However, it does not directly learn representations of entities in a KG, so that it is unclear how it can be used as a model that learns representations of a KG. It addition, its training cost is much higher than that of pure KGEs, so that it is unclear

to what extent it can scale to real-world KGs.

2.3.2 Feature-Based Models

In contrast to embedding models, which exploit latent features in a graph, some models exploit observed features in the graph to either learn representations of the graph, or perform some task such as link prediction. For example, the path-ranking algorithm or PRA (Lao and Cohen, 2010; Gardner et al., 2014) was designed for predicting missing links in knowledge graphs from features constructed by performing random walks on the graph. Another example is the RDF2Vec model (Ristoski and Paulheim, 2016). It performs random walks to extract paths from a KG, which are then embedded with the *word2vec* approach (Mikolov et al., 2013a) to obtain vector representations of entities in a KG.

While the RDF2Vec model is more closely related to KGEs in that it provides representations that can be used in downstream applications, the performance of such feature-based models is seldom compared with KGEs models. An exception are rule-based models, which have been shown to perform competitively with KGEs on link prediction (Meilicke et al., 2018, 2019). In addition, several models based on graph neural networks have been recently developed that achieve state-of-the-art performance on link prediction and other tasks that can also be performed with KGE models (Yu et al., 2021a; Zhu et al., 2021). Thus, we use these two types of models as baselines in later chapters, and introduce them in more detail in the following sections.

2.3.3 Rule-Based Models

In this section, we introduce rule-based models in a way that describes the models that are relevant for this thesis. For a more comprehensive introduction, see De Raedt (2008).

We refer to rule-based models as models that learn logical rules that explicitly encode patterns in a knowledge graph (Lao and Cohen, 2010; Galárraga et al., 2013; Meilicke et al., 2018, 2019). For example, given knowledge graph $\mathcal{K}$, such a model might learn the rule $capitalOf(X, Y) \rightarrow locatedIn(X, Y)$. Thus, rule-based models learn to represent a KG as a set of logical rules. The types of rules that a model can learn is referred to as the *language bias* of the

model. For example, models may learn *path* rules of the form

$$p_1(X, Z) \leftarrow p_2(X, Y_1) \wedge p_3(Y_1, Y_2) \wedge \ldots \wedge p_n(Y_{n-1}, Z), \tag{2.9}$$

where $p_i \in \mathcal{R}$ is a (binary) predicate, X, Y_i, Z are variables, $p_1(\ldots)$ is the head of the rule, and $p_2(\ldots) \wedge \ldots \wedge p_n(\ldots)$ is the body of the rule. Entities in a KG are constants that may *ground* rules, resulting in new predicted triples in the KG. For example, given the rule $locatedIn(X, Y) \leftarrow capitalOf(X, Y)$, the grounding $X = Austin$ and $Y = Texas$ would result in the prediction of the triple $locatedIn(Austin, Texas)$. Note that a triple always corresponds to a grounded binary predicate.

Rules are often learned along with a confidence score that reflects, based on the patterns observed about $\mathcal{K}$ during learning, how likely is a rule to make a correct prediction. These confidence scores are used to perform link prediction as follows. Given an incomplete query $p(a, ?)$, we select all rules with predicate p in the head. For each rule, we look for groundings of their body in $\mathcal{K}$ where constant a replaces the corresponding variable in the body. Each rule with a matching body then predicts a candidate for the missing slot, which we associate with the confidence score of the rule that made this prediction. All predictions are then ranked by their scores to produce a final ranking of candidates, as done with KGE models. Then, the same evaluation protocols for link prediction may be used, e.g. entity ranking (see Section 2.2.6). In practice, the process is more involved, as a single candidate is often predicted by more than one rule, and given that there are often dependencies between different rules, e.g. if one rule fires, a more general form of the same rule also fires, it is not trivial to determine how to aggregate scores from different rules to assign a single score to each candidate.

In the following, we describe two rule-based models in more detail: RuleN (Meilicke et al., 2018), which is used as baseline in Chapter 4, and AnyBURL (Meilicke et al., 2019), which is used as baseline in Chapter 3.

RuleN. Proposed by Meilicke et al. (2018), RuleN was designed to be a simple model for link prediction on knowledge graphs. When released, it achieved competitive results on benchmark datasets.

In addition to path rules, such as those described above, RuleN learns *constant* rules. Concretely, let X_i and Y refer to variables that are quantified over $\mathcal{E}$, $k, p_i \in \mathcal{R}$ and let a be a constant entity. RuleN learns path rules (P_n)

and constant rules (C) defined as follows:

$$k(X_1, X_{n+1}) \leftarrow p_1(X_1, X_2) \wedge \ldots \wedge p_n(X_n, X_{n+1}) \quad (P_n)$$
$$k(X, a) \leftarrow \exists Y\ k(X, Y). \quad (C)$$

The rule $practicesSport(X, \text{football}) \leftarrow \exists Y\ practicesSport(X, Y)$ is an example of a type C rule that might be learned in a KG where a considerable number of people practice football.

In the original work, the maximum allowed length n of path rules is set to be small for simplicity and efficiency. To further reduce costs, RuleN does not use the entire training set when searching for groundings of the body of a rule. Instead, it samples triples during this process. For details about the learning and application of rules, as well as confidence scores used for link prediction, see Meilicke et al. (2018).

AnyBURL. In contrast to RuleN, AnyBURL (Meilicke et al., 2019) learns rules in a bottom-up approach. Specifically, it samples paths from the graph and treats them as "bottom" path rules, i.e. specific path rules that can be generalized to represent a more commonly observed pattern in the graph. The authors propose that any useful generalization of a path rule of length n can be described by the following rules:

$$k(X, Y) \leftarrow p_1(X, A_2) \wedge \ldots \wedge p_n(A_n, Y) \quad (P_n)$$
$$k(c_0, X) \leftarrow p_1(X, A_2) \wedge \ldots \wedge p_n(A_n, c_{n+1}) \quad (AC_1)$$
$$k(c_0, X) \leftarrow p_1(X, A_2) \wedge \ldots \wedge p_n(A_n, A_{n+1}), \quad (AC_2)$$

where $c_i \in \mathcal{E}$ are constants and X, Y, A_i are variables. Thus, AnyBURL focuses exclusively on learning path rules.

Similar to RuleN, AnyBURL learns P_n rules, but extends its language bias by learning AC_i rules, which generalize the C rules learned by RuleN by allowing: (i) paths of increasing length and (ii) the use of predicates in the body that are not present in the head. During training, AnyBURL samples paths of increasing length iteratively up to a user-specified time limit. At the time of publication, AnyBURL achieved competitive results on benchmark datasets for link prediction, sometimes even when training for only 10 seconds. This highlights one of the main advantages of rule-based models over KGE models, which we discuss in the following.

Advantages over KGEs. As mentioned in Section 2.2.4, KGEs are faster

than rule-based models at inference time, as seen by the runtimes reported by Meilicke et al. (2019). However, compared to KGEs, rule-based models have some advantages as link prediction models. First, they require a considerably shorter training time, mostly due to the hyperparameter optimization required by KGE models. In addition, rule-based models can provide explanations for the predictions they make (Meilicke et al., 2019), which makes them interpretable.

Due to the advantages of rule-based models described in this section, and the advantages of KGE models described in Section 2.2.4, some works have studied methods to combine the predictions of both types of models to improve performance on link prediction. For example, Meilicke et al. (2018) used standard ensembling approaches on the predictions of both types of models, while Meilicke et al. (2021) proposed a heuristic to combine the predictions of AnyBURL with those of KGE models, based on observations about which types of models are better at which types of relations in the graph.

In this thesis, we show in Chapter 3 that AnyBURL performs competitively with KGE models on link prediction, sometimes after learning rules for only a small fraction of the time it takes to train a single KGE model during hyperparameter optimization. In addition, we show in Chapter 4 that RuleN considerably outperforms KGE models on knowledge base completion, a task we introduce as a more general form of link prediction. All of this makes rule-based approaches a viable alternative to link-prediction and similar applications on knowledge graphs.

2.3.4 Graph Convolutional Neural Networks

In Chapter 6, we conduct a large experimental study on the usability of KGE models in node-level downstream applications, such as classification (e.g. predicting a person's profession) or regression (e.g. predicting a person's age). Since graph convolutional neural networks (GCNs) are a different family of models aimed at addressing such tasks, we use a state-of-the-art GCN as baseline in our experiments on downstream applications. In this section, we briefly introduce spatial-based GCNs from the perspective of how they model graph data. We then describe the baseline model we use in Chapter 6. For a comprehensive introduction to graph neural networks, see Wu et al. (2020).

Graph convolutional neural networks. These models were developed

as generalizations of convolutional neural networks that can be applied to graph data. Their main purpose is to address graph-related tasks, such as node classification, graph classification or link prediction. As such, their applications sometimes overlap with the goal of KGE models. However, while there are some similarities between the two families of models, there are also some important differences in the way they model graph data to solve a given task.

GCNs are designed to learn node (and sometimes edge) representations by iteratively applying a function that updates the representation of each node based on the transformed and aggregated representations of the nodes in its 1-hop neighborhood. Specifically, let $\boldsymbol{h}_i^{(t)}$ be the representation of node i in iteration t. We have:

$$\boldsymbol{h}_i^{(t)} = U^{(t)} \left(\sum_{j \in N(i)} M^{(t)}(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)}) \right), \tag{2.10}$$

where $N(i)$ is the 1-hop neighborhood of node i, $M^{(t)}$ is a parameterized *message passing* function for layer t and $U^{(t)}$ is a (possibly) parameterized *update* function for layer (t). M is described as a message passing function because it combines the representation of a given node with each of its immediate neighbors. It also often includes node features $\boldsymbol{x}_i, \boldsymbol{x}_j$ and edge features $\boldsymbol{x}^e_{(i,j)}$ in its computation, which we omit for brevity. The update function U transforms the aggregated computations from function M. While U can be a parameterized transformation, it is often used as an activation function, e.g. ReLU. Note that the aggregation function in Equation 2.10 is the sum, but this can be generalized to any (possibly parameterized) aggregation function. Different choices of M, U and the aggregation function result in different types of GCNs.

Layers that apply over all nodes in a graph, such as the one described in Equation 2.10, can be stacked to increase the size of the neighborhood over which information is passed from node to node. This is known as the propagation step. Once all GCN layers have been applied, the resulting node representations can be projected down to a desired size and used as input features for node-level tasks, or passed to a function that aggregates them to get a representation of the entire graph for graph-level tasks. These representations can also be used as input for KGE models to perform link prediction. This was first tested experimentally when relational graph convolutional

neural networks were proposed, which we introduce in the following.

Relational GCNs. Schlichtkrull et al. (2018) proposed a generalization of GCNs that applied to knowledge graphs called relational graph convolutional neural networks (R-GCNs). To this end, they proposed the use of relation-specific message passing functions, as well as relation specific aggregation functions. Specifically, using a sum as aggregation function, we have:

$$\boldsymbol{h}_i^{(t)} = U^{(t)}\left(\sum_{k\in\mathcal{R}}\sum_{j\in N_k(i)} M_k^{(t)}(\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)})\right), \tag{2.11}$$

where $N_k(i)$ is the 1-hop neighborhood of all nodes connected to node i via relation k, and M_k is the message passing function for relation k. Note that before aggregating over all types of relations in the KG, we aggregate the output of M_k over $N_k(i)$. While different choices of the aggregation function, the M function, and the U function, lead to different relational GCNs, the authors proposed a specific instance where $M_k^{(t)} = \lambda \boldsymbol{W}_k^{(t)} \boldsymbol{h}_j^{(t-1)}$, with λ acting as a normalization constant set to $1/|N_k(i)|$. As part of the overall aggregation, they also included a transformation $\boldsymbol{W}_0$ applied between $\boldsymbol{h}_i^{(t)}$ and $\boldsymbol{h}_i^{(t-1)}$, i.e. between a node and its representation in the previous layer.

In addition to R-GCNs, Schlichtkrull et al. (2018) proposed an encoder-decoder architecture for link prediction, where any R-GCN model could act as an encoder that learns entity representations, and any KGE score function could act as a decoder that uses the representations provided by the encoder. In their experiments, they combined their proposed R-GCN model with DistMult's score function and showed that their proposed architecture provided improved performance compared to standard DistMult. However, it has since been shown with more rigorous experiments that using such encoders in combinations with KGEs for link prediction provides little benefit, and that any observed improvement comes from the additional transformations that the encoder applies to node representations, and not from its modeling of the structure of the graph (Zhang et al., 2022).

KE-GCN. Under the name of KE-GCN, Yu et al. (2021a) proposed a general framework for relational GCNs that uses score functions for given triples as part of the relation-specific message passing functions. Specifically,

they proposed the following propagation function:

$$\boldsymbol{h}_i^{(t)} = U^{(t)}\left(\sum_{k\in\mathcal{R}}\sum_{j\in N_k(i)} \boldsymbol{W}_k^{(t)}\frac{\partial s^{(t-1)}(j,k,i)}{\partial \boldsymbol{h}_i^{(t-1)}} + \boldsymbol{W}_0^{(t)}\boldsymbol{h}_i^{(t-1)}\right), \tag{2.12}$$

where $s^{(t-1)}(i,k,j)$ is a score function that uses embeddings $\boldsymbol{h}_i^{(t-1)}, \boldsymbol{h}_j^{(t-1)}$ and $\boldsymbol{r}_k^{(t-1)}$. Note their use of relation embeddings. Indeed, they also propagate information to iteratively update the relation embeddings as follows:

$$\boldsymbol{r}_k^{(t)} = U^{(t)}\left[\boldsymbol{W}_{rel}^{(t-1)}\left(\sum_{(i,j)\in N(k)}\frac{\partial s^{(t-1)}(i,k,j)}{\partial \boldsymbol{r}_k^{(t-1)}} + \boldsymbol{r}_k^{(t-1)}\right)\right], \tag{2.13}$$

where $N(k)$ is the set of entity tuples that interact with relation k and $\boldsymbol{W}_{rel}$ is a transformation applied exclusively to relation embeddings.

Yu et al. (2021a) showed that their framework subsumes existing GCNs, such as CompGCN (Vashishth et al., 2020) and W-GCN (Shang et al., 2019). For example, the model from Schlichtkrull et al. (2018) described above can be obtained with score function $s(i,k,j) = \boldsymbol{h}_i^T\boldsymbol{h}_j$ and $\boldsymbol{r}_k = \boldsymbol{0}$, i.e. setting all relation embeddings to zero. In their experiments, they used KGE score functions such as TransE's or RotatE's.

In Chapter 6, we compare the performance of the KE-GCN model proposed by Yu et al. (2021a) with simple downstream models that take representations from pre-trained KGE models as input to perform node classification or node regression tasks. We find that, despite extensive hyperparameter optimization for both families of models, KGE models outperform KE-GCN most of the time, despite the latter training directly on the downstream task.

Comparison to KGE models. There are two aspects of GCNs worth discussing in the context of KGE models. First, GCN-based models are often designed for specific tasks, e.g. node classification tasks (Yu et al., 2021a; Vashishth et al., 2020), node regression tasks (Huang et al., 2021), or graph-level tasks (Yu et al., 2021a). In contrast, while KGEs have more recently been studied as link prediction models, they are not only motivated by, but also used as, learned representations of KGs (El-Kishky et al., 2022; Ilyas et al., 2022). In contrast, we are unaware of GCN-based models that are used as pre-trained representations of KGs in downstream application scenarios.

The fact that GCN-based models are not used as pre-trained representations of KGs may relate to the second important difference between KGEs and GCNs: the cost of training. The propagation step in GCNs can be prohibitively expensive with large graphs and deep networks. Thus, it is known that, at least compared to KGE models, GCNs are much more expensive to train (Zhang et al., 2022). Despite this cost, GCN-based models can outperform KGE models on some tasks. For example, the NBFNet model (Zhu et al., 2021) outperforms KGE models in link prediction on some commonly used benchmark datasets. In addition, this model is designed to perform inductive predictions, which is something that pure KGEs cannot do (see Section 2.2.8).

2.4 Benchmark Datasets

In this section, we introduce the benchmark datasets used in the experiments presented in later chapters. These have been the most commonly used benchmarks datasets in KGE research for many years. In addition, we briefly discuss other KGE datasets that are used in the literature for different purposes. The statistics of all benchmark datasets introduced in this section are described in Table 2.3.

FB15K. Introduced by Bordes et al. (2013b), FB15K is a subset of the Freebase knowledge graph introduced in Section 2.1.4. It was constructed by selecting the subset of entities that had at least 100 mentions in the entire KG, and whose corresponding Wikipedia page was linked by at least one website in Google's web index at the time of creation. This information was available in the Wikilinks database, which is no longer active.[1] For the set of relations, they chose those that interacted with this subset of entities, so long as they too had at least 100 mentions in Freebase. From the resulting set of relations, they removed those that were explicitly marked as being inverse relations of other ones present in this set.

WN18. This dataset was created by Bordes et al. (2013a). It is a subset of the WordNet knowledge graph introduced in Section 2.1.4. Without specified criteria, they selected 18 relations from the WordNet KG as relations for this dataset, and used all entities that interacted at least 15 times with this set of relations.

FB15K-237. Toutanova and Chen (2015) developed a link prediction

[1] https://code.google.com/archive/p/wiki-links/downloads

Dataset	Entities	Relations	Training	Validation	Test
FB15K	14 951	1 345	483 142	50 000	59 071
WN18	40 943	18	141 442	5 000	5 000
FB15K-237	14 505	237	272 115	17 535	20 466
WNRR	40 559	11	86 835	3 034	3 134
YAGO3-10	123 182	37	1 079 040	5 000	5 000
WIKIDATA5M	4 818 679	828	21 343 681	5 357	5 321

Table 2.3: Statistics of benchmark datasets used throughout this thesis.

model designed to exploit observed features in a KG, in contrast to the latent features learned by KGE models. This model was based on binary features that indicated the existence of simple patterns in the graph, such as different relations that interacted with the same pairs of entities, or the existence of inverse or near-inverse relations based on the entities they interacted with. Their proposed model significantly outperformed KGE models in their experiments, especially on the FB15K dataset (introduced above). They found that this difference in performance was explained by the fact that the majority of the test triples were closely related to existing triples in the training set. Specifically, given test triple (i, k, j), they found that almost 81% of the time, either triple (i, k', j) or (j, k', i) was present in the training set, where $k \neq k'$. This motivated them to create the FB15K-237 dataset (originally called FB15KSelected).

The FB15K-237 dataset was created by first selecting the 401 most frequent relations in the dataset and then filtering out relations that were either duplicates or inverse of other existing relations. They determined that two relations were the same (or inverse) if at least 97% of the entity pairs (or inverse entity pairs) they interacted with were the same. In addition, they removed from the validation and test splits any triple (i, k, j) if a triple (i, k', j) was present in the training split, where $k \neq k'$. This step was done to further increase the difficulty of the task, despite understanding that such a scenario could be present in real-world applications.

WNRR. Dettmers et al. (2018) found that the same issues that motivated the creation of the FB15K-237 dataset were also present in the WN18 dataset (introduced above). They thus created the WNRR dataset (originally called WN18RR) by applying the same procedure to WN18 that Toutanova and

Chen (2015) applied to FB15K to create the FB15K-237 dataset.

YAGO3-10. Dettmers et al. (2018) introduced this dataset as a subset of YAGO3 (Mahdisoltani et al., 2014), itself a multilingual extension of the YAGO KG introduced in Section 2.1.4. YAGO3-10 was constructed by selecting entities in YAGO3 that interacted with at least 10 different relations. This resulted in a considerably larger benchmark dataset than other commonly used ones at the time of its publication.

WIKIDATA5M. Wang et al. (2021) proposed to jointly learn a language model and a KGE model with the purpose of producing both a more factually accurate language model and a text-enhanced KGE model that can make inductive predictions (see Section 2.2.8) by creating representations of unseen entities based on their textual descriptions. To train their proposed model, Wang et al. (2021) constructed WIKIDATA5M, a large dataset that included textual descriptions of its entities and relations, and an additional data split to test a KGE model's ability to make inductive predictions. The authors constructed this dataset based on the Wikidata knowledge graph introduced in Section 2.1.4. Specifically, for each entity in the KG with a corresponding Wikipedia page, they extracted the first section of its Wikipedia page as its textual description, so long as this description was at least 5 words long. They then obtained all relational facts from Wikidata for this entity, where a fact was valid if it involved another entity from this set. The resulting dataset is much larger than all other datasets introduced in this section. In this thesis, we do not use the textual descriptions of entities provided in this dataset, and we do not test models on inductive link prediction.

Other datasets. Some benchmark datasets used in the KGE literature are designed for link prediction in specific domain areas. For example, with the purpose of fostering reproducible research in biological systems, Walsh et al. (2020) created the BioKG dataset, a KG that unifies and standardizes data from several open biological databases.

Other datasets in the literature include additional data aside from the commonly provided set of triples. For example, the CoDEx dataset (Safavi and Koutra, 2020) includes additional information about its entities and relations, such as their types, as well as textual descriptions from Wikidata and Wikipedia. CoDEx also includes manually annotated negative triples, i.e. true negatives, in the validation and test splits, with the goal of improving the quality of evaluation in some tasks like triple classification (see Section 2.2.6). The set of negatives that was manually annotated was constructed from

unobserved but type-accurate triples predicted by trained KGE models given known positive triples. Thus, they were intended to be non-trivial, e.g. type-inconsistent, negatives. Finally, datasets like ILPC2022 introduced by Galkin et al. (2022) were designed for inductive link prediction, i.e. performing inference over a graph with entities unseen during training.

CHAPTER

THREE

LINK PREDICTION

A vast number of different KGE models for multi-relational link prediction have been proposed in the literature; e.g., RESCAL (Nickel et al., 2011), TransE (Bordes et al., 2013b), DistMult (Yang et al., 2015), ComplEx (Trouillon et al., 2016), ConvE (Dettmers et al., 2018), RotatE (Sun et al., 2019), HittEr (Chen et al., 2021a), MQuadE (Yu et al., 2021b) and many others covered in recent surveys (Ji et al., 2021; Wang et al., 2017). As discussed in Chapter 2, model architectures generally differ in the way the entity and relation embeddings are combined to model the presence or absence of an edge in a KG, typically represented as a subject-predicate-object triple. The types of models include factorization models (e.g., RESCAL, DistMult, ComplEx, TuckER), translational models (TransE, RotatE), and more advanced models such as those based on convolutional neural networks (ConvE) or attention-based models (HittEr). In many cases, the introduction of new models also came with new approaches for training these models—e.g., new training types (such as negative sampling or 1vsAll scoring), new loss functions (such as pairwise margin ranking or binary cross entropy), new forms of regularization (such as unweighted and weighted L2), or the use of reciprocal relations (Kazemi and Poole, 2018; Lacroix et al., 2018)—and ablation studies were not always performed. Table 3.1 shows an overview of selected models along with some training techniques they introduced.

The diversity in model training makes it difficult to compare performance results for various model architectures, especially when results are repro-

Publication	Model	Loss	Training	Regularizer	Optimizer
Nickel et al. (2011)	**RESCAL**	MSE	Full	**L2**	ALS
Bordes et al. (2013b)	**TransE**	**MR**	**NegSamp**	**Emb. Norm.**	**SGD**
Yang et al. (2015)	**DistMult**	MR	NegSamp	**Weighted** L2	**Adagrad**
Trouillon et al. (2016)	**ComplEx**	**BCE**	NegSamp	Weighted L2	Adagrad
Kadlec et al. (2017)	DistMult	**CE**	NegSamp	Weighted L2	**Adam**
Dettmers et al. (2018)	**ConvE**	BCE	**KvsAll**	**Dropout**	Adam
Lacroix et al. (2018)	ComplEx	CE	**1vsAll**	Weighted **L3**	Adagrad

MSE = mean squared error, MR = margin ranking, BCE = binary cross entropy, CE = cross entropy, Emb. Norm. = embedding normalization

Table 3.1: Selected KGE models and training strategies from the literature. Entries marked in bold were introduced (or first used) in the context of KGE in the corresponding publication.

duced from prior studies that used a different experimental setup. Model hyperparameters are commonly tuned using grid search on a small grid involving hand-crafted parameter ranges or settings known to "work well" from prior studies. However, a grid suitable for one model may be suboptimal for another. Indeed, it has been observed that newer training strategies can considerably improve model performance (Kadlec et al., 2017; Lacroix et al., 2018; Salehi et al., 2018).

In this chapter, we look at the empirical study we conducted with the goal of summarizing and quantifying the impact of different model architectures and different training strategies on model performance on the link prediction task (Ruffinelli et al., 2020). For this thesis, we extended this study by including more datasets and KGE models in our experiments, as well as extending the discussion of our results. We performed an extensive set of experiments using popular model architectures and training strategies in a common experimental setup. In contrast to most prior work, we considered many training strategies as well as a large hyperparameter space, and we performed model selection using quasi-random search (instead of grid search) followed by Bayesian optimization. We found that this approach was able to find good (and often superior to prior studies) model configurations with relatively low effort.

Through this study, we found that:

1. When trained appropriately, the performance of a particular model architecture can by far exceed the performance observed in older studies. For example, RESCAL (Nickel et al., 2011), which constitutes one of the first KGE models but is rarely considered in newer work, showed very strong performance in our study: it was competitive to or outperformed more recent architectures such as ConvE (Dettmers et al., 2018) and TuckER (Balazevic et al., 2019).

2. More generally, we found that the relative performance differences between various model architectures often shrunk and sometimes even reversed when compared to prior results. This suggests that training strategies have a significant impact on model performance and may account for a substantial fraction of the progress made in recent years.

3. We also found that suitable training strategies and hyperparameter settings vary significantly across models and datasets, indicating that a small grid search may bias results on model performance. Fortunately, as indicated above, large hyperparameter spaces can be (and should be) used with little additional training effort.

This study focused solely on pure KGE models, which do not exploit auxiliary information such as textual data or logical rules (Ji et al., 2021; Wang et al., 2017). Since many of the studies on these non-pure models did not (and, to be fair, could not) use current training strategies and consequently underestimated the performance of pure KGE models, their results and conclusions need to be revisited.

As part of this study and to promote robust experimental settings in future work by the research community, we developed and provided implementations of relevant training strategies, models, and evaluation methods as an open source framework called LibKGE (Broscheit et al., 2020), which emphasizes reproducibility and extensibility.

We begin this chapter by discussing the various training methods for KGE models in Section 3.1. We present and discuss our experiments in Section 3.2. We discuss related work in Section 3.3 and summarize our contributions in Section 3.4.

Algorithm 3.1: 1vsAll Training

Require: $\mathcal{T}$: set of training triples,
$\mathcal{E}$: set of entities in knowledge graph $\mathcal{K}$
θ: model parameters
Ensure: Updated model parameters θ

$\mathcal{T}_{\text{all}} \leftarrow \mathcal{T}$
foreach $(i,k,j) \in \mathcal{T}$ **do**
  // object prediction
  $\mathcal{N}_o \leftarrow$ construct $|\mathcal{E}| - 1$ negatives (i,k,j') with every $j' \neq j \in \mathcal{E}$
  // subject prediction
  $\mathcal{N}_s \leftarrow$ construct $|\mathcal{E}| - 1$ negatives (i',k,j) with every $i' \neq i \in \mathcal{E}$
  $\mathcal{T}_{\text{all}} \leftarrow \mathcal{T}_{\text{all}} \cup N_o \cup N_s$
$S_{\text{all}} \leftarrow$ Compute_Scores$(\mathcal{T}_{\text{all}})$
$L_{\text{all}} \leftarrow$ Compute_Loss$(S_{\text{all}}, \mathcal{T}_{\text{all}})$
$\theta \leftarrow$ Update_Parameters(θ, L_{all})

3.1 Training Components

In Chapter 2, we discussed the general approach to training KGE models. In this section, and as part of our study to assess their impact on performance, we discuss some of the most important training components applied in the literature. Many, but not all, of the methods introduced in the following, are part of this comparative study. We describe the scope of our experiments in detail in Section 3.2.1.

Training types. There are three commonly used approaches to train KGE models, which differ mainly in the way negative examples are used. First, as explained in Section 2.2.7, training with negative sampling (NegSamp) (Bordes et al., 2013b) means obtaining for each positive triple $t = (i,k,j)$ from the training data a set of (pseudo-)negative triples by randomly perturbing the subject, relation, or object position in t (and optionally verifying that the so-obtained triples do not exist in the KG).

An alternative approach (Lacroix et al., 2018), which we term *1vsAll*, is to omit sampling and take *all* triples that can be obtained by perturbing the subject and object positions as negative examples for t (even if these tuples exist in the KG). 1vsAll is generally more expensive than NegSamp, but it is feasible (and even surprisingly fast in practice) if the number of entities is

Algorithm 3.2: KvsAll Training

Require: $\mathcal{T}$: set of training triples,
$\mathcal{E}$: set of entities in knowledge graph $\mathcal{K}$
θ: model parameters
Ensure: Updated model parameters θ

$\mathcal{T}_{\text{all}} \leftarrow \epsilon$ // empty collection
foreach unique (i, k) in triples $(i, k, j) \in \mathcal{T}$ **do**
  // object prediction
  $O_p \leftarrow$ get all entities $j' \in \mathcal{E}$ such that $(i, k, j') \in \mathcal{T}$
  $P_o \leftarrow$ construct positive triples (i, k, j') with every $j' \in O_p$
  $O_n \leftarrow$ get all entities $j' \in \mathcal{E}$ such that $(i, k, j') \notin \mathcal{T}$
  $N_o \leftarrow$ construct negative triples (i, k, j') with every $j' \in O_n$
  $\mathcal{T}_{\text{all}} \leftarrow \mathcal{T}_{\text{all}} \cup P_o \cup N_o$
foreach unique (k, j) in triples $(i, k, j) \in \mathcal{T}$ **do**
  // subject prediction
  $S_p \leftarrow$ get all entities $i' \in \mathcal{E}$ such that $(i', k, j) \in \mathcal{T}$
  $P_s \leftarrow$ construct positive triples (i', k, j) with every $i' \in S_p$
  $S_n \leftarrow$ get all entities $i' \in \mathcal{E}$ such that $(i', k, j) \notin \mathcal{T}$
  $N_s \leftarrow$ construct negative triples (i', k, j) with every $i' \in S_n$
  $\mathcal{T}_{\text{all}} \leftarrow \mathcal{T}_{\text{all}} \cup P_s \cup N_s$
$S_{\text{all}} \leftarrow$ Compute_Scores$(\mathcal{T}_{\text{all}})$
$L_{\text{all}} \leftarrow$ Compute_Loss$(S_{\text{all}}, \mathcal{T}_{\text{all}})$
$\theta \leftarrow$ Update_Parameters(θ, L_{all})

not excessively large. The 1vsAll training type is described by Algorithm 3.1.

Finally, Dettmers et al. (2018) proposed a training type that we term *KvsAll* (originally called 1-N scoring by the authors), and is described by Algorithm 3.2. This approach (i) constructs batches from non-empty rows $(i, k, *)$ or $(*, k, j)$ instead of from individual triples, and (ii) labels all such triples as either positive (occurs in training data) or negative (otherwise). This results in a training set whose size is not proportional to the number of unique triples, but the number of unique $(i, k, *)$ and $(*, k, j)$ tuples.

By training with either NegSamp or 1vsAll, we treat the link prediction task as a multi-class problem, as for every triple there is a single correct answer from a set of possible predictions (*True* or *False* for NegSamp and the single correct entity for 1vsAll. However, by training with KvsAll we treat the task as a multi-label problem, as there are potentially many possible correct

answers for a given tuple $(i, k, *)$ or $(*, k, j)$ as seen in the training set.

Loss functions. Several loss functions for training KGEs have been introduced so far. RESCAL (Nickel et al., 2011) originally used squared error between the score of each triple and its label (positive or negative). Thus, the authors modeled the task as reconstructing the binary 3-way tensor that corresponds to a given training set.

TransE (Bordes et al., 2013b) used pairwise margin ranking with hinge loss (MR), where each pair consists of a positive triple and one of its negative triples (only applicable to NegSamp and 1vsAll) and the margin η is a hyperparameter. Trouillon et al. (2016) proposed to use binary cross entropy (BCE) loss: it applies a sigmoid to the score of each (positive or negative) triple and uses the cross entropy between the resulting probability and that triple's label as loss.

Finally, Kadlec et al. (2017) used cross entropy (CE) between the model distribution (softmax distribution over scores) and the data distribution (labels of corresponding triples, normalized to sum to 1). CE has been used for multi-class classification (as in NegSamp and 1vsAll), but also in the multi-label setting (KvsAll).

Mohamed et al. (2019) found that the choice of loss function can have a significant impact on model performance, and that the best choice is data and model dependent. Our experimental study provides additional evidence for this finding.

Regularization. Several forms of regularization have been used when training KGEs. The most popular form of regularization in the literature is L2 regularization on the embedding vectors, either unweighted or weighted by the frequency of the corresponding entity or relation in the training set (Yang et al., 2015). Lacroix et al. (2018) proposed to use L3 regularization. TransE normalized the embeddings to unit norm after each update. ConvE used dropout (Srivastava et al., 2014) in its hidden layers (and only in those), and Balazevic et al. (2019) used dropout in the embedding layer. Wang et al. (2022a) proposed the use of constraints on the embedding space to encourage embedding similarity between semantically similar entities.

Reciprocal relations. A particular hyperparameter that is exclusive to KGE training is the technique of *reciprocal relations*, introduced by Kazemi and Poole (2018) and Lacroix et al. (2018). Observe that during evaluation and also most training methods discussed above, the model is asked to score

either subjects (for questions of the form $(?, k, j)$) or objects (for questions of the form $(i, k, ?)$). The idea of reciprocal relations is to use separate scoring functions s_{sub} and s_{obj} for each of these tasks, resp. Both scoring functions share entity embeddings, but they do not share relation embeddings: each relation thus has two embeddings. Thus, we can view such an approach as only predicting objects, but doing so for both each original relation (k) and a new reciprocal relation formed by switching subject and object (k^{-1}).

The use of reciprocal relations may decrease the computational cost (as in the case of ConvE), and it may also lead to better model performance as in Lacroix et al. (2018) (e.g., for relations in which one direction is easier to predict). On the downside, the use of reciprocal relations means that a model does not provide a single triple score $s(i, k, j)$ anymore, since in general, $s_{\text{sub}}(i, k, j) \neq s_{\text{obj}}(i, k, j)$. While we are unfamiliar with a principled way to handle this, some studies have explored this issue. For example, Kazemi and Poole (2018) proposed to take the average of the two triple scores and explored the resulting models.

Other hyperparameters. Aside from the choice of training type, loss function and the various forms of regularization, many more hyperparameters have been used in prior work. This includes, for example, different methods to initialize model parameters, different optimizers, different optimizer parameters such as the learning rate or batch size, the number of negative examples for NegSamp, the regularization weights for entities and relations, and so on. To deal with such a large search space, most prior studies favor grid search over a small grid where most of these hyperparameters remain fixed (Zhang et al., 2023; Chen et al., 2021b; Abboud et al., 2020; Sun et al., 2019). However, as discussed before, this approach may lead to bias in the results.

3.2 Experimental Study

In this section, we report on the design and results of our experimental study. The goal of these experiments is to better assess the impact on model performance caused by (i) model architectures and (ii) training strategies. To that end, all models are implemented in the same codebase and are given the same resources in terms of hyperparameter optimization, training and evaluation.

Dataset	Entities	Relations	Training	Validation	Test
FB15K	14 951	1 345	483 142	50 000	59 071
WN18	40 943	18	141 442	5 000	5 000
FB-237	14 505	237	272 115	17 535	20 466
WNRR	40 559	11	86 835	3 034	3 134

Table 3.2: Statistics of datasets used in this study.

3.2.1 Experimental Settings

Datasets. We used FB15K (Bordes et al., 2013b), WN18 (Bordes et al., 2013b), FB15K-237 (Toutanova and Chen, 2015) (referred to as FB-237 for brevity) and WNRR (Dettmers et al., 2018). FB15K was extracted from the Freebase (Bollacker et al., 2008) knowledge base. As described in Section 2.4, FB-237 was constructed as a "harder" variant of FB15K by removing triples from training that might trivialize the link prediction task. Similarly, WN18 was extracted from WordNet (Miller, 1995) and WNRR was built as the "harder" variant of it. We chose these datasets because (i) they are frequently used in prior studies, (ii) some are known to be "easy", while some are designed to be "hard", (iii) they are diverse in that relative model performance often differs, and (iv) they are of reasonable size for a large study. Dataset statistics are given in Table 3.2.

Models. We selected RESCAL (Nickel et al., 2011), TransE (Bordes et al., 2013b), DistMult (Yang et al., 2015), ComplEx (Trouillon et al., 2016), ConvE (Dettmers et al., 2018) and RotatE (Sun et al., 2019) for our study. These are all well-known KGE models. We include both early and more recent models, and chose them to have a diverse set of scoring functions in our experiments. RESCAL, DistMult and ComplEx are three different bilinear models, TransE and RotatE are translational and ConvE is based on convolutional neural networks. As baseline, we also include AnyBURL (Meilicke et al., 2019), a rule-based model designed for link prediction that showed competitive performance compared to KGE models. For details about these KGE models, see Section 2.2.5. For details about AnyBURL, see Section 2.3.3.

Evaluation. We use the entity ranking evaluation protocol and report filtered MRR (%) and filtered Hits@10 (%) (also denoted by H@10 for brevity). See Section 2.2.6 for details on the evaluation method and reported met-

rics. We use test data to compare final model performance (Table 3.4) and validation data for our more detailed analysis.

Hyperparameters. We used a large hyperparameter space to ensure sure that suitable hyperparameters for each model are not excluded a priori. We included all major training types (NegSamp, 1vsAll, KvsAll), use of reciprocal relations, loss functions (MR, BCE, CE), regularization techniques (none/L1/L2/L3, dropout), optimizers (Adam, Adagrad), and initialization methods (4 in total) used in the KGE community as hyperparameters. We considered three embeddings sizes (128, 256, 512) and used separate weights for dropout/regularization for entity and relation embeddings. Table 3.3 provides a complete description of the search space available for all models. To train TransE and RotatE with BCE, we implemented an offset as a hyperparameter that is applied to the scores produced by the model. This is equivalent to the *gamma* hyperparameter in Sun et al. (2019). To the best of our knowledge, no prior study had used such a large hyperparameter search space.

Training. All models were trained for a maximum of 400 epochs. We validated models using filtered MRR (on validation data) every five epochs and performed early stopping with a patience of 50 epochs. To keep search tractable, we stopped training on models that did not reach $\geq 5\%$ filtered MRR after 50 epochs; in preliminary results, such configurations did not produce good models.

Model selection. Model selection was performed using filtered MRR on validation data. We used the Ax framework (`https://ax.dev/`) to conduct quasi-random hyperparameter search via a Sobol sequence. Quasi-random search methods aim to distribute hyperparameter settings evenly and try to avoid "clumping" effects (Bergstra and Bengio, 2012). More specifically, for each dataset and model, we generated 30 different configurations per valid combination of training type and loss function (7 combinations for each model, 3 for TransE and RotatE which were only trained with NegSamp). After the quasi-random hyperparameter search, we added a short Bayesian optimization phase (best configuration so far + 20 new trials, using expected improvement; also provided by Ax) to tune the numerical hyperparameters further. Finally, we trained five models with the so-obtained hyperparameter configuration and selected the best-performing model according to validation MRR as the final model.

Reproducibility. We implemented all models, training strategies, evalua-

Hyperparameter	Random search	Bayesian optim.
Embedding size	$\{128, 256, 512\}$	See Tab. A.3
Training type	{NegSamp, 1vsAll, KvsAll}	See Tab. A.3
Reciprocal	{True, False}	See Tab. A.3
No. sub. samples (NegSamp)	$[1, 1000]$, log scale	$[1, 1000]$, log scale
No. obj. samples (NegSamp)	$[1, 1000]$, log scale	$[1, 1000]$, log scale
Label smoothing (KvsAll)	$[0.0, 0.3]$	$[0.0, 0.3]$
Loss	{BCE, MR, CE}	See Tab. A.3
Margin (MR)	$[0, 10]$	$[0, 10]$
L_p-norm (TransE)	{1, 2}	See Tab. A.3
Optimizer	{Adam, Adagrad}	See Tab. A.3
Batch size	$\{128, 256, 512, 1024\}$	See Tab. A.3
Learning rate	$[10^{-4}, 1]$, log scale	$[10^{-4}, 1]$, log scale
LR scheduler patience	$[0, 10]$	$[0, 10]$
L_p regularization	{L1, L2, L3, None}	See Tab. A.3
Entity emb. weight	$[10^{-20}, 10^{-5}]$	$[10^{-20}, 10^{-5}]$
Relation emb. weight	$[10^{-20}, 10^{-5}]$	$[10^{-20}, 10^{-5}]$
Frequency weighting	{True, False}	See Tab. A.3
Emb. normalization (TransE)		
Entity	{True, False}	See Tab. A.3
Relation	{True, False}	See Tab. A.3
Dropout		
Entity embedding	$[0.0, 0.5]$	$[0.0, 0.5]$
Relation embedding	$[0.0, 0.5]$	$[0.0, 0.5]$
Feature map (ConvE)	$[0.0, 0.5]$	$[0.0, 0.5]$
Projection (ConvE)	$[0.0, 0.5]$	$[0.0, 0.5]$
Embedding initialization	{Normal, Unif, Xavier}	See Tab. A.3
Std. deviation (Normal)	$[10^{-5}, 1.0]$	$[10^{-5}, 1.0]$
Interval (Unif)	$[-1.0, 1.0]$	$[-1.0, 1.0]$
Gain (XvNorm)	1.0	1.0
Gain (XvUnif)	1.0	1.0

Table 3.3: Hyperparameter search space used in our study. Settings that apply only to certain configurations are indicated in parenthesis.

tion, and hyperparameter search in the LibKGE framework (Broscheit et al., 2020), an framework we developed for this study and made available as open source.[1] The framework emphasizes reproducibility and extensibility and is highly configurable. We provide all configurations used in this study as well as the detailed data of our experiments for further analysis.[2] These resources have since been used by various studies to conduct robust experiments. The work of Safavi and Koutra (2020) and Gema et al. (2023) are examples of such studies.

3.2.2 Model Performance

Table 3.4 shows the filtered MRR and filtered Hits@10 on test data of various models both from prior studies and the best models (according to filtered validation MRR) found in our study.

First reported performance vs. our observed performance. We compared the first reported performance on FB-237 and WNRR ("First" block of Table 3.4) with the performance obtained in our study ("Ours" block). **We found that the performance of a single model can vary wildly across studies.** For example, ComplEx was first run on FB-237 by Dettmers et al. (2018), where it achieved a filtered MRR of 24.7%. This was a relatively low number by the standards at the time. In our study, ComplEx achieved a competitive MRR of 34.8%, which is a large improvement over the first reports. Studies that report the lower performance number of ComplEx (i.e., 24.7%) thus do not adequately capture its performance. Similar remarks hold for RESCAL and DistMult as well as (albeit to a smaller extent) ConvE and TransE. The exception is RotatE, which outperforms all other models in the "First" block. This is likely because it is the most recently published model in this block, and thus should arguably be compared with models in the "Recent" block. To the best of our knowledge, our results for RESCAL and ConvE constitute the best results for these models obtained at the time.

Relative model performance (our study). Next, we compared the performance across the models used in our study ("Ours" block). **We found that the relative performance differences between model architectures often shrunk and sometimes reversed when compared to prior results ("First" block).** For example, ConvE showed the best overall performance in prior

[1] https://github.com/uma-pi1/kge
[2] https://github.com/uma-pi1/kge-iclr20

		FB-237		*WNRR*		*FB15K*		*WN18*	
		MRR	H@10	MRR	H@10	MRR	H@10	MRR	H@10
First	RESCAL[*]	27.0	42.7	42.0	44.7	35.4	58.7	89.0	92.8
	TransE[†]	29.4	46.5	22.6	50.1	–	47.1	–	89.2
	DistMult[‡]	24.1	41.9	43.0	49.0	65.4	82.4	82.2	93.6
	ComplEx[‡]	24.7	42.8	44.0	51.0	69.2	84.0	94.1	94.7
	ConvE[‖]	32.5	50.1	43.0	52.0	65.7	83.1	94.3	95.6
	RotatE[#]	**33.8**	**53.3**	**47.6**	**57.1**	**79.9**	**89.2**	**95.3**	**95.8**
Ours	RESCAL	**35.6**	**54.1**	46.7	51.7	64.4	54.4	94.8	95.6
	TransE	33.0	52.0	23.3	52.3	68.2	87.0	61.5	93.5
	DistMult	34.3	53.1	45.2	53.0	**84.1**	**90.3**	94.1	95.4
	ComplEx	34.8	53.6	47.5	54.7	83.8	89.3	**95.1**	**95.8**
	ConvE	33.9	52.1	44.2	50.4	82.5	89.6	94.7	95.3
	RotatE	33.3	52.2	**47.8**	**55.3**	78.3	87.8	94.6	95.3
Recent	TuckER[**]	**35.8**	**54.4**	**47.0**	52.6	69.6	84.2	81.9	**96.4**
	SACN[††]	35.0	54.0	**47.0**	54.4	**79.7**	88.4	94.9	95.9
	RotatE[‡‡]	27.3	48.2	44.8	**60.2**	–	–	–	–
Large	DistMult[§§]	35.7	54.8	45.5	54.4	84.1	**91.4**	91.1	**96.1**
	ComplEx-N3[‖‖]	**37.0**	**56.0**	**49.0**	**58.0**	**86.0**	91.0	**95.0**	96.0
Rules	AnyBURL[##]	34.6	52.3	49.2	57.7	83.9	89.4	95.2	96.2

[*] FB15K, WN18: Nickel et al. (2016); FB-237, WNRR: Wang et al. (2019)
[†] FB15K, WN18: Bordes et al. (2013b); FB-237, WNRR: Nguyen et al. (2018)
[‡] FB15K, WN18: Trouillon et al. (2016); FB-237, WNRR: Dettmers et al. (2018) [‖] Dettmers et al. (2018) [#] Sun et al. (2019) [**] Balazevic et al. (2019) [††] Shang et al. (2019) [‡‡] Ali et al. (2021) [§§] Salehi et al. (2018)
[‖‖] Lacroix et al. (2018) [##] Meilicke et al. (2019)

Table 3.4: Model performance on test data in prior studies and our study. We report MRR and Hits@10 (H@10). *First*: first reported performance on each dataset (oldest models first); *Ours*: performance in our study; *Recent*: best performance of selected models obtained in recent studies; *Large*: best performance achieved in prior studies using more expensive models (not part of our search space). Bold numbers indicate best performance in group. References indicate where the performance number was reported. Results for TransE and RotatE on FB15K and WN18 were not available in prior studies.

studies, but is consistently outperformed by RESCAL, DistMult, and ComplEx in our study. As another example, RESCAL (Nickel et al., 2011), which constitutes one of the first KGE models but is rarely considered in newer work, showed strong performance and outperformed almost all models in three of the four datasets, being only slightly outperformed by ComplEx and RotatE on some datasets. The exception is FB15K, where RESCAL is considerably outperformed by all models. Since RESCAL has a considerably higher number of parameters compared to other models, we think this is likely due the much larger dataset, which results in RESCAL overfitting to the training set. Taken together, these results suggest that training strategies have a significant impact on model performance and may account for a large fraction of the progress made throughout the years.

Relative model performance (overall). Table 3.4 additionally shows the best performance results obtained in prior studies of selected recent models ("Recent" block) and very large models with very good performance ("Large" block). **We found that TuckER (Balazevic et al., 2019), RotatE (Sun et al., 2019), and SACN (Shang et al., 2019) all achieve state-of-the-art performance, but the performance difference to the best prior models ("Ours" block) in terms of MRR is small or even vanishes.** Even for HITS@10, which we did not consider for model selection, the advantage of more recent models is often small, if present. The exception is the RotatE obtained by RotatE (Ali et al., 2021), which shows impressive performance in Hits@10 on WNRR, but is outperformed by most models in terms of MRR.

The models in the last block ("Large") show the best performance numbers we have seen in the literature at the time. For DistMult and ComplEx, these numbers have been obtained using very large embedding sizes (up to 4000). Such large models are an order of magnitude larger than the embedding sizes available in our search space. We found that model performance is stable, as indicated by the standard deviation of the validation MRR, which is relatively low; see Table 3.5.

Rule-based baseline. Table 3.4 includes the performance of AnyBURL (Meilicke et al., 2019), a rule-based model that performs on par with state-of-the-art models on the link-prediction benchmark datasets. **Compared to KGE models, AnyBURL performs competitively on most datasets and even outperforms all reported KGE models in MRR on WNRR.** As discussed in Section 2.3.3, this shows that rule-based approaches are a viable alternative to link-prediction and a useful baseline to compare against. While they can

		RESCAL	TransE	DistMult	ComplEx	ConvE	RotatE
FB-237	MRR	36.1±0.3	33.5±0.1	35.0±0.0	35.3±0.1	34.3±0.1	34.0±0.0
	Hits@10	54.3±0.5	52.3±0.2	53.5±0.1	53.9±0.2	52.4±0.2	52.6±0.1
WNRR	MRR	46.8±0.2	23.0±0.1	45.4±0.1	47.6±0.1	44.3±0.1	47.7±0.2
	Hits@10	51.8±0.2	52.1±0.1	52.4±0.3	54.1±0.4	50.4±0.2	54.9±0.2
FB15K	MRR	–	68.2±0.1	84.0±0.1	83.8±0.2	82.1±0.2	78.2±0.2
	Hits@10	–	86.8±0.1	90.2±0.0	89.2±0.0	89.3±0.1	87.6±0.2
WN18	MRR	94.7±0.1	60.9±0.4	93.9±0.1	95.0±0.0	94.6±0.0	94.6±0.1
	Hits@10	95.4±0.1	92.9±0.2	95.1±0.1	95.5±0.1	95.3±0.1	95.1±0.2

Table 3.5: Mean and standard deviation of the validation data performance over five runs of each model using its best hyperparameter configuration. Most models are very stable, with low standard deviation.

have a higher runtime cost during prediction compared to embedding-based approaches, they require a considerably shorter training time (mostly due to the hyperparameter optimization required by KGE models), and can provide explanations for their predictions (Meilicke et al., 2019).

Model stability. In order to check the stability of the best performing models obtained in our study, we trained the best configurations for each combination of model and dataset five times. We report the mean and standard deviation of the validation MRR of these five runs in Table 3.5. **We observe that most models are very stable, with only RESCAL on FB-237 and TransE on WNRR showing relatively high standard deviation.**

Limitations. We note that all models used in our study are likely to yield better performance when hyperparameter tuning is further improved. For example, in preliminary tests we found a ComplEx configuration (of size 100) that achieved 49.0% MRR and 56.5% Hits@10 on WNRR, and a RESCAL configuration that achieved 36.3% MRR and 54.6% Hits@10 on FB-237. Sun et al. (2019) report on a TransE configuration that achieved 33.3% MRR (52.2% Hits@10) on FB-237. Since the focus of this study is to compare models in a common experimental setup and without manual intervention, we do not further report on these results.

In addition, more recent models, e.g. BoxE (Abboud et al., 2020), and newer training components, e.g. the regularization method proposed by Wang et al. (2022a), can be considered in an extension of this study. We discuss this further in Section 3.4.

3.2.3 Impact of Hyperparameters

Anatomy of search space. Figure 3.1 shows the distribution of filtered MRR for each model and dataset (on validation data, Sobol configurations only). Each distribution consists of 210 different quasi-random hyperparameter configurations (except TransE, for which 90 configurations were used). Perhaps unsurprisingly, **we found that all models showed a wide dispersion on both datasets and only very few configurations achieved state-of-the-art results.** There are, however, notable differences across datasets and models. For example, on FB-237, the median ConvE configuration is best, whereas ComplEx, DistMult and ConvE have better median MRR on FB15K. Similarly, on WNRR and WN18, DistMult and Complex have a much higher median MRR. Generally, the impact of the hyperparameter choice is more pronounced on WNRR and WN18 (higher variance) than on FB-237 and FB15K.

Best configurations (quasi-random search). The hyperparameters of the best performing models during our quasi-random search are reported in Table 3.6 for selected models and hyperparameters. For all models and hyperparameters, see Tables A.3 to A.6 in Appendix A. **We found that the optimum choice of hyperparameters is often model and dataset dependent: with the exception of the loss function (discussed below), almost no hyperparameter setting was consistent across datasets and models.** For example, the NegSamp training type was frequently selected on FB-237 and FB15K, but not on WNRR or WN18, where the most frequent choices where

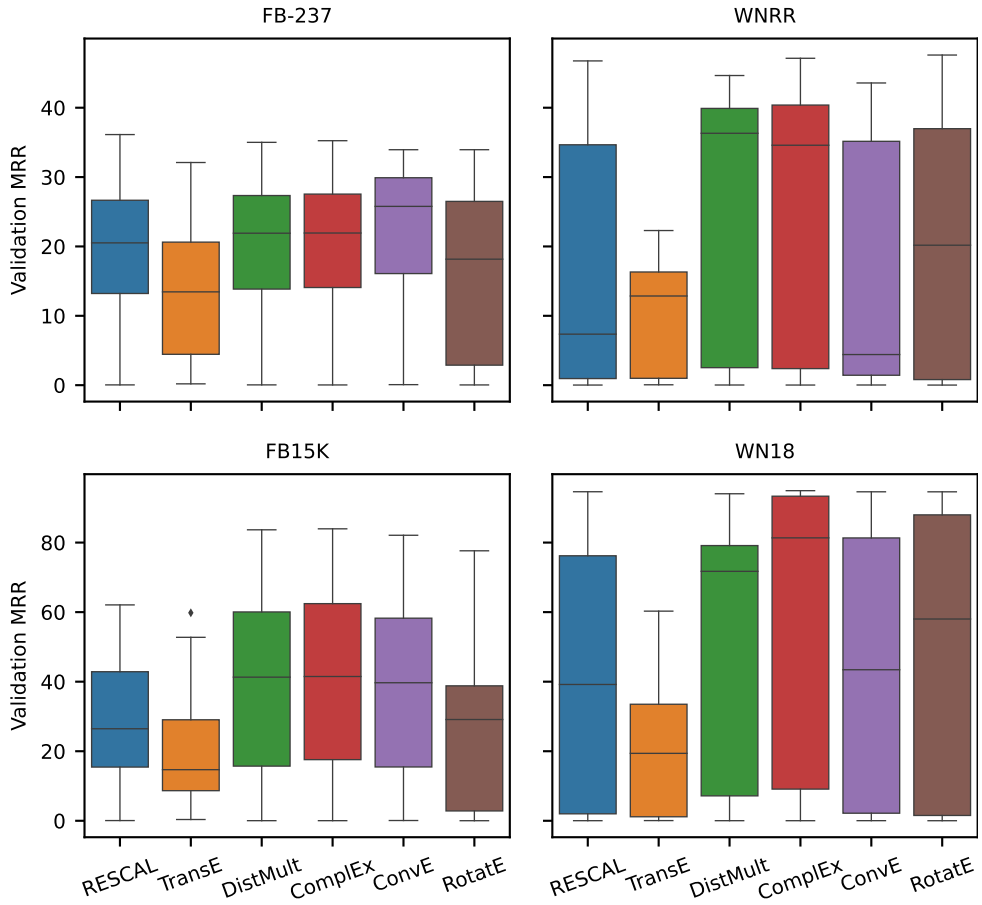


Figure 3.1: Distribution of filtered MRR (%) on validation data over the quasi-random hyperparameter configurations explored in our study.

KvsAll and 1vsAll, respectively. These findings provide further evidence that grid search on a small search space is not suitable to compare model performance because the result may be heavily influenced by the specific grid points being used. Moreover, the budget that we used for quasi-random search was comparable to a grid search over a small (or at least not very large) grid: we tested merely 30 different model configurations for every combination of training type and loss function. It is therefore both feasible and beneficial to work with a large search space.

Best configurations (Bayesian optimization). We already obtained good configurations solely from quasi-random search. **After Bayesian optimization**

		RESCAL	TransE	DistMult	ComplEx
FB-237	Valid MRR	*36.1*	*33.5*	*35.0*	*35.3*
	Emb. size	128 *(-0.5)*	512 *(-2.8)*	256 *(-0.2)*	256 *(-0.3)*
	Batch size	512 *(-0.5)*	128 *(-2.8)*	1024 *(-0.2)*	1024 *(-0.3)*
	Train type	1vsAll *(-0.8)*	NegSamp –	NegSamp *(-0.2)*	NegSamp *(-0.3)*
	Loss	CE *(-0.9)*	CE *(-2.8)*	CE *(-3.1)*	CE *(-3.8)*
	Optimizer	Adam *(-0.5)*	Adagr. *(-2.8)*	Adagr. *(-0.2)*	Adagr. *(-0.5)*
	Initializer	Norm. *(-0.8)*	XvNorm *(-2.8)*	Unif. *(-0.2)*	Unif. *(-0.5)*
	Reciprocal	No *(-0.5)*	Yes *(-3.0)*	Yes *(-0.3)*	Yes *(-0.3)*
WNRR	Valid MRR	*46.8*	*23.0*	*45.4*	*47.6*
	Emb. size	128 *(-1.0)*	512 *(-0.9)*	512 *(-1.1)*	128 *(-1.0)*
	Batch size	128 *(-1.0)*	128 *(-0.9)*	1024 *(-1.1)*	512 *(-1.0)*
	Train type	KvsAll *(-1.0)*	NegSamp –	KvsAll *(-1.1)*	1vsAll *(-1.0)*
	Loss	CE *(-2.0)*	CE *(-3.4)*	CE *(-2.4)*	CE *(-3.5)*
	Optimizer	Adam *(-1.2)*	Adagr. *(-2.6)*	Adagr. *(-1.5)*	Adagr. *(-1.5)*
	Initializer	Unif. *(-1.0)*	XvNorm *(-2.1)*	Unif. *(-1.3)*	Unif. *(-1.5)*
	Reciprocal	Yes *(-1.0)*	Yes *(-2.1)*	Yes *(-1.1)*	No *(-1.0)*
FB15K	Valid MRR	– –	*68.2*	*84.0*	*83.8*
	Emb. size	– –	512 *(-18.1)*	512 *(-1.2)*	256 *(0.0)*
	Batch size	– –	256 *(-15.4)*	1024 *(-1.2)*	256 *(0.0)*
	Train type	– –	NegSamp –	1vsAll *(-5.9)*	1vsAll *(-7.2)*
	Loss	– –	BCE *(-15.4)*	CE *(-3.1)*	CE *(-1.7)*
	Optimizer	– –	Adagr. *(-23.3)*	Adam *(-1.2)*	Adagr. *(0.0)*
	Initializer	– –	Norm. *(-15.4)*	XvNorm *(-1.2)*	XvUnif *(0.0)*
	Reciprocal	– –	Yes *(-18.1)*	Yes *(-1.2)*	No *(0.0)*
WN18	Valid MRR	*94.7*	*60.9*	*93.9*	*95.0*
	Emb. size	256 *(-0.3)*	256 *(-1.4)*	512 *(0.1)*	512 *(-0.1)*
	Batch size	256 *(-0.3)*	512 *(-1.4)*	128 *(0.1)*	1024 *(-0.1)*
	Train type	1vsAll *(-0.3)*	NegSamp –	1vsAll *(-4.4)*	KvsAll *(-0.1)*
	Loss	CE *(-1.5)*	CE *(-8.0)*	CE *(-9.8)*	CE *(-0.5)*
	Optimizer	Adagr. *(-0.3)*	Adagr. *(-1.4)*	Adagr. *(-1.1)*	Adagr. *(-0.2)*
	Initializer	XvUnif *(-0.3)*	XvNorm *(-2.1)*	Norm. *(-0.4)*	Unif. *(-0.1)*
	Reciprocal	Yes *(-0.3)*	Yes *(-1.4)*	Yes *(-9.8)*	Yes *(-0.1)*

Table 3.6: Hyperparameters of best performing models after quasi-random hyperparameter search and Bayesian optimization w.r.t. filtered MRR on validation data. For each hyperparameter, we also give the reduction in filtered MRR for the best configuration that does not use this value of the hyperparameter (in parenthesis).

(which we used to tune the numerical hyperparameters), the performance on test data almost always improved only marginally. The notable exception was TransE on FB15K, for which Bayesian optimization provided an improvement of about 10%. The best hyperparameters after the Bayesian optimization phase are reported in Appendix A.

Impact of specific hyperparameters. It is difficult to assess the impact of each hyperparameter individually. As a proxy for the importance of each hyperparameter setting, we report in Table 3.6 the best performance of some models in terms of filtered MRR on validation data for a *different choice* of value for each hyperparameter (difference in parenthesis). We report results for all models and all hyperparameters in Appendix A. For example, the best configuration during quasi-random search for RESCAL on FB-237 did not use reciprocal relations. The best configuration that did use reciprocal relations had an MRR that was 0.5 points lower (35.6 instead of 36.1). This does not mean that the gap is explained by the use of the reciprocal relations alone, as other hyperparameters may also be different, but it does show the best performance we obtained when enforcing reciprocal relations. **We can see that most hyperparameters appear to have moderate impact on model performance: many hyperparameter settings did not produce a significant performance improvement that others did not also produce.** A clear exception here is the use of the loss function, which is consistent across models and datasets (always CE), partly with large performance improvements over alternative loss functions. The training type shows a similar impact but to a lesser degree, since this is only observed on some models on FB15K and WN18. The larger performance gaps observed in DistMult and TransE are due to a combination of outlier models and considerable improvement after Bayesian optimization.

Impact of loss function. In order to further explore the impact of the loss function, we show in Figure 3.2 the distribution of the filtered MRR on validation data for every (valid) combination of training type and loss function. **We found that the use of CE (the three rightmost bars) generally led to better configurations than using other losses.** This is surprising (and somewhat unsatisfying) in that we use the CE loss, which is a multi-class loss, for a multi-label task. However, similar observations have been made for computer vision tasks (Joulin et al., 2016; Mahajan et al., 2018). Note that the combination of KvsAll with CE (on the very right in figure) has not been explored previously.

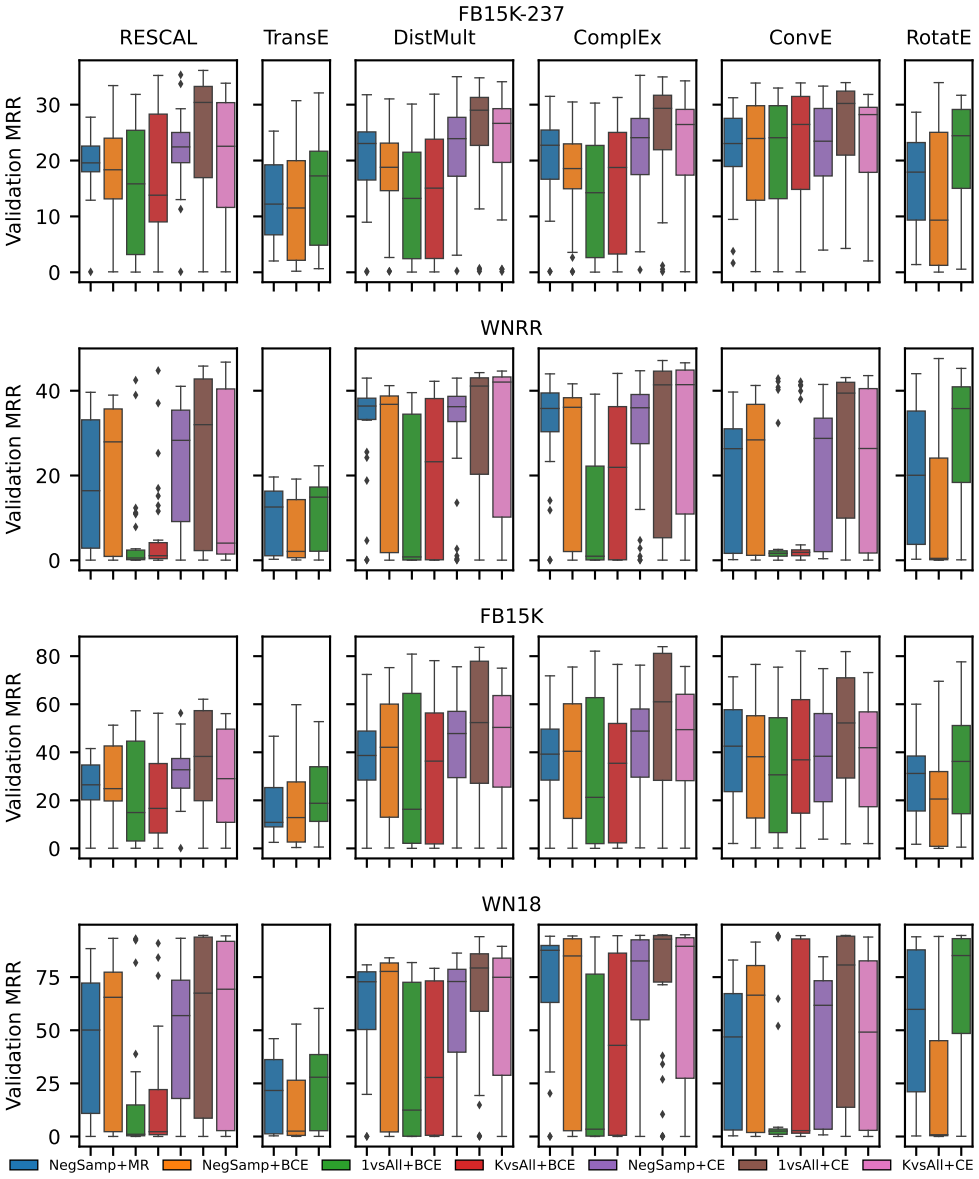


Figure 3.2: Distribution of filtered MRR (%) on validation data over the quasi-random hyperparameter configurations for different training type and loss functions. Use of the CE loss generally leads to better hyperparameter configurations.

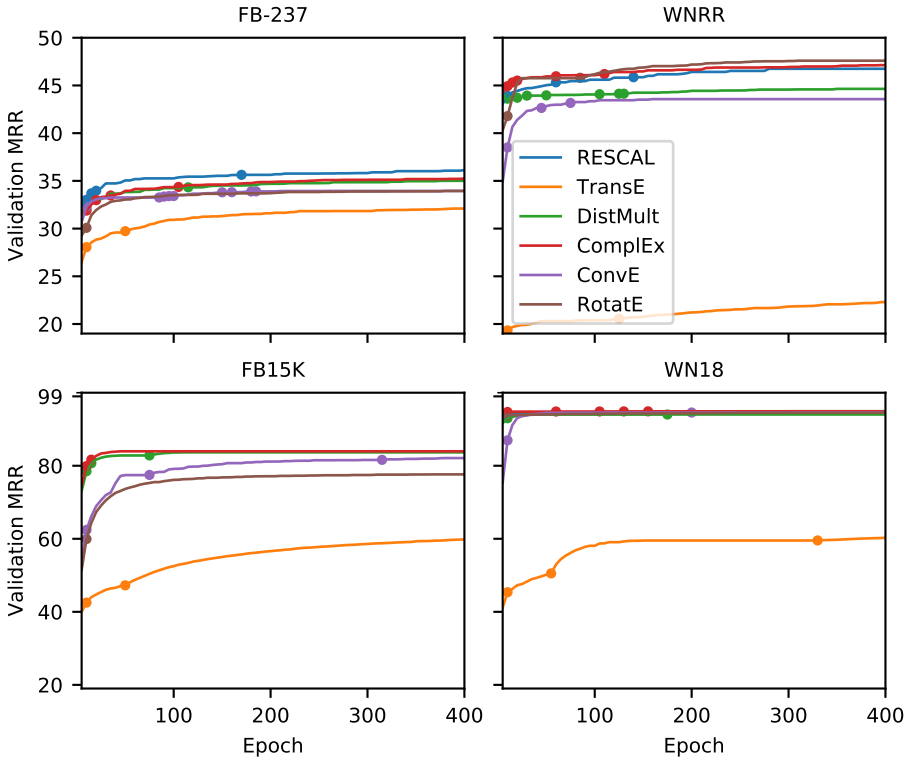


Figure 3.3: Best filtered MRR (%) on validation data achieved during quasi-random search as a function of the number of training epochs. Changes in the corresponding "best-so-far" hyperparameter configuration are marked with a dot. The best choice for hyperparameter configuration was clear in less than 200 epochs, most of the time.

Model Selection. We briefly summarize the behavior of various models during model selection. Figure 3.3 shows the best validation MRR (over all quasi-random hyperparameter configurations) achieved by each model when trained for the specified number of epochs. The corresponding hyperparameter configuration may change as models were trained longer; in the figure, we marked such changes with a dot. **We found that a suitable hyperparameter configuration as well as a good model can be found using significantly less than the 400 epochs used in our study (less than 200 in almost every case).** Nevertheless, longer training did help in many settings and some models may benefit when trained for even more than 400 epochs.

	FB-237		*WNRR*	
	MRR	Hits@10	MRR	Hits@10
RESCAL	35.6	54.1	46.7	51.7
RESCAL *w/o unseen*	35.6	54.1	49.9	55.2
TransE	31.3	49.7	22.8	52.0
TransE *w/o unseen*	31.3	49.8	24.4	55.7
DistMult	34.3	53.1	45.2	53.0
DistMult *w/o unseen*	34.3	53.2	48.3	56.6
Complex	34.8	53.6	47.5	54.7
Complex *w/o unseen*	34.8	53.6	50.9	58.7
ConvE	33.9	52.1	44.2	50.4
ConvE *w/o unseen*	33.9	52.1	47.3	53.8

Table 3.7: Comparison of model performance metrics when filtering out evaluation triples that contain entities not included in the training set (*w/o unseen*).

3.2.4 Impact of Variations in Evaluation

Our results in the previous sections show that KGE models are sensitive to, and can thus perform wildly differently, with different training and hyperparameter optimization settings. In the following, we discuss a few choices in the entity ranking evaluation protocol that can also make a significant difference in model performance, and should therefore taken into consideration when designing experiments and when comparing results across different publications.

Evaluating unseen entities. One issue follows the fact that some datasets have validation or test triples with entities that are not included in the training split, which we refer to as *unseen entities*. Some studies include these triples as part of the evaluation, since they are part of *benchmark* datasets. Others filter these triples out, and do not include the corresponding unseen entities as candidates in the ranking step. To measure the effect of this type of filtering, we evaluated some of our best models using both types of evaluations and report these results in Table 3.7. **We observe that while this type of filtering has almost no effect on FB-237, it has a considerable effect on WNRR.** This

		MRR	MRR *fwt*	Hits@10	Hits@10 *fwt*
FB-237	ComplEx	31.0	35.1	50.0	53.8
	ComplEx *fwt*	31.0	35.2	50.5	54.1
WNRR	ComplEx	45.9	46.9	52.3	52.5
	ComplEx *fwt*	46.0	47.1	52.9	53.2

Table 3.8: Comparison of performance metrics on validation data both without as well as with filtering with test data (*fwt*) of models selected with (ComplEx *fwt*) and without (ComplEx) using filtering with test data for model selection.

is due to the proportion of such triples in both datasets: about 0.1% and 10% of the test set, respectively. FB15K and WN18 contain no such triples.

Handling score ties. Another issue that can cause a considerable impact in KGE evaluation is the way ranking ties are handled during evaluation. As explained in Section 2.2.6, models rank candidates for a given evaluation query based on their scores. Depending on the model, it is more or less likely that two or more candidates get the same score and thus are ranked on the same position. Sun et al. (2020) studied the effect of using different tie-handling policies, and report that using the top rank of a set of tied candidates can lead to serious overestimation of model performance and might introduce a bias for models to score candidates equally.

Recent frameworks, such as PyKEEN (Ali et al., 2021) and our own LibKGE (Broscheit et al., 2020) offer three different tie handling policies: *top* which selects the top rank in a set of tied candidates, *bottom* which selects the bottom rank, and *average* which selects the average rank in the set. **In this work, we reported model performance using the *average* tie-handling policy, which has since become common practice.** However, we evaluated our best models with all three tie-handling policies, and all results are identical, except for a few models showing differences after the fourth decimal at times.

	Model A	Model B
1st ranked prediction	Unknown triple	Validation triple
2nd ranked prediction	Validation triple	Test triple
3rd ranked prediction	Unknown triple	Test triple
4th ranked prediction	Unknown triple	Validation triple
5th ranked prediction	Unknown triple	Validation triple

Table 3.9: Example to illustrate the problem of filtering predictions with test data. Blue indicates correct predictions according to metrics on *validation* data. Red indicates incorrect predictions. If we select between two models using Hits@3 without filtering predicted test triples, both models have the same performance. But if we filter out predicted test triples, Model B performs better, so we promote the model based on its performance of test data.

Filtering with test data. It is common practice in the literature to filter out known test data triples when computing evaluation metrics on validation data (see filtered metrics in Section 2.2.6). In this thesis we follow this practice for the purpose of comparison to other works in the literature. However, we suggest that the use of test data be exclusive to the evaluation of models on unseen data. Excluding test set triples from the set of filtering triples during evaluation should have almost no impact in model performance or in model selection. To assess the impact of this practice, Table 3.8 shows the performance on validation data of ComplEx when selected with and without filtering with test data. In our tests, the selected models were always the same, and we can see that the performance difference is minimal.

To illustrate why using test data to filter results is potentially problematic, Table 3.9 shows an example of two models and their top 5 predictions on validation data. Blue indicates correct predictions according to metrics on *validation* data. Red indicates incorrect predictions. Say we use Hits@3 to select the best performing model. Without filtering out known test triples, both models perform the same, they make one correct prediction in the top 3 ranked predictions according to known validation data. However, if we filter out test triples, which are correct predictions but treated as incorrect when evaluating on validation data, Model B improves performance. Thus, **when filtering out test triples, we would select Model B as the best model, partly based on its performance on test data.**

3.3 Related Work

Our study complemented and expanded on the results of Kotnis and Nastase (2018), who focused on negative sampling, and Mohamed et al. (2019), who focused on loss functions. Since the publication of this study and its conclusions, similar studies have been motivated by our work, e.g. Ali et al. (2021), and conducted an even larger comparative study using the same implementation and experimental settings across models. The included a larger set of models and training settings than in our study. There are, however, other comparative studies, e.g. Rossi et al. (2021), which look at model performance across different publications, each with different implementations and experimental settings. Based on the results presented in this chapter, we must assume that the observations drawn from such studies may change when all models are given the same training resources.

In addition to studies that focus on KGE models, similar studies have been conducted in other areas, including language modeling (Melis et al., 2017), generative adversarial networks (Lucic et al., 2018), or sequence tagging (Reimers and Gurevych, 2017).

3.4 Summary

In this chapter, we focused on link prediction, the most commonly used task for training and evaluating KGE models. We reported the results of an extensive experimental study with popular KGE model architectures and training strategies across a wide range of hyperparameter settings. We found that when trained appropriately, the relative performance differences between various model architectures often shrunk and sometimes even reversed when compared to prior results. This suggests that training and hyperparameter optimization strategies can have a significant impact on model performance and may account for a substantial fraction of the progress reported in KGE model performance.

Based on our results, and to enable an insightful model comparison, we recommend that new studies: (i) compare models in a common experimental setup using a sufficiently large hyperparameter search space, and (ii) if necessary to compare results with other publications, use the best known performance of each model rather than the first reported one. Our study

shows that such an approach is possible with reasonable computational cost, and we provide our implementation as part of the LibKGE framework to facilitate such model comparison. Indeed, as mentioned before, some studies have followed this recommendation by conducting their experiments using LibKGE, e.g. Safavi and Koutra (2020) and Gema et al. (2023). Moreover, we believe that many of the more advanced architectures and techniques proposed in the literature need to be revisited to reassess their individual benefits.

In the next chapters, we look at some limitations in the link prediction approach commonly used in the literature both with respect to KGE models' ability to capture graph properties and their use in downstream applications.

CHAPTER

FOUR

KNOWLEDGE BASE COMPLETION

The most often cited motivation for developing KGE models is *to predict missing links in an incomplete knowledge graph*, e.g. Yu et al. (2021b); Abboud et al. (2020); Sun et al. (2019); Trouillon et al. (2016). In practice, this has almost always been done by evaluating model performance on the link prediction (LP) task, which is most commonly evaluated with the *entity ranking* (ER) protocol. As described in Section 2.2.6, this evaluation protocol takes as input a set of previously unobserved evaluation triples (typically from the validation or test splits), such as *(Einstein, bornIn, Ulm)*, and uses the embedding model to rank all possible answers to the questions *(Einstein, bornIn, ?)* and *(?, bornIn, Ulm)*. Model performance is then assessed based on the rank of the answer present in the evaluation triple (*Einstein* and *Ulm*, respectively). Since each question in ER is constructed from an existing evaluation triple, the protocol ensures that questions are *syntactically accurate*, i.e. they respect schema constraints (see Section 2.1.1), and always have a correct answer.

In this chapter, we argue that the link prediction task, and by extension the ER protocol that is most commonly used to evaluate it, is not well suited for the goal of adding missing links to an incomplete KG. To illustrate our point, we propose the task of *knowledge base completion* (KBC),[1] a generalization of the LP task that we define as follows:

[1]The terms link prediction and knowledge base completion are often used interchangeably in the literature. We define them as different tasks in this thesis.

Definition 4.0.1 (Knowledge Base Completion)**.** Given a knowledge graph $\mathcal{K} \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ and relation $k \in \mathcal{R}$, knowledge base completion is the task of inferring true missing facts about relation k, i.e. answering the query *(?, k, ?)*.

In other words, given relation *bornIn*, models should answer the question *(?, bornIn, ?)* with facts such as *(Einstein, bornIn, Ulm)*. This task is different from LP since no information about potential missing triples is provided upfront. It thus requires that a model be able to more generally make predictions about the entire KG, which as we illustrate below, is arguably a more suitable task for the incompleteness problem that motivates KGEs in the first place.

Throughout this chapter, we ground our arguments on the ER evaluation protocol as a proxy for the LP task, and argue that the ER protocol is not well-suited to assess model performance for KBC. To see this, observe that models that assign high confidence scores to syntactically inaccurate triples such as *(Ulm, bornIn, Einstein)* are not penalized by the ER protocol because the corresponding questions, e.g., *(Ulm, bornIn, ?)*, are never asked. Thus a model that performs well on ER may still not perform well on KBC. Such a model may still make incorrect predictions about a large number of link prediction queries. This is because, unlike LP where the focus is on answering meaningful questions, KBC has different goals: (1) to add missing true triples to the knowledge base and (2) to avoid adding false triples. The first point relates to recall, while the second relates to precision. A model that performs well under ER may have low precision on KBC, because it may assign high scores to a large number of false triples (e.g., nonsensical triples), despite its good performance on ER. If we used such an LP model for adding missing facts to a KB using the high-scoring triples, we would add these false triples and deteriorate precision. This is undesirable, since many KGs are constructed to be highly precise—e.g. YAGO has a precision of 95% (Suchanek et al., 2007)—and a drop in precision would not only be an inaccurate representation of the KG, it would also negatively affect downstream applications. Thus, models that assign high scores to false triples should be penalized. We expand on this discussion in Section 4.1, by showing concretely why some commonly used KGE models are inherently not well-suited for KBC, and thus not recommended for adding missing links in a KG.

To assess model performance on the KBC task, we propose the *entity-pair ranking* (PR) evaluation protocol. Given a relation such as *bornIn*, PR uses

the KGE model to rank all possible answers—i.e., all entity pairs—to the question $(?, bornIn, ?)$, and subsequently assess model performance based on the rank of the evaluation triples for relation *bornIn* in the answer. The protocol ensures that a model's performance is negatively affected if the model assigns high scores to false triples such as *(Ulm, bornIn, Einstein)*.

We conducted an experimental study presented in Section 4.3 on commonly used benchmark datasets under the ER and the PR protocols. We found that:

1. The performance of popular embedding models was often good under the ER but unsatisfactory under the PR protocol, even on "simple" datasets that are generally considered to be too easy for LP.

2. A simple rule-based model often provided superior performance for PR, suggesting that it would be a more suitable choice for adding missing facts to an incomplete KG.

3. To see whether the lower performance of embedding models was mostly due to the nonsensical, i.e. syntactically inaccurate, triples considered in PR, we applied the background knowledge of domain-range constraints to filter out many of these clearly nonsensical triples. We found that model performance on KBC indeed suffers because models make many syntactically inaccurate predictions, but that their performance is still unsatisfactory even after removing these triples during evaluation. This suggests that models also make a considerable number of *semantically inaccurate* predictions, i.e. predictions do not reflect real-world facts (see Section 2.1.1).

Overall, our findings suggest that evaluating performance on the LP task is limited in that it does not reflect a model's ability to *generally* predict missing links in a KG, and that more research is needed into embedding models as well as their training strategies for the more general task of knowledge base completion.

This chapter is organized as follows. We briefly review the ER evaluation protocols for KGE models and discuss its limitations in Section 4.1. We introduce our proposed Entity-Pair Ranking evaluation protocol for the KBC task in Section 4.2. We describe our experimental study and discuss our results in Section 4.3. We discuss related work in Section 4.4, and we summarize our findings in Section 4.5.

4.1 Predicting Missing Links

In this section, we first review the entity ranking (ER) evaluation protocol for LP (for details, see Section 2.2.6). We then argue that this protocol is not well-suited for assessing KBC performance, because it focuses on a small subset of all possible facts for a given relation, and thus does not reflect a model's ability to represent an entire KG. We then introduce the entity-pair ranking (PR) protocol in Section 4.2 and discuss its advantages and potential shortcomings. We assume throughout that only true but no false triples are available (as is commonly the case), and that the available true triples are divided into training, validation, and test triples.

Entity ranking (ER). ER assesses model performance by testing its ability to perform LP. In particular, for each evaluation triple $t = (i, k, j)$, two questions $q_s = (?, k, j)$ and $q_o = (i, k, ?)$ are generated. For question q_s, all entities $i' \in \mathcal{E}$ are ranked based on the score $s(i', k, j)$. To avoid misleading results, entities $i' \neq i$ that correspond to observed triples in the dataset—i.e., (i', k, j) occurs in the training/validation/test triples—are discarded to obtain a *filtered ranking*. The same process is applied for question q_o. Model performance is evaluated based on the recorded positions of the evaluation triples in the filtered ranking. Models that tend to rank evaluation triples (known to be true) higher than unknown triples (assumed to be false) are thus considered superior. Usually, the micro-average of *filtered Hits@K*—i.e., the proportion of evaluation triples ranking in the top-K—and *filtered MRR*—i.e., the mean reciprocal rank of the evaluation triples—are used to assess model performance.

Discussion. To better illustrate why ER, and thus the LP task it is based on, can lead to a misleading assessment of a model's KBC performance, consider the DistMult model and the asymmetric relation *nominatedFor*. As described in Section 2.2.5, DistMult models all relations as symmetric, i.e. $s(i, k, j) = s(j, k, i)$ for any given triple (i, k, j). Now consider triple $t =$ *(H. Simon, nominatedFor, Nobel Prize)*, and let us suppose that the model correctly assigns t a high score $s(t)$. Then the inverse triple $t' =$ *(Nobel Prize, nominatedFor, H. Simon)* will also obtain a high score since $s(t') = s(t)$. Thus the score produced by DistMult does not discriminate between the true triple t and the false triple t'. In ER, however, questions about t' are never asked; there is no evaluation triple for this relation containing either *Nobel Prize* as subject or *H. Simon* as object. The symmetry

of DistMult's predictions thus barely affects its performance under the ER protocol.

For another example, consider TransE and the relation $k = \textit{marriedTo}$, which is symmetric but not reflexive. One can show that for all (i, k, j), the TransE scores satisfy

$$\begin{aligned} s(i,k,j) + s(j,k,i) &= -\|\boldsymbol{e}_i + \boldsymbol{r}_k - \boldsymbol{e}_j\| - \|\boldsymbol{e}_j + \boldsymbol{r}_k - \boldsymbol{e}_i\| \\ &\leq -\|\boldsymbol{e}_i + \mathbf{0} - \boldsymbol{e}_j\| - \|\boldsymbol{e}_j + \mathbf{0} - \boldsymbol{e}_i\|. \end{aligned}$$

For symmetric relations, TransE should aim at assigning high scores to both (i, k, j) and (j, k, i). To do so, TransE has the tendency to push the relation embedding $\boldsymbol{r}_k$ towards $\mathbf{0}$ as well as $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ towards each other. But when $\boldsymbol{r}_k \approx \mathbf{0}$, then $s(i, k, i)$ is high for all i, so that the relation is treated as if it were reflexive. We show empirical evidence for this in Section 4.3.2. Again, in ER, this property only slightly influences the results: there is only one "reflexive" tuple in each filtered entity list so that the correct answer i for question $(?, k, j)$ ranks at most one position lower.

More expressive models such as RESCAL or ComplEx do not have such inherent limitations. Nevertheless, our experimental study shows that these models also tend to assign high scores to false triples.

4.2 Entity-Pair Ranking Protocol

We propose an alternative evaluation protocol called entity-pair ranking (PR), described by Algorithm 4.1. The protocol is designed to be a more suitable method to assess a model's performance on the KGC task, although it does come with challenges, which we discuss below. PR proceeds as follows: for each relation k, we use the KGE model to rank all triples for a specified relation k, i.e., to rank all answers to question $(?, k, ?)$. As in ER, we filter out all triples that occur in the training and validation data to obtain a filtered ranking, i.e., to only rank triples that were not used during model training. If a model tends to assign a high score to negative triples, its performance is likely to be negatively affected because it becomes harder for true triples to rank high. Figure 4.1 shows the contrast between the number of candidate triples considered for PR and those considered for ER.

Note that the number of candidate answers considered by PR is much

Algorithm 4.1: Entity-Pair Ranking (PR) Evaluation Protocol

Require: $\mathcal{T}$: set of evaluation triples,
$\mathcal{E}$: set of entities in knowledge graph $\mathcal{K}$
Ensure: Aggregated metrics m_{all} of all evaluation triples

$m_{\text{all}} \leftarrow 0$
foreach unique k in triples $(i, k, j) \in \mathcal{T}$ **do**
  $C \leftarrow |\mathcal{E} \times \mathcal{E}|$ candidates (i', k, j') with every $(i', j') \in \mathcal{E} \times \mathcal{E}$
  $S \leftarrow \textsc{Scores}(C)$
  $L \leftarrow$ label known positives in C, remaining candidates negative
  $R \leftarrow \textsc{Ranks}(S, L)$ `// compute ranks of known positives`
  $m \leftarrow \textsc{Compute\_Metric}(R)$
  $w \leftarrow \textsc{Weight}(k)$ `// compute weight of k`
  $m_{\text{all}} \leftarrow m_{\text{all}} + wm$

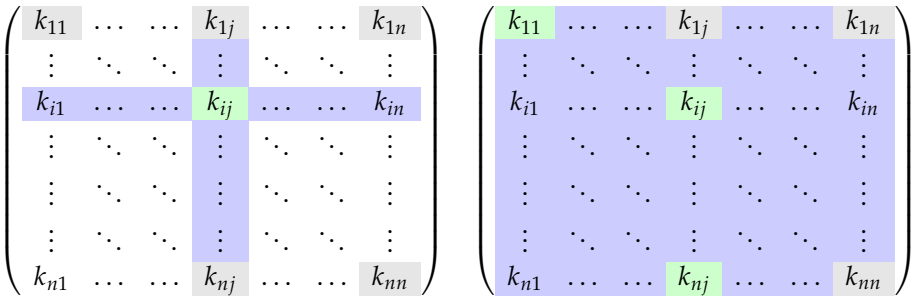
larger than those considered by ER. Consider a relation k and let $\mathcal{T}_k$ be the set of evaluation triples for relation k. Then ER considers $2|\mathcal{T}_k|\,|\mathcal{E}|$ candidates in total during evaluation, while PR considers $|\mathcal{E}|^2$ candidates. Moreover, PR considers all evaluation triples in $\mathcal{T}_k$ simultaneously instead of sequentially. For this reason, we do not rely on the MRR metric commonly used in ER. Instead, we assess model performance using weighted *MAP@K*—i.e., the weighted mean average precision in the top-K filtered results—and weighted *Hits@K*—i.e., the weighted percentage of evaluation triples in the top-K filtered results. We weight the influence of relation k proportionally to its number of evaluation triples (capped at K), thereby closely following ER:

$$\text{MAP@}K = \sum_{k \in \mathcal{R}} \text{AP}_k\text{@}K \times \frac{\min(K, |\mathcal{T}_k|)}{\sum_{k' \in \mathcal{R}} \min(K, |\mathcal{T}_{k'}|)}$$

$$\text{Hits@}K = \sum_{k \in \mathcal{R}} \text{Hits}_k\text{@}K \times \frac{\min(K, |\mathcal{T}_k|)}{\sum_{k' \in \mathcal{R}} \min(K, |\mathcal{T}_{k'}|)}.$$

Here $\text{AP}_k\text{@}K$ is the average precision of the top-K list (w.r.t. evaluation triples $\mathcal{T}_k$) and $\text{Hits}_k\text{@}K$ refers to the fraction of evaluation triples in the top-K list. Note that K should be chosen much larger for PR than for ER since it roughly corresponds to the number of triples we aim to predict for relation k.

The PR protocol is more suited to evaluate KBC performance because it considers all model predictions. However, the protocol also has some



(a) Entity Ranking **(b)** Entity-Pair Ranking

Figure 4.1: Candidate entities considered by different evaluation protocols for a given query involving relation k. Green cells are target entities, blue cells are candidate entities, gray cells are filtered out (observed) entities. **(a)** Entity Ranking (ER): given query $(k_{i1}, k, ?)$, ER considers all candidates in the same column and row as target entity k_{ij}. **(b)** Entity-Pair Ranking (PR): given query $(?, k, ?)$, PR considers all possible candidate triples for relation k.

disadvantages. First, as ER, the PR protocol may underestimate model performance due to unobserved true triples ranked high by the model. Since a larger number of candidates is considered, PR may be more sensitive to this problem than ER. We explore the effect of underestimation in our empirical study in Sec. 4.3.3. Another concern with PR is its potentially high computational cost. For current benchmark datasets, we found that the PR evaluation is feasible. Generally, one may argue that an embedding model is suitable for KBC only if it is feasible to determine high-scoring triples in a sufficiently efficient way. Since PR only requires the computation of the top-K predictions, performance can potentially be improved using techniques such as maximum inner-product search (Shrivastava and Li, 2014). In practice, we found that PR is 3–4 times more expensive than ER, but with GPU-ready implementation, PR costs up to a few minutes in the worst case.

Dataset	Entities	Relations	Training	Validation	Test
FB15K	14 951	1 345	483 142	50 000	59 071
WN18	40 943	18	141 442	5 000	5 000
FB-237	14 505	237	272 115	17 535	20 466
WNRR	40 559	11	86 835	3 034	3 134

Table 4.1: Statistics of datasets used in this study.

4.3 Experimental Study

We conducted an experimental study with the goal of assessing the performance of various KGE models for KBC. As part of our study, we investigated the extent to which PR underestimates model performance due to unobserved true triples, and the impact that filtering out syntactically inaccurate predictions have on model performance. This experimental study was done prior to the work presented in Chapter 3. Thus, some experimental settings do not follow the guidelines proposed in that study, and the resulting performance in link prediction is below what is reported in Section 3.2.2. However, the results in this study are still relevant, as we show in Section 4.3.4 where we reproduce some of these results using the experimental settings from our study in Section 3.2.

4.3.1 Experimental Settings

Datasets. We used four common KBC benchmark datasets: FB15K, WN18, FB15K-237 (referred to as FB-237 for brevity), and WNRR. The first two are subsets of WordNet and Freebase, respectively (Bordes et al., 2013b), and it is known that the regularities in the datasets can be explained via simple implication rules (Dettmers et al., 2018; Meilicke et al., 2018). We will see that despite their simplicity, these datasets can still shed some light on the behavior of various models under PR. FB-237 and WNRR are constructed from FB15K and WN18 respectively, to make the datasets more challenging (Toutanova and Chen, 2015), and model performance is indeed considerably lower on these datasets (Akrami et al., 2018). We describe these datasets in more detail in Section 2.4. Key dataset statistics are summarized in Table 4.1.

Models. We chose five commonly used KGE models: RESCAL (Nickel

et al., 2011), Transe (Bordes et al., 2013b), DistMult (Yang et al., 2015), ComplEx(Trouillon et al., 2016), and Analogy(Liu et al., 2017). Some other KGE models do not support KBC directly due to their architecture. For example, ConvE (Dettmers et al., 2018) uses reciprocal relations to make object predictions, but KBC requires a single embedding for relations. We also considered a simple rule-based system called *RuleN* (Meilicke et al., 2018) to contrast against KGE models. We describe RuleN in more detail in Section 2.3.3.

Evaluation. For all models, we performed evaluation under both the ER and PR protocols in order to assess their performance for the LP and KBC tasks, respectively. For ER, we report MRR and Hits@10 (also denoted by H@10 for brevity). For PR, we report MAP@100 and Hits@100 (also denoted by M@100 and H@100, respectively).

Negative sampling. We trained the embedding models with negative sampling (see Section 2.2.7). Since we are testing performance on the more general KBC task, we consider three sampling strategies in our experiments:
Corrupt-1: For each training triple $t = (i, k, j)$, sample pseudo-negative triples by randomly replacing either i or j with a random entity (but such that the resulting triple is unobserved). This ensures that at least one entity is observed in the training data of relation k. This strategy matches ER, which is based on questions $(?, k, j)$ and $(i, k, ?)$, and is the most common approach to negative sampling in the literature.
Corrupt-1R: For each training triple $t = (i, k, j)$, sample pseudo-negative triples by randomly replacing either i, k or j. This results that some negative examples potentially belonging to a different relation. This approach to generate negatives is generally less adopted, especially when studying the link prediction task. The generated negative samples are not compared with the training set as done in some studies, e.g. (Liu et al., 2017).
Corrupt-2: For each training triple $t = (i, k, j)$, obtain pseudo-negative triples by randomly replacing both i and j, each with a random entity. This method appears more suited to PR, as it includes triples during training that belong to the same relation, but are potentially type-inconsistent.

Training and implementation. We trained DistMult, ComplEx, Analogy and RESCAL with AdaGrad (Duchi et al., 2011) using binary cross-entropy loss. We used pair-wise ranking loss for TransE (as it always produces negative scores). All embedding models are implemented on top of the code

of Liu et al. (2017)[2] in C++ using OpenMP. For RuleN, we use the original implementation provided by the authors. The evaluation protocols were written in Python, with Bottleneck[3] used for efficiently obtaining the top-K entries for PR. We found PR (which took ≈30–90 minutes) was about 3–4 times slower than ER. We discuss in Section 4.3.4 how using more advance training settings proposed after this publication yields generally the same results.

Hyperparameters. The best hyperparameters were selected based on MRR for ER and MAP@100 for PR on the validation data. For both protocols, we performed an exhaustive grid search over the following hyperparameter settings: $d_e \in \{100, 150, 200\}$, weight of l_2-regularization $\lambda \in \{0.1, 0.01, 0.001\}$, learning rate $\eta \in \{0.01, 0.1\}$, negative sampling strategies *Corrupt-1*, *Corrupt-2* and *Corrupt-1R*, and margin hyperparameter $\gamma \in \{0.5, 1, 2, 3, 4\}$ for TransE. For each training triple, we sampled 6 pseudo-negative triples. To reduce cost, we only used the most frequent relations from each dataset for model evaluation on validation data (top-5, top-5, top-15, and top-30 most frequent relations for WN18, WNRR, FB-237 and FB-15K, respectively). We trained each model for up to 500 epochs during grid search. In all cases, we evaluated model performance every 50 epochs and used the overall best-performing model. For RuleN, we used the best settings reported by the authors for ER (Meilicke et al., 2018). That is, for FB15K and FB-237, we learned path rules of length 2 and constant rules, and the sampling size was set to 1000. For WN18 and WNRR, we learned path rules of length 2 with sampling size of 1000, and path rules of length 3 with sampling size of 100, as well as constant rules. For PR, we learned path rules of length 2 using a sampling size of 500 for FB15K and FB-237. For WN18 and WNRR, we learned path rules of length 3 and sampling size of 500. As mentioned above, Section 4.3.4 shows that using a larger search space such as the one used in Section 3.2.1 yields generally the same results and is not always beneficial for model performance.

4.3.2 Model Performance

Entity ranking. Table 4.2 summarizes the ER results. ComplEx and Analogy outperform other models on FB-237 and WN18, but are sometimes outper-

[2] https://github.com/quark0/ANALOGY
[3] https://pypi.org/project/Bottleneck/

	FB15K		FB-237		WN18		WNRR	
	MRR	H@10	MRR	H@10	MRR	H@10	MRR	H@10
DistMult	.660	**.845**	.270	.432	.790	.937	.432	.474
TransE	.500	.777	**.290**	**.466**	.720	.908	.220	**.491**
ComplEx	**.700**	.835	.280	.435	.940	**.948**	**.440**	.481
Analogy	**.700**	.836	.270	.433	**.941**	.942	**.440**	.486
RESCAL	.464	.699	.270	.427	.920	.939	.420	.447
RuleN	.805	.870	.260	.420	.950	.958	–	.536

Table 4.2: Results with the entity ranking protocol (ER), which assesses LP performance. We report test data MRR and Hits@10 (H@10). Bold entries show best KGE model performance per dataset.

formed by DistMult and TransE on other datasets, the latter two of which are known to be less expressive (Wang et al., 2018b). In particular, although DistMult can only model symmetric relations, and although most relations in these datasets are asymmetric, DistMult has good ER performance in general. Likewise, TransE achieved great performance in Hits@10 on all datasets, including WN18 which contains a large number of symmetric relations, a type of relation that is not easily modeled by TransE, as discussed in Section 4.1. When compared to the rule-based baseline, embedding models perform competitively with respect to RuleN on all datasets, except for their MRR performance on FB15K. This holds in general even for the more restricted models (TransE and DistMult) on the more challenging datasets, which were created after criticizing FB15K and WN18 as too easy (Toutanova and Chen, 2015; Dettmers et al., 2018). Note that similar observations can be made about the results in Section 3.2.2 in the previous chapter. In fact, in Table 3.4, DistMult is not only competitive, but even outperforms all other models on the FB15K dataset. **In short, link prediction performance as reported by the ER protocol shows that less expressive models like DistMult and TransE are competitive with, or even outperform, more expressive models like ComplEx.**

Entity-pair ranking. The evaluation results of PR with $K = 100$ are summarized in Table 4.3. Note that Tables 4.2 and 4.3 are not directly comparable: they measure different tasks using different metrics. Also note that we use a different value of K, which in PR corresponds to the number of

Dataset	FB15K		FB-237		WN18		WNRR	
Model	M@100	H@100	M@100	H@100	M@100	H@100	M@100	H@100
DistMult	.013	.104	.030	.042	.079	.097	.141	.178
TransE	.211	.363	**.079**	**.176**	.223	.315	.020	.013
ComplEx	**.311**	**.486**	.071	.166	**.825**	**.904**	**.168**	**.200**
Analogy	.188	.348	.049	.143	.776	.874	.154	.198
RESCAL	.150	.303	.067	.150	.482	.609	.131	.138
RuleN	.774	.837	.076	.158	.948	.968	.215	.251

Table 4.3: Results with the entity-pair ranking protocol (PR), which assesses KBC performance. We report test data MAP@100 and Hits@100 (H@100). Bold entries show best KGE model performance per dataset.

predicted facts per relation. We discuss the effect of the choice of K later.

For the embeddings, we observe that **with the exception of Analogy and ComplEx on WN18, the performance of all models is unsatisfactory on all datasets, especially when compared with RuleN on FB15K and WN18, which were previously considered to be too easy for embedding models.** Specifically, DistMult's Hits@100 is slightly less than 10% on WN18, meaning that if we add the top 100 ranked triples to the KB, over 90% of what is added is likely false. Even when using ComplEx, the best model on FB15K, we would potentially add more than 50% false triples. This implies that embedding models cannot make predictions about the graph more generally than what is implied by the LP task. The notable exceptions are ComplEx and Analogy on WN18, although both are still behind RuleN. TransE and DistMult did not achieve competitive results on WN18. In addition, DistMult did not achieve competitive results on FB15K and FB-237 and TransE did not achieve competitive results in WNRR. In general, ComplEx and Analogy performed consistently better than other models across different datasets. When compared with the RuleN baseline, however, the performance of these models was often not satisfactory. This suggests that better KGE models and/or training strategies are needed for KBC.

RuleN did not perform well on FB-237 and WNRR, likely because the way these datasets were constructed makes them intrinsically difficult for rule-based methods (Meilicke et al., 2018). This is reflected in both ER and PR results.

	Model									
Relation	DistMult		TransE		ComplEx		Analogy		RESCAL	
hyponymy	1	(1)	18	(32)	99	(99)	99	(99)	92	(93)
hypernymy	0	(0)	5	(33)	99	(99)	99	(99)	96	(98)
deriv. related form	100	(100)	0	(0)	100	(100)	100	(100)	6	(68)
member meronym	0	(0)	18	(41)	74	(84)	83	(85)	44	(63)
member holonym	0	(0)	16	(47)	74	(83)	83	(85)	37	(54)

Table 4.4: Number of test triples in the top-100 filtered predictions on WN18. An estimate of the number of true triples in the top-100 list is given in parentheses.

Top 100 predictions. To better understand the change in PR performance of TransE and DistMult compared to their ER performance, we investigated their predictions for the top-5 most frequent relations on WN18. Table 4.4 shows the number of test triples appearing in the top-100 predictions for each relation after filtering triples from the training and validation sets. The numbers in parentheses are discussed in Section 4.3.3.

We found that DistMult worked well on the symmetric relation *derivationally related form*, because all 100 top predictions it makes are triples from the test set. This is where its symmetry assumption clearly helps, because 93% of the training data consists of symmetric pairs (i.e., (i,k,j) and (j,k,i)), and 88% of the test triples have its symmetric counterpart in the training set. In contrast, TransE contained no test triples for *derivationally related form* in the top-100 list. We found that the norm of the embedding vector of this relation was 0.1, which was considerably smaller than for the other relations (avg. 1.4). This supports our argument that TransE tends to push symmetric relation embeddings to $\mathbf{0}$.

Note that while *hyponymy*, *hypernymy*, *member meronym* and *member holonym* are semantically transitive, the dataset contains almost exclusively their transitive core, i.e., the dataset (both train and test) does not contain many of the transitive links of the relations. As a result, models are not exposed to the transitive property of these relations during training. This might explain why models that cannot handle transitivity well may still produce good results, and more specifically, why TransE performed better for these relations than for *derivationally related form*. DistMult did not perform

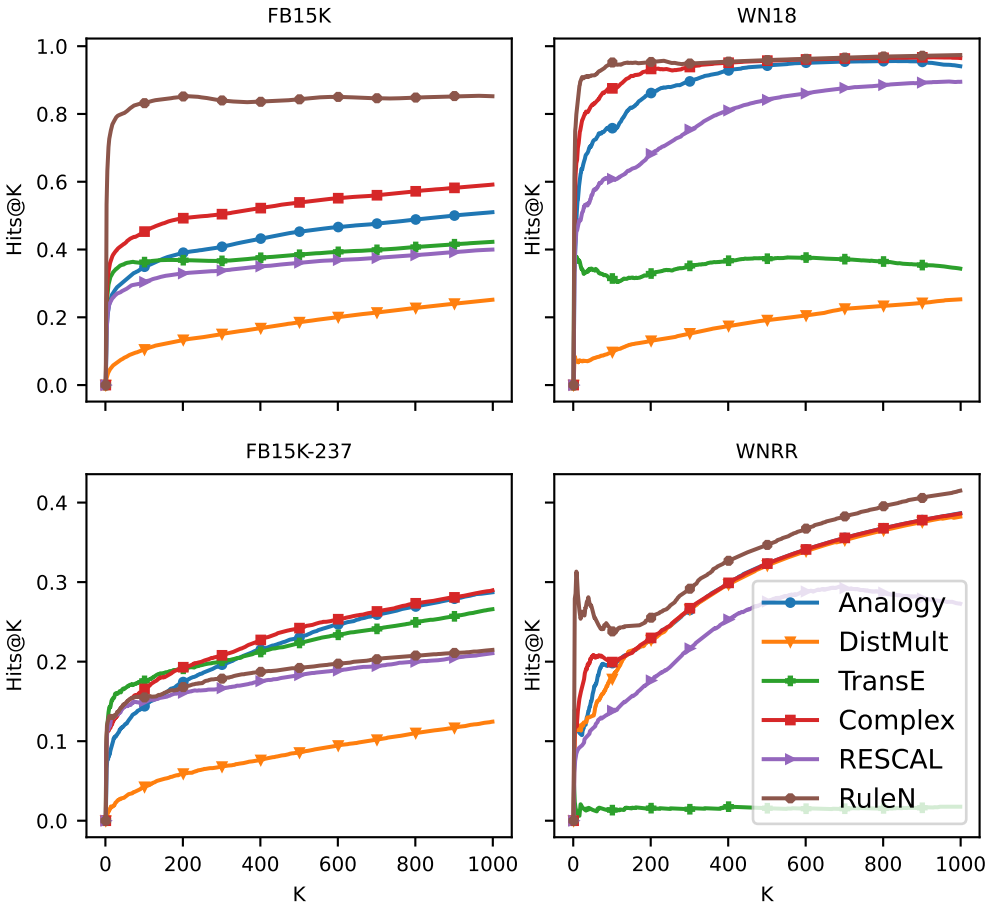


Figure 4.2: Hits@K with PR as a function of K

well on these relations (they are asymmetric). ComplEx and Analogy showed superior performance across all relations. RESCAL is in between, most likely due to difficulties in finding a good parameterization. However, it is unclear to us why TransE performed well on FB15K and FB-237.

In general, these results suggest that the limitations of DistMult and TransE do have an impact on their performance on the KBC task, as measured by the PR evaluation protocol.

Choice of K. To investigate model performance in PR for different values of K, we give the curves of Hits@K as a function of K for all datasets in Figure 4.2. Curves of MAP@K are given in Appendix B. ComplEx and

Analogy, which are universal models in terms of expressive power (see Section 2.2.5), performed best for large K compared to other embedding models. Similarly, TransE performs the best for small values of K on FB15K and FB-237. Note that RuleN performs considerably better on FB15K, WN18 and WNRR, while it still performs competitively on FB-237. In general, **all of the results discussed above hold for all considered values of K.**

4.3.3 Underestimation and Type Filtering

Influence of Unobserved True Triples. Since all datasets are based on incomplete knowledge bases, all evaluation protocols may systematically underestimate model performance. In particular, any true triple t that is neither in the training, nor validation, nor test data is treated as negative during ranking-based evaluations. A model which correctly ranks t high is thus penalized. PR might be particularly sensitive to this due to the large number of candidates considered.

It is generally unclear how to design an automatic evaluation strategy that avoids this problem. Manual labeling can be used to address this, but it may sometimes be infeasible given the large number of relations, entities, and models for KBC. It might even require expert knowledge.

To explore such underestimation effect in PR, we decoded the unobserved triples in the top-100 predictions of the 5 most frequent relations of WN18, by computing the symmetric/transitive closure of the available data in the entire WordNet knowledge base. In Table 4.4, we give the resulting number of triples in parentheses (i.e., number of test triples + implied triples). **We observed that underestimation indeed happened. TransE was mostly affected, but still did not lead to competitive results when compared to ComplEx and Analogy.** While not included in Table 4.4, RuleN achieves the best possible results in all 5 relations. That is, all 100 top prediction it makes are true triples by our estimation. These results suggest that (1) underestimation is indeed a concern, and (2) the results in PR can nevertheless give an indication of relative model performance.

Type Filtering. When background knowledge is available, embedding models only need to score triples consistent with this background knowledge. We explored whether their performance can be improved by filtering out type-inconsistent triples from each model's predictions. Note that this is inherently what rule-based approaches do, since all predicted candidates will

	Model	MAP@K (%)		Hits@K (%)	
FB15K	DistMult	.188	(+.175)	.364	(+.260)
	TransE	.257	(+.045)	.417	(+.054)
	ComplEx	.531	(+.220)	.696	(+.210)
	Analogy	.413	(+.225)	.615	(+.267)
	RESCAL	.167	(+.017)	.328	(+.025)
	RuleN	.774	(.000)	.837	(.000)
FB-237	DistMult	.095	(+.092)	.181	(+.139)
	TransE	.113	(+.034)	.212	(+.036)
	ComplEx	.113	(+.042)	.218	(+.052)
	Analogy	.105	(+.056)	.209	(+.066)
	RESCAL	.102	(+.035)	.190	(+.040)
	RuleN	.76	(.000)	.158	(.000)

Table 4.5: Results with PR using type filtering (K = 100).

be type-consistent. In particular, we investigated how model performance is affected when we filter out predictions that violate type constraints (domain and range of each relation). If a model's performance improves with such type filtering, it must have ranked syntactically inaccurate tuples high in the first place. We can thus assess to what extent models capture entity types as well as the domain and range of the relations.

We extracted from Freebase[4] type definitions for entities and domain and range constraints for relations. We also added the domain (or range) of a relation k to the type set of each subject (or object) entity which appeared in k. We obtained types for all entities in both FB datasets, and domain/range specifications for roughly 93% of relations in FB15K and 97% of relations in FB-237. The remaining relations were evaluated as before. Since WordNet contains no type or domain/range information, we focused on the Freebase-derived datasets for this experiment.

We report in Table 4.5 the Hits@100 and MAP@100 as well as their absolute improvement (in parentheses) w.r.t. Table 4.3. We also include the results of RuleN from Table 4.3, which are already type-consistent. **The results show that all KGE models improve with type filtering; thus all models make**

[4] https://developers.google.com/freebase/

	FB-237		WN18		WNRR	
	M@100	H@100	M@100	H@100	M@100	H@100
DistMult	.030	.042	.079	.097	.141	.178
DistMult LibKGE	.002	.027	.084	.104	.141	.149
ComplEx	.071	.166	**.825**	**.904**	.168	.200
ComplEx LibKGE	.111	.191	.794	.836	**.204**	**.270**
RESCAL	.067	.150	.482	.609	.131	.138
RESCAL LibKGE	**.124**	**.209**	.702	.767	.200	.240
RuleN	.260	.420	.950	.958	–	.536

Table 4.6: Comparison of Entity-Pair Ranking (PR) performance between models in Table 4.3 (including RuleN) and an implementation with LibKGE that followed the same experimental settings as in Section 3.2.1. We report test data MAP@100 and Hits@100 (H@100). Bold entries show best performance per dataset. Results from RuleN are included again for reference.

syntactically inaccurate predictions, specifically, triples where the entity types do not match the relations domain/range restrictions. In particular, DistMult shows considerable improvement on both datasets. Indeed, about 90% of the relations in FB15K (about 85% for FB-237) have a different type for their domain and range. As DistMult treats all relations as symmetric, it introduces a wrong triple for each true triple into the top-K list on these relations; type filtering allows us to ignore these wrong tuples. This is also consistent with DistMult's improved performance under ER, where type constraints are implicitly used since only questions with correct types are considered. Interestingly, **ComplEx and Analogy improved considerably on FB15K, suggesting that the best performing embedding models on this dataset are still making a considerable number of type-inconsistent predictions.** On FB15K, the relative ranking of the models with type filtering is roughly equal to the one without type filtering. On the harder FB-237 dataset, all models now perform similarly. Note that when compared with RuleN, embedding models are still behind on FB15K, but are no longer behind on FB-237.

4.3.4 Reproduction with LibKGE

To test the impact of using some of the more advanced training strategies that were proposed after this study was published, we reimplemented the PR evaluation protocol in LibKGE, our open source framework introduced in Chapter 3. We then evaluated PR performance of the best models selected in our study in Section 3.2. The results are reported in Table 4.6.

We found that the models implemented with LibKGE, which used more advanced training strategies, yielded considerably better performance for the most part, with a few exceptions: ComplEx performs worse on WN18, and DistMult is worse in FB-237, comparable in WNRR and slightly better in WN18. This shows that more advanced training strategies that are successful for link prediction are not always beneficial for PR performance.

More generally, **the reproduced results using LibKGE show that the takeaways from previous sections with respect to the KBC task still hold.** In particular, DistMult is considerably weaker than other models in PR, in contrast to its competitive performance in ER as reported in Tables 3.4 and 4.2. This suggests that PR evaluation is better suited for the KBC task, a task that is better aligned with the main motivation behind the design of KGE models: to add missing facts to incomplete KGs. In addition, results show that PR performance for all models is unsatisfactory, especially when compared to RuleN, which still far outperforms KGE models on the KBC task. This suggests that new models and/or training strategies are needed to get better performance in KBC.

In addition, one advantage that the LibKGE implementation provides is that runtimes are much lower compared to our original implementation, with PR evaluation taking between one and three minutes depending on model and dataset. The difference in cost compared to ER is still as reported before, with PR being 3–4 times slower than ER.

4.4 Related Work

One line of related work involves studies that focus on a different form of KGE evaluation. In early (and rarely in recent) work, KGE models were evaluated using *triple classification* (Socher et al., 2013; Wang et al., 2014; Lin et al., 2015; Wang et al., 2022b). As discussed in Section 2.2.6, we do

not consider it in our experiments, as it is typically overly optimistic and misleading unless hard negatives are used (Safavi and Koutra, 2020). A more recent proposal comes from Chang et al. (2020), who evaluated KGE models on the relation prediction task. We also consider this task as one of the evaluation tasks in Chapter 5.

Another line of related work are studies about probing KGE models to better understand their performance (Widjaja et al., 2022; Allen et al., 2021; Rim et al., 2021; Meilicke et al., 2018). These studies focus on a model's link prediction performance on relations with different types of properties, e.g. symmetric. In contrast, our focus is on the more general KBC task, which is designed to measure a model's ability to make predictions about a KG more generally that the standard link prediction task.

4.5 Summary

In this chapter, we showed empirically that the focus on link prediction as a task to assess KGE performance is limited in that it does not encourage models to *generally* predict missing links in a KG. We illustrated this with examples that show that models with limited ability for predicting links in a KG still perform competitively or even outperform state-of-the-art models on the link prediction task, as commonly defined in the literature. We then proposed the more general task of knowledge base completion (KBC), which we argue is more suitable for the goal of predicting missing links in a KG. We designed a corresponding entity-pair ranking (PR) evaluation protocol to assess model performance on the KBC task. This protocol evaluates a model's ability to predict links in a graph more generally than what is required for the link prediction task.

We conducted a large experimental study with commonly used KGE models and various benchmark datasets. We found that good link prediction performance, as measured by the entity ranking evaluation protocol, is not indicative of a model's ability to generally predict missing links in a KG. This suggests that standard KGE models are not generally suitable for adding missing links to a KG.

In the next chapter, we propose new training methods that go beyond the link prediction task to test whether KGE models can more generally capture multiple KG properties simultaneously.

CHAPTER

FIVE

GRAPH-STRUCTURE PREDICTION

KGE models are often described as representations that capture semantic properties of the entities in a KG (Bordes et al., 2011, 2013b; Nickel et al., 2015; Wang et al., 2017; Ji et al., 2021). And indeed, KGE models have been used as pre-trained representations to inject the structured knowledge encoded in a KG into various downstream applications, such as question answering systems (Ilyas et al., 2022) and recommender systems (El-Kishky et al., 2022; Wang et al., 2018a), among others.

At the same time, most of the literature in KGE models has focused on the link prediction (LP) task, which is commonly defined as answering questions such as *(Austin, capitalOf, ?)* by reasoning over an incomplete KG. As described in Chapter 2, this focus on link prediction has influenced the design of the standard training and evaluation methods. Consequently, the question of whether KGE models provide generally useful representations—i.e. representations that provide good results beyond link prediction—remains largely open. To our knowledge, no previous work has studied the impact that different training approaches have on the learned KG embeddings in any way other than by assessing performance on link prediction.

In this chapter, we study whether out-of-the-box KGE models capture general properties of the graph. We do this by evaluating their performance on a new set of basic *graph-structure prediction* tasks that are similar to link prediction, such as predicting the relation of a triple (e.g., the relationship between *Austin* and *Texas*), the domain and range of a relation (e.g., whether

Austin is a capital), as well as entity and relation neighborhood of each entity (e.g., which other entities are related to *Austin*). These tasks differ from the knowledge graph completion (KBC) task introduced in Chapter 4 in that these are different types of predictions about a graph, whereas KBC is a generalization of link prediction that includes all of the KG.

To evaluate a model on this set of tasks, we generalized the entity ranking protocol commonly used for link prediction to include any number of user-defined tasks. We refer to this approach as multi-task ranking (MTR). We found that the best performing models on LP where never the best performing models for MTR and that the relative performance of models changed considerably between the different tasks. This suggests that models that are good at LP are not necessarily those that capture more properties about the graph, challenging the intuition that KGE models preserve the structure of a KG.

As a result of this observation, and inspired by multi-task approaches in other areas—such as natural language processing (Aribandi et al., 2022; Sanh et al., 2022) or computer vision (Doersch and Zisserman, 2017)—, we included the graph-structure prediction tasks discussed above as additional training objectives and used MTR as evaluation measures during model selection. Specifically, we generalized the standard training approach based on link prediction to a multi-task training objective (MTT), which can be used with any KGE model class and without a substantial increase in computational cost. The resulting multi-task KGE models had significantly better overall performance across all graph-structure prediction tasks, suggesting that the learned representations capture more information about the graph, at the cost of a minor decrease in link prediction performance. This is an indication that the focus on the link prediction task is too narrow for training KGE models so that they generally capture the properties of a KG.

We begin this chapter by introducing our proposed set of graph-structure prediction tasks in Section 5.1. We introduce the multi-task ranking evaluation in Section 5.2. In Section 5.3 we describe our proposed multi-task training approach. We describe the settings and discuss the findings of our experimental study in Section 5.4. We discuss related work in Section 5.5 and summarize our contributions in Section 5.6.

Knowledge graph	Task	Example query	Some answers
(Dallas, locatedIn, Texas)	Link (LP)	*(Austin, locatedIn, ?)*	*Texas, USA*
(Texas, locatedIn, USA)		*(?, locatedIn, Texas)*	*Austin*
(Austin, capitalOf, Texas)	Relation (REL)	*(Austin, ?, Texas)*	*locatedIn*
(Austin, locatedIn, Texas)	Domain (DOM)	*(*, locatedIn, ?)*	*Texas, USA*
(Arkansas, borders, Texas)		*(?, locatedIn, *)*	*Dallas, USA*
(USA, locatedIn, North A.)	Entity	*(Austin, *, ?)*	*Texas, USA*
(Austin, locatedIn, USA)	neighb. (NBE)	*(?, *, Texas)*	*Arkansas*
	Relation	*(Austin, ?, *)*	*capitalOf*
	neighb. (NBR)	*(*, ?, Texas)*	*borders*

Table 5.1: Graph-structure prediction tasks used for self-supervised pre-training and evaluation along with example queries. Here ? denotes the prediction target and $*$ acts as a wildcard.

5.1 Graph-Structure Tasks

The new set of tasks we designed to explore the suitability of KGE models for basic graph-structure prediction tasks is summarized and exemplified in Table 5.1. We describe the form of the *queries* for each task as a triple such as $(s, ?, *)$, where s or o denote input entities, p denotes an input relation, ? denotes the prediction target, and $*$ acts as a wildcard. Using this notation, we consider the following tasks and queries:

- **Link prediction** (LP): Given a relation and a subject, predict the object (denoted $(s, p, ?)$). Likewise, given a relation and an object, predict the subject (denoted $(?, p, o)$).
- **Relation prediction** (REL, Chang et al. (2020); Chen et al. (2021c): Given two entities s and p, predict the relation between them (denoted $(s, ?, o)$).
- **Domain prediction** (DOM): Given a relation, predict its domain (denoted $(?, p, *)$) or its range (denoted $(*, p, ?)$).
- **Entity neighborhood prediction** (NBE): Given a subject entity, predict related objects (denoted $(s, *, ?)$). Likewise, given an object, predict related subjects (denoted $(?, *, o)$).

- **Relation neighborhood prediction** (NBR): Given a entity, predict the relations where it occurs as subject (denoted $(s, ?, *)$) and where it occurs as object (denoted $(*, ?, o)$).

Note that we use the wildcard to denote existential quantification. For example, given a ground-truth KG $\mathcal{G}$ and domain prediction query $(?, p, *)$, an entity $s \in \mathcal{E}$ is a correct answer if there exists an entity $o \in \mathcal{E}$ such that $(s, p, o) \in \mathcal{G}$.

We chose this particular set of tasks because they are simple, they capture basic information about the graph structure beyond link prediction, and they only have one prediction target (an entity or a relation). The latter property allows efficient pre-training and evaluation, as discussed below. For this reason, we exclude tasks such as KBC (see Section 4 (denoted $(?, p, ?)$) or reconstruction (Nickel et al., 2011) (denoted $(?, ?, ?)$). This allowed us to include some datasets that are larger than those used in the experimental studies in previous chapters.

5.2 Multi-Task Ranking Protocol

To evaluate the performance of KGE models on the graph-structure prediction tasks, we generalize the entity ranking (ER) protocol for link prediction to a multi-task ranking (MTR). Intuitively, for each of the nine queries (REL as well as LP/DOM/NBE/NBR for both subject and object targets), we construct from each evaluation triple (s, p, o), the queries corresponding to each task as given in Table 5.1. We then obtain, for each query, a ranking of the prediction targets (entity or relation) that do not already occur in the training data, and then use metrics such as MRR or Hits@K. The final MTR metric is given by the micro-average over all nine queries.

Obtaining rankings for some of these tasks is not as straightforward as obtaining rankings for LP. First, for a REL query of the form $(s, ?, o)$, we proceed as in Chang et al. (2020) and rank all $r' \in \mathcal{R}$ such that $(s, r', o) \notin \mathcal{G}_{\text{train}}$ in descending order of their scores $s(s, r', o)$, where s is the score function of the KGE model we are evaluating. For the other tasks, which involve wildcards, it is not immediately clear how to perform prediction using a KGE model. We first discuss scoring and ranking, then filtering of training data. Consider for example the NBR query $(s, ?, *)$, where our goal is to rank relations. The perhaps simplest approach to obtain a relation ranking

Algorithm 5.1: Extended Score Function (accepts wildcards)

Require: t: triples to compute score, e.g. $(i, k, *)$
s: model score function
$\mathcal{C}$: set of candidates for wildcard slot
Ensure: Score of given triple t

```
max_score ← 0
if wildcard not slot in t then
    max_score ← s(t)
else
    foreach c ∈ C do
        candidate_t = (i, k, c)                    // e.g. t = (i, k, *)
        candidate_score = s(candidate_t)
        if candidate_score ≥ max_score then
            max_score ← candidate_score
return max_score
```

is to first rank all triples of the form (s, r', o'), for every $r' \in \mathcal{R}$ and $o' \in \mathcal{E}$, and then rank relations by their first appearance (e.g., the relation of the highest-scoring triple is ranked at the top).

The process described above can be seen as an *extended score function* that accepts wildcards, which we describe in Algorithm 5.1. That approach corresponds to using $s(s, r', *) = \max_{o' \in \mathcal{E}} s(s, r', o')$, i.e, the score of a relation r' is the score of its most plausible triple. Although other aggregation functions are feasible, we only consider max-aggregation because it does not make any additional assumptions on the scoring function.

To filter training data during model evaluation, we remove all relations r' such that $(s, r', o') \in \mathcal{G}_{\text{train}}$ for some $o' \in \mathcal{E}$; i.e., we remove all prediction targets that are already implied by the training data. We proceed similarly for all other tasks involving wildcards. Note that the number of score computations needed to predict entity targets for queries without wildcards is $O(|\mathcal{E}|)$, whereas the one for queries with wildcards is $O(|\mathcal{E}||\mathcal{R}|)$. We discuss in Section 5.3 how the latter cost can be reduced to $O(|\mathcal{E}|)$.

5.3 Multi-Task Training

We now generalize the standard KGE training approach to include all of the graph-structure prediction tasks, called multi-task training (MTT). Our goal is to be able to improve model performance on these tasks, while at the same time keeping training and prediction cost low. To this end, we constructed a task-specific cost function for each individual training task; the final cost function is then given as a weighted linear combination of the task-specific costs (and additional regularization terms), where the weights are hyperparameters.

To illustrate how training objectives are constructed using more than one training task, i.e. query, we define both the standard training objective (STD) based on link prediction (introduced in Section 2.2.7) and our proposed multi-task objective (MTT) as follows. Let $T_o = \{(t, l)\}$ be the set of relevant positive and negative examples t and corresponding label l induced by the link prediction query $(s, p, ?)$ in a given training set. Let T_s be the analogous set of examples for query $(?, p, o)$. For some loss function L, the STD training approach optimizes the following objective function (we omit the penalty term for brevity):

$$f(\theta) = \underset{\theta}{\operatorname{argmin}} \left(\frac{1}{|T_s|} \sum_{(t,l) \in T_s} L(s(t), l) + \frac{1}{|T_o|} \sum_{(t,l) \in T_o} L(s(t), l) \right) \tag{5.1}$$

where s is a KGE score function parameterized by model parameters θ. We generalize this objective to define the following multi-task training (MTT) objective:

$$f(\theta) = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{T_i \in \mathcal{T}} \sum_{(t,l) \in T_i} \lambda_i L(s(t), l) \tag{5.2}$$

where $\mathcal{T} = \{T_1, T_2, \ldots\}$ is a superset of training examples for queries T_i, N is the sum of the cardinalities of each T_i and λ_i a hyperparameter that controls the impact of query i in the training objective. Chen et al. (2021c) have already followed this training approach by adding the relation prediction task, i.e. $(i, ?, j)$ to Eq. 5.1. They set $\lambda_s = \lambda_o = 1$ and tune λ_r. We note that Equations 5.1 and 5.2 do not describe the exact training objective with some loss functions, e.g. some losses require a positive and corresponding set of negatives to compute a loss value. However, the MTT objective can be

Algorithm 5.2: Multi-task Training (MTT)

Require: $\mathcal{T}$: set of training triples,
$\mathcal{E}$: set of entities in knowledge graph $\mathcal{K}$
θ: model parameters,
$\mathcal{W}$: set of training queries,
$\mathcal{Q}$: set of weights for given training queries

Ensure: Updated model parameters θ

foreach $q, w \in \mathcal{Q}, \mathcal{W}$ **do**
  $N \leftarrow$ construct set of negatives for q using $\mathcal{T}$
  $\mathcal{T}_{\text{all}} \leftarrow \mathcal{T} \cup N$
  $s_{\text{all}} \leftarrow \text{COMPUTE\_SCORES}(\mathcal{T}_{\text{all}})$
  $l_q \leftarrow w * \text{COMPUTE\_LOSS}(s_{\text{all}}, \mathcal{T}_{\text{all}})$ `// loss weighted by` w
  $\theta \leftarrow \text{UPDATE\_PARAMETERS}(\theta, l_q)$

reformulated for every loss function commonly used to train KGE models. We provide such a general description of the MTT approach in Algorithm 5.2. As with loss functions, the MTT approach is agnostic to the choice of model and training task. However, in this thesis we focus on the set of tasks summarized in Table 5.1.

The examples for the task-specific cost functions for link prediction and relation prediction are obtained as in standard training (see Sec. 2.2.7): for each positive triple $(s, p, o) \in \mathcal{G}$, we construct a set of negatives according to the corresponding query (i.e., by perturbing the position of the prediction target) and then apply the loss function (e.g., cross entropy).

For the other tasks, which involve wildcards, we proceed differently. Instead of performing some form of (costly) score aggregation during training as in the extended score function described in Algorithm 5.1, we "convert" tasks with wildcards into tasks without wildcards. To do so, we make use of three virtual *wildcard entities*—one for subjects (any_S), one for relations (any_R), and one for objects (any_O)—and learn embeddings for these entities. During training, we conceptually replace wildcards by their corresponding wildcard entity and proceed as before. For example, for training triple (s, p, o) and NBR query $(s, ?, *)$, we consider the virtual triple (s, p, any_O) along with query $(s, ?, any_O)$. By doing so, we converted the NBR task into a REL task. We also use the so-obtained wildcard embeddings during prediction time in the same fashion; e.g., we set $s(s, r', *) = s(s, r', any_O)$. Instead of performing

score aggregation, the model thus directly learns extended scores.

The advantage of the MTT approach is that (i) the prediction costs remain stable, i.e., the cost of graph-structure prediction or downstream task prediction is unaffected by the number or choice of pre-training tasks, and (ii) the pre-training costs increase only linearly in the number of tasks (see Table 5.6). It is important that MTT be scalable, as learning representations of an entire KG can be very costly, which is also a motivation for learning generally useful representations of the KG.

Note that the wildcard embeddings are not used for entity-level downstream tasks. Nevertheless, using wildcard entities during training affects all other entities as well. This is because the embedding of each entity occurs in all graph-structure prediction tasks. The entity embeddings of a good KGE model thus needs to be suitable for all these tasks, not just for link prediction.

5.4 Experimental Study

We conducted a large experimental study to test our proposed MTT approach for model training and our proposed MTR approach for model evaluation. Specifically, our goals were (i) to evaluate whether standard KGE models capture various properties of a KG by assessing their performance on new graph-structure prediction tasks, including link prediction, and (ii) to determine whether (and by how much) KGEs improve their performance on these tasks when simultaneously trained for them. To our knowledge, almost no prior work has focused on studying different training objectives for KGE models. We discuss related work further in Section 5.5.

5.4.1 Experimental Settings

Datasets. We chose three commonly used benchmark datasets for evaluating KGE models: FB15K-237 (Toutanova and Chen, 2015) (referred to as FB-237 for brevity), WNRR (Dettmers et al., 2018), YAGO3-10 (Mahdisoltani et al., 2014), and WIKIDATA5M (Wang et al., 2021) (referred to as WIKI5M for brevity). Each dataset is associated with a training, a validation and a test split. As described in Section 2.4, FB-237 and WNRR are designed to be harder benchmarks for link prediction. YAGO3-10 is a considerably larger dataset than the two, but WIKI5M is much larger than all other datasets. Due

Dataset	Entities	Relations	Training	Validation	Test
FB-237	14 505	237	272 115	17 535	20 466
YAGO3-10	123 182	37	1 079 040	5 000	5 000
WNRR	40 559	11	86 835	3 034	3 134
WIKI5M	4 818 679	828	21 343 681	5 357	5 321

Table 5.2: Statistics of datasets used in this study.

to the high cost of multi-task ranking (MTR) on models without wildcard embeddings, we only report results on WIKI5M of MTT models. The statistics of these datasets are summarized in Table 5.2.

KGE models. We chose four popular, representative KGE models: TransE (Bordes et al., 2013b) and DistMult (Yang et al., 2015) (basic translational and factorization models, respectively) as well as RotatE (Sun et al., 2019) and ComplEx (Trouillon et al., 2016) (translational and factorization models). Due to cost, we only train ComplEx and TransE on WIKI5M.

KGE training. We used LibKGE (Broscheit et al., 2020) for *STD training* (LP only) as a baseline and extended it to support MTT/MTR for model training/evaluation. All KGE models were trained for a maximum of 200 epochs with early stopping on validation MRR checked every 10 epochs. We used cross-entropy as loss function, as it systematically outperformed other losses in most prior studies. We used *1vsAll* training with FB-237 and WNRR (to achieve good results) and *NegSamp* with YAGO3-10 and WIKI5M to scale to these larger datasets. Models were selected w.r.t. performance (MRR) on the validation data. We selected STD models with LP task and MTT models with the MTR task. For MTT training, we used all tasks in Table 5.1. We discuss the impact of using fewer tasks during training in Section 5.4.4.

KGE evaluation. As with training, we evaluate KGE models with respect to each of the five graph-structure prediction tasks of Sec. 5.1 (LP, REL, DOM, NBE, NBR) using filtered MRR on test data. We also aggregate these metrics into the multi-task ranking MRR (MTR).

KGE hyperparameters. We closely follow the approach of the experimental study in Section 3.2 to perform hyperparameter optimization. We performed 30 random trials using SOBOL sampling (Bergstra and Bengio, 2012) over a large search space to tune several hyperparameters, e.g. regular-

Hyperparameter	Values
Embedding size†	$\{128, 256, 512\}$
Training type	{NegSamp (YAGO3-10), 1vsAll (FB15K, WNRR)}
Task Weights (MTT)	$[0.1, 10]$, log scale
No. subject samples (NegSamp)	$[1, 10000]$, log scale
No. object samples (NegSamp)	$[1, 10000]$, log scale
Optimizer	{Adam, Adagrad}
Batch size*	$\{128, 256, 512, 1024$ (except on YAGO3-10)$\}$
Learning rate	$[10^{-4}, 1]$, log scale
LR scheduler patience	$[0, 10]$
L_p regularization	{L1, L2, L3, None}
Entity emb. weight	$[10^{-20}, 10^{-5}]$
Relation emb. weight	$[10^{-20}, 10^{-5}]$
Frequency weighting	{True, False}
Embedding normalization (TransE)	
Entity	{True, False}
Relation	{True, False}
Dropout	
Entity embedding	$[0.0, 0.5]$
Relation embedding	$[0.0, 0.5]$
Embedding initialization	{Normal, Unif, XvNorm, XvUnif}
Std. deviation (Normal)	$[10^{-5}, 1.0]$
Interval (Unif)	$[-1.0, 1.0]$
Gain (XvNorm)	1.0
Gain (XvUnif)	1.0

† For RotatE, embedding size is fixed 128 on WNRR and set to either 128 or 256 for YAGO3-10. For Transe, this is set to either 128 or 256 for FB-237 and fixed to 128 for WNRR and 1024 for YAGO3-10.

* For RotatE, batch size is fixed to 256 in YAGO3-10 and to 128 on FB-237 and WNRR. For Transe, this is set to either 128 or 256 on YAGO3-10.

Table 5.3: Hyperparameter search space for pre-training KGE models. Restrictions for RotatE and TransE are due to higher memory consumption and runtime.

	Training	*Selection*	*Graph-structure prediction - MRR (↑)*					
			LP	REL	DOM	NBE	NBR	MTR
FB-237	STD	LP	**.346**	.919	.624	.051	.136	.342
	MTT	MTR	.331	**.977**	**.813**	**.210**	**.925**	**.606**
YAGO3-10	STD	LP	**.550**	.900	.178	.400	.656	.432
	MTT	MTR	.538	**.954**	**.861**	**.591**	**.967**	**.759**
WNRR	STD	LP	**.474**	.794	.396	**.432**	.881	.553
	MTT	MTR	.459	**.874**	**.593**	.426	**.955**	**.633**
WIKI5M	STD	LP*	**.288**	–	–	–	–	–
	MTT	MTR	.250	.908	.185	.169	.503	.347

* Not evaluated on new graph-structure prediction tasks due to high cost.

Table 5.4: Best performance with STD and MTT training on test data of graph-structure prediction tasks. Bold entries show best performance per task and dataset. MTT outperforms STD training all tasks almost every time, suggesting that standard KGE models do not capture general information about the graph unless trained for this purpose.

ization, embedding size, batch size, dropout, initialization and training task weights, each in $[0.1, 10.0]$, log scale. To keep our study feasible, we reduced the maximum batch and embedding size for larger datasets and expensive models. The full search space is described in Table 5.3.

5.4.2 Model Performance

Overall performance. In Table 5.4, we report test MRR of the best performance achieved by any of the KGE models with each training approach, i.e. standard training (STD) with the LP task for model selection and our proposed multi-task training (MTT) and model selection (MTR). This includes the standard KGE evaluation based on link prediction (LP); our proposed tasks: relation domain (DOM), neighboring entities (NBE), neighboring relations (NBR); and our proposed MTR evaluation protocol that aggregates all of the mentioned tasks. Bold entries show the best performance for each task and dataset.

The results show that across all datasets and KGE models, STD training performed considerably worse on all graph-structure tasks compared to MTT

training, except on LP. This suggests that, unless trained for it, standard KGE models fail to capture graph structure more generally than what is required for link prediction. The performance for these tasks improved significantly when they were introduced as auxiliary training objectives with MTT training in almost all cases, especially in DOM, NBE and NBR, and without a significant increase in training time (linear in the number of training tasks, see Table 5.6). MTT models had slightly lower performance on LP, but the decrease was often small and outweighed by significantly improved performance over the other tasks. A notable exception was NBE, which is the only task that uses wildcard embeddings for relations. Here STD outperformed MTT on YAGO-10 on WNRR. **In general, MTT improved significantly on STD for graph structure prediction and can thus be used to improve KGE's ability to learn multiple graph tasks simultaneously.** Note that new training tasks can be designed and used with MTT to capture specific properties of the graph that may be desired.

Model performance. Table 5.5 shows the same comparison between STD and MTT training as in Table 5.4, but at the model level. For easier comparison between STD and MTT, underlined entries highlight the best performance compared to the entry with the same corresponding KGE model on the same dataset, but that uses the other training method. The results show that, indeed, every model is able to capture more information about the KG when trained for it on multiple tasks simultaneously. For a given model, the improvement can be large, often by a factor of 2x and up to 10x depending on model, task and dataset. Moreover, it is often the case with STD training that the best models for LP are far outperformed on other tasks by different STD models. For example, the best LP performance for STD models on FB-237 is ComplEx, but RotatE STD outperforms it considerably on REL and TransE STD on DOM. Similar observations can be made about the best models on MTR, but the compromise in performance w.r.t. the best models for each task is significantly smaller. **These results suggest that MTR is a more suitable task when the goal is training models that capture more information about the KG during training.**

Performance per task. When comparing performance across individual tasks in Table 5.5, we see that **the size of the performance difference between the two training approaches depends on the task, with STD and MTT often performing similarly on some tasks like LP or REL, but significantly differently on others like DOM or NBE.** In addition, some tasks seem to

		Training	*Selection*	*Graph-structure prediction - MRR (↑)*					
				LP	REL	DOM	NBE	NBR	MTR
FB-237	ComplEx	STD	LP	**.346**	.805	.423	.016	.046	.274
		MTT	MTR	.331	**977**	.773	**.210**	**.925**	**.606**
	DistMult	STD	LP	.342	.388	.045	.009	.036	.139
		MTT	MTR	.327	.939	.780	.142	.879	.577
	RotatE	STD	LP	.312	.919	.581	.051	.136	.342
		MTT	MTR	.314	.964	**.813**	.160	.922	.598
	TransE	STD	LP	.330	.900	.624	.038	.054	.332
		MTT	MTR	.288	.960	.708	.112	.911	.555
YAGO3-10	ComplEx	STD	LP	**.550**	.900	.120	.215	.517	.411
		MTT	MTR	.538	.930	.836	**.591**	.940	.749
	DistMult	STD	LP	.539	.881	.010	.327	.613	.429
		MTT	MTR	.536	.941	**.861**	.581	**.967**	**.759**
	RotatE	STD	LP	.436	.809	.046	.400	.656	.432
		MTT	MTR	.427	.933	.032	.550	.694	.482
	TransE	STD	LP	.504	.860	.178	.287	.175	.349
		MTT	MTR	.048	**.954**	.686	.046	.798	.457
WNRR	ComplEx	STD	LP	**.474**	.782	.396	.246	.690	.488
		MTT	MTR	.459	.831	**.593**	.426	.953	**.633**
	DistMult	STD	LP	.447	.767	.081	.253	.702	.415
		MTT	MTR	.431	.804	.573	.342	.952	.600
	RotatE	STD	LP	.469	.794	.311	**.432**	.881	.553
		MTT	MTR	.431	**.874**	.512	.239	**.955**	.572
	TransE	STD	LP	.174	.707	.044	.171	.332	.239
		MTT	MTR	.094	.603	.476	.095	.827	.399
WIKI5M	ComplEx	STD*	LP	**.288**	–	–	–	–	–
		MTT	MTR	.215	.804	.087	.136	.342	.263
	TransE	STD*	LP	**.288**	–	–	–	–	–
		MTT	MTR	.250	**.908**	**.185**	**.169**	**.503**	**.347**

* Not evaluated on new graph-structure prediction tasks due to high cost.

Table 5.5: Performance on test data of graph-structure prediction tasks with STD and MTT training. Bold entries show best performance per task and dataset. Underlined entries show best performance between STD and MTT. MTR is more suitable than LP for representing models that capture a graph more generally.

		Average training epoch time in seconds			
		FB-237	YAGO3-10	WNRR	WIKI5M
ComplEx	STD	4.92	97.88	2.32	823.80
	MTT	10.83	137.13	8.13	1635.90
TransE	STD	78.76	141.62	98.45	1115.65
	MTT	245.05	219.42	278.60	2124.29

Table 5.6: Average training epoch time in seconds over first 5 epochs of best models with STD and MTT training. All tests were done with an 11th gen. Intel Core i7-11700K, 64GB of RAM and an NVIDIA GeForce RTX 3090.

be more difficult than others, e.g. NBE results are often low even with MTT training. This may be because it is difficult to learn some subsets of tasks simultaneously. We discuss this further in Section 5.4.4.

5.4.3 Discussion

From a training objective perspective, these results are not surprising, as STD training only focuses on the LP task (we discuss similar effects on MTT in Section 5.4.4). However, **these results do contradict what is usually argued about the ability of these models to generally capture KG properties** (Bordes et al., 2011, 2013b; Nickel et al., 2015; Wang et al., 2017), which in turn inspired their use as KG representations in downstream applications (El-Kishky et al., 2022; Ilyas et al., 2022; He et al., 2020; Zhang et al., 2019b; Wang et al., 2018a; Baier et al., 2017). This is despite the fact that KGE training has almost exclusively focused on the link prediction task and almost no prior work studies different training approaches (see Section 5.5). In addition, some of the new tasks are similar enough to link prediction, and arguably simpler, that the results are indeed unexpected. For example, a model that is good at link prediction may be able to answer *(Austin, capital of, ?)* and *(?, capital of, Texas)*, yet it may not be able to predict that *capital of* is a relation connected to Austin and/or Texas (NBR). Similar arguments can be made for other tasks. We discuss in Section 5.4.4 how excluding the LP task during pre-training can result in improved performance on other graph-structure tasks.

Generally, when the goal is the link prediction task, STD training is more suitable. But we show empirically in Chapter 6 that the choice of training

objective has an impact on the learned representations and that including the LP task during pre-training can at times be detrimental for the performance of downstream models.

5.4.4 Impact of Training Task Selection

Table 5.7 summarizes our results from exploring the impact of pre-training task selection and, in particular, whether all proposed MTT tasks are beneficial. To keep computational costs feasible, we focused on FB-237. We explored performance using MTT, and MTT without either the LP, REL, DOM, NBE, or NBR pre-training task, as well as without LP+REL or without DOM+NBR. We report only on the sets of tasks that provided relevant results for our discussion. All combinations of pre-training tasks are reported in Table C.1 in Appendix C.

We found that for graph-structure predictions, excluding a task generally led to lower performance on that task, as expected. It may also, however, lead to a boost in performance on other tasks, For example, the best REL and NBE performance for TransE is obtained when LP is excluded during training. **These results suggest that improving performance on some pre-training tasks may be detrimental to the performance of other pre-training tasks, so that more research is required to understand the relation between different training tasks and their impact on downstream applications.**

5.5 Related Work

As with our work with the KBC task proposed in Chapter 4, our goal with the MTR evaluation protocol is related to studies about probing KGE models to better understand their performance (Widjaja et al., 2022; Allen et al., 2021; Rim et al., 2021; Meilicke et al., 2018). The main difference is that while these studies focus on a model's link prediction performance over different types of relations, we assess model performance on a new a new set of graph-structure prediction tasks designed to measure a model's ability to make structures in the graph beyond link prediction.

W.r.t. MTT, and as mentioned before, most of the work on KGE models is based on the same training approach. However, the are some exceptions in the literature. One exception is the work of Nickel et al. (2011) when

	Training	*Selection*	*Graph-structure prediction (↑)*					
			LP	REL	DOM	NBE	NBR	MTR
ComplEx	STD	LP	**.346**	.805	.423	.016	.046	.274
	MTT	MTR	.331	**.977**	.773	**.210**	.925	**.606**
	w/o LP	MTR	.154	.972	.831	.200	.932	.579
	w/o NBE	MTR	.315	.958	**.850**	.005	.936	.575
	w/o LP+REL	MTR	.001	.009	.843	.177	**.939**	.436
DistMult	STD	LP	.342	.388	.045	.009	.036	.139
	MTT	MTR	.327	.939	.780	.142	.879	.577
	w/o LP	MTR	.159	.954	.826	.087	.937	.553
	w/o DOM	MTR	.323	.948	.703	.106	.914	.560
	w/o NBE	MTR	.316	.928	.848	.003	.937	.571
RotatE	STD	LP	.312	.919	.581	.051	.136	.342
	MTT	MTR	.314	.964	.813	.160	.922	.598
	w/o LP	MTR	.204	.914	.842	.126	.928	.568
	w/o DOM	MTR	.319	.965	.661	.170	.883	.559
	w/o NBR	MTR	.318	.964	.710	.168	.673	.522
TransE	STD	LP	.330	.900	.624	.038	.054	.332
	MTT	MTR	.288	.960	.708	.112	.911	.555
	w/o LP	MTR	.271	.968	.781	.138	.901	.572
	w/o NBE	MTR	.330	.966	.801	.012	.904	.562
	w/o NBR	MTR	.329	.966	.723	.125	.790	.545

Table 5.7: Performance on test data of graph-structure prediction and downstream tasks for FB-237 of STD with LP model selection and various forms of multi-task training, all using MTR for model selection. Objectives such as w/o LP are MTT objectives with all tasks in Table 5.1 except one, in this case, LP. Results show that excluding the LP task during pre-training often results in improved downstream performance, and that using all pre-training tasks is often not the best choice.

proposing RESCAL, one of the earliest KGE models. They trained their model the *reconstruction* task. This task aims to construct the entire training data using cost functions such as $\sum_{s,p,o} \| I[(s,p,o) \in \mathcal{G}_{\text{train}}] - s(s,p,o) \|_2^2$, where $I[\cdot]$ is a $0/1$ indicator and $\mathcal{G}_{\text{train}}$ is the training set of some knowledge graph $\mathcal{G}$. In contrast to the local-closed world assumption made by commonly used training methods, this task makes the arguably stronger closed world assumption. A similar approach was explored by Li et al. (2021). We do not consider such methods further because training costs are excessive (at least unless squared error is used) and the empirical performance reported by Li et al. (2021) is generally far behind KGE models trained with link prediction.

Another exception is Chen et al. (2021c), who proposed to augment the link prediction task with *relation prediction* during training (but not evaluation). In this study, we expanded upon this work by considering additional pre-training tasks and by focusing on graph-structure prediction performance including the LP task.

5.6 Summary

In this chapter, we looked beyond the link prediction task and asked whether KGE models are indeed able to capture structures in the graph more generally, as is often argued in the literature. We proposed a new set of tasks similar to link prediction to assess whether KGE models capture these different graph properties, and generalized the standard entity ranking protocol to include these (and any number of) tasks for model evaluation. We also generalized the standard training approach based on link prediction to a proposed multi-task training method that allows KGE models to be trained on any number of tasks simultaneously, including link prediction. To efficiently train models for some of these new types of tasks, we introduced the concept of wildcard embeddings, which are a few additional parameters that effectively and reliably extend the ability of KGE models to answer new types of queries about graph structure.

To test our proposed methods, we conducted a large experimental study where we included our proposed set of graph-structure prediction tasks as additional training objectives. The results showed that with additional training objectives, popular KGE models are able to capture more information about the graph and thus significantly improve performance on graph-structure

prediction tasks compared to models that only train with the standard link prediction approach. This suggests that standard link prediction models do not capture as much information in a KG as possible, and that new forms of training KGE models may be more suitable for obtaining more general representations of KGs.

In the next chapter, we explore the impact that capturing more information about the graph has on the resulting learned representations when they are used in downstream applications.

CHAPTER

SIX

DOWNSTREAM APPLICATIONS

In this chapter, we extend the questions we asked in Chapter 5 to also include downstream applications. Despite being used as KG representations in downstream applications, e.g. recommender systems (El-Kishky et al., 2022; Wang et al., 2018a) and language models (He et al., 2020; Zhang et al., 2019b), almost all previous research on KGEs models focuses on link prediction performance as a way to select good KGE models. Thus, it is not well understood how choices taken in model training and model selection affect the usability of these KG representations in downstream applications. This stands in contrast to the use of learned representations in other fields such as natural language processing, where pre-training objectives are often designed with downstream tasks in mind, and the quality of the learned representations is assessed in downstream applications (Devlin et al., 2019; Mikolov et al., 2013b).

Our goal is to explore the impact that different training approaches have on the learned representations provided by KGE models in terms of how useful they are in downstream applications. Specifically, we investigate whether KGE models are suitable pre-trained representations for node-level downstream tasks such as entity classification (e.g., the profession of a person) or regression (e.g., the average rating of a movie). To this end, we conducted a large empirical study using 35 downstream tasks on three different KGs, where we compared the performance of downstream models that use representations from KGEs trained with the standard link prediction approach,

and with our multi-task training (MTT) approach introduced in Chapter 5.

We found that KGE models trained with the standard approach often perform decent on these tasks and, in fact, the best KGE models often (but not always) exceed the performance of recent graph neural networks that train directly on the downstream task, such as KE-GCN (Yu et al., 2021a). However, the KGE models with best downstream task performance were often not the best-performing models for link prediction. For example, we found that the basic TransE model (Bordes et al., 2013b) can be superior to more recent KGE models better suited for link prediction such as ComplEx (Trouillon et al., 2016) or RotatE (Sun et al., 2019). This suggests that good link prediction performance is not necessarily indicative of good downstream task performance. This is another indication that, as seen in previous chapters, the focus on link prediction is too narrow for pre-training KGE models, i.e., to provide generally useful features for downstream applications.

We further we found that multi-task training often (but not always) improved downstream performance. In fact, we found that excluding the link prediction task during pre-training resulted in better downstream performance more often than not. However, our results also show that capturing more information about the graph does not directly translate to better downstream performance, as the more useful models were often those that were pre-trained without using all available training tasks. In general, the best choice of pre-training tasks depends on the dataset, KGE model class, and type of downstream task, suggesting that more research is needed to better understand the relation between capturing properties of the graph and providing useful features for downstream applications.

This chapter is organized as follows. In Section 6.1 we briefly review how KGE models are used in some downstream applications. We discuss our experimental study in Section 6.2. We discuss related work in Section 6.3 and summarize our contributions in Section 6.4.

6.1 Pre-Trained Knowledge Graph Representations

As illustrated in Figure 6.1, many applications use KGE models as representations of knowledge graphs with the purpose of injecting structured knowledge into their downstream pipelines. In this sense, KGEs are seen as pre-trained models that encode the information in the KG. Generally, the

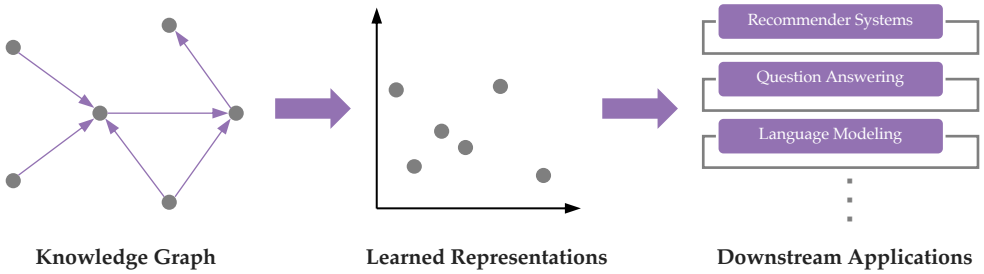


Figure 6.1: The KGE pipeline. The learned representations of knowledge graphs are used to inject structured data into downstream applications.

learned entity and/or relation representations are used as feature vectors in different ways, the specifics of which depend on the type of downstream application. We describe some examples in the following.

El-Kishky et al. (2022) use KGE models as part of a wide variety of recommender systems. The authors encode different types of interactions between users and other components of their system into a knowledge graph. They then embed this graph using TransE and the standard link prediction training approach. They find that the additional information provided by these pre-trained representations results in improvements across a range of recommender tasks.

Ilyas et al. (2022) constructed a large knowledge graph with the purpose of serving factual knowledge to several industry applications, such as question answering systems. To improve this service, they learn representations of this KG with TransE and DistMult using the standard training approach, and apply vector-based similarity search to the learned representations for fact ranking, fact verification and missing fact imputation.

Wang et al. (2021) proposed to jointly learn a language model and a KGE model with the purpose of producing both a more factually accurate language model and a text-enhanced KGE model that can make inductive predictions (see Section 2.2.8) by creating representations of unseen entities based on their textual descriptions. To this end, the authors combined the training objectives of a masked language model and a KGE model, and conditioned the representations of the KG entities to be generated by the same text encoder used in the language model based on given textual descriptions of each entity.

Other examples of using KGE models as pre-trained representations

include other approaches to training language models that incorporate the knowledge encoded in the KG embeddings (He et al., 2020; Zhang et al., 2019b), and visual models that use semantic information encoded in KGEs to improve performance on the task of mapping images to their descriptions in natural language (Baier et al., 2017).

Training for downstream applications. In general, there are different approaches to training KGE models with the goal of using the resulting representations in downstream applications. A more general approach is to pre-train models on self-supervised tasks that encourage a model to encode the information in a KG, so that it may later be used in downstream applications. Such an approach is general in that no information about any downstream task is used during pre-training. This may be beneficial because (i) it may be expensive to pre-train the models, e.g. when learning on an entire KG like Freebase (Zheng et al., 2020), and (ii) it may not be desirable to include information about specific downstream tasks during training, as this may impact the general usability of the representations on other downstream applications that may not be known at the time of training.

An alternative to training only on self-supervised tasks, such as those studied in Chapter 5, is to incorporate information about relevant downstream applications during training. This may be done by including them as training objectives, or by performing model selection based on performance on these tasks. Such an approach may be desirable when the application scenario is clear, but as mentioned before, it is possible that such an approach results in representations that favor some applications over others.

In the experimental study discussed in the next section, we take the more general approach of training without downstream task information. While we do test the impact of using performance on downstream applications for model selection, we leave the general question on the impact of including downstream applications during training for future work.

6.2 Experimental Study

We conducted a large experimental study with the goals of (i) assessing performance of standard KGE models on downstream tasks, and (ii) assessing the impact of different training approaches on downstream task performance. We used the pre-trained models from our experimental study in Chapter 5

Benchmark	Name	Train	Validation	Test
FB-237	Entity Type	6 719	–	1 680
	Profession	2 537	–	635
	Organization Type	342	–	86
	Writer Type	136	–	34
YAGO3-10	Entity Type	69 592	–	17 398
	Player Type	33 928	–	8 483
	Profession	14 480	–	3 621
	Writer Type	4 870	–	1 218
	Scientist Type	2 041	–	511
	Organization Type	1 248	–	312
	Artists Type	520	–	130
	Waterbody Type	195	–	49

Table 6.1: Statistics of datasets for entity classification downstream tasks used to evaluate pre-trained KGEs.

as input features for several downstream models on a large number of downstream task datasets. In order to observe the impact of pre-training choices more generally, we aimed at providing results across a considerable number of downstream tasks. Thus, our study focuses on simple classification and regression tasks that use the KGE embeddings as input features. We do not consider in this study applications such as recommender systems, as these pipelines are generally more involved and require more effort to implement and more expertise to analyze. This would make our study less scalable, so we leave that for future work.

6.2.1 Experimental Settings

Downstream tasks. We collected or created data for 35 downstream tasks on FB15K-237 (referred to as FB-237 for brevity), YAGO3-10 or WIKI5M. This includes the datasets of Jain et al. (2021) for entity classification on FB-237 and YAGO3-10, which aim to predict the types of entities at different granularities. For regression, we use the datasets of Pezeshkpour et al. (2018) for YAGO3-10, which consist of temporal prediction tasks (e.g., the year an event took place), and the dataset of Huang et al. (2021) for node importance prediction. We

Benchmark	Name	Train	Validation	Test
FB-237	Node Importance	9 877	1 380	2 823
	Birth Year	3 538	442	444
	Latitude	2 568	321	322
	Longitude	2 560	320	322
	Person Height	2 295	287	288
	Size Area	1 731	216	218
	Population	1 543	193	193
	Film Release Year	1 493	186	188
	Org Year Founded	985	123	124
	Film Rating	591	73	75
YAGO3-10	Born on Year	60 409	–	6 730
	Created on Year	23 896	–	2 638
	Died on Year	13 582	–	1 513
	Destroyed on Year	1 630	–	186
	Happened on Year	749	–	73
WIKI5M	Date of Birth	992 126	124 015	124 017
	Album Publication	29 156	3 644	3 645
	Asteroid Magnitude	16 722	2 090	2 091
	River Length	10 092	1 261	1 262
	Airport Elevation	9 054	1 131	1 133
	Sports Season Start	7 631	953	955
	Village Population	3 691	461	462
	Municipality Area	3 158	394	396

Table 6.2: Statistics of datasets for regression downstream tasks.

also created several regression tasks for FB-237 from the multi-modal data of García-Durán et al. (2018) by predicting literals associated to entities (e.g., a date, a person's height, the rating of a movie). To create regression tasks for WIKI5M, we followed the same approach using numerical relations extracted from Wikidata (Van Veen, 2019). Datasets statistics are given in Tables 6.1 and 6.2.

KGE models. Since we are interested in pre-trained KGE models, we used the KGE models trained for the experiments discussed in Section 5.4. Thus, no information from downstream tasks was used for KGE model training and selection; i.e. each of the pre-trained KGE models are used as input

for each of the downstream tasks in our experiment. For model selection, we selected STD models with LP task (represented by the standard entity ranking evaluation protocol), but combined MTT models with the LP task or the MTR task. Further improvements may be made by using downstream tasks during training (Aribandi et al., 2022) at the cost, perhaps, of obtaining less general representations, but we leave such exploration to future work.

We note that these are not the latest KGE models, but they can reach state-of-the-art performance with reasonable embedding sizes (Ruffinelli et al., 2020) and allow us to conduct a large-scale comparative study. Specifically, RotatE and ComplEx are the methods of choice for low-cost embeddings with good prediction performance (Ruffinelli et al., 2020; Sun et al., 2019), and—with an increase in model size and/or training cost (Lacroix et al., 2018; Chen et al., 2021c)—can perform as well as more involved state-of-the-art models such as the transformer-based HittER model (Chen et al., 2021a). Some recent models achieve better performance on link prediction, but these are often models that focus exclusively on the link prediction task and do not directly provide entity representations that can be used on downstream applications. For example, the main idea with HittER is to add entity context for link prediction and it is not immediately clear how to use such a model for graph-structure prediction or downstream tasks. Similarly, the NBFNet model (Zhu et al., 2021) exclusively focuses on link prediction and does not provide entity embeddings for downstream models. Such models have considerably higher training costs, which limits their ability to scale entire large industry-scale KGs. It is because of scalability that simpler models are often preferred in the industry (El-Kishky et al., 2022).

Downstream models. We use implementations from scikit-learn (Pedregosa et al., 2011). Each model uses only the node embeddings of the pre-trained KG model as input features. For classification, we use multi-layer perceptrons (MLP), logistic regression, KNN, and random forests. For regression, we use MLP and linear regression.

Downstream training. Each model was trained using 5-fold cross validation and selected based on mean validation performance across folds (we discuss specific metrics below). We then retrained the selected model on the union of the training and validation split (if present). To tune hyperparameters, we use 10 trials of random search with SOBOL sampling for each downstream model. The search space for each downstream model is given in Table 6.3. We treat the choice of downstream model as a hyperparameter.

Model	Hyperparameter	Values
MLP	Hidden Layer	{(100,), (10,), (100, 100), (10, 10)}
	Alpha	$[0.00001, 0.001]$
	Learning Rate	$[0.001, 0.01]$
	Solver	[Adam, LBFGS]
Logistic Regression	C	$[100, 100000]$
KNN	n_neighbors	$[1, 10]$
Random Forest	num_estimators	$[10, 50, 100, 200]$
Linear Regression	Alpha	$[0.00001, 0.001]$
KE-GCN	Dimension	$\{16, 32, 64\}$
	Additional Layers	{0, 1, 2}
	Learning Rate	{0.001, 0.005, 0.01, 0.05, 0.1}
	Alpha	{0.3, 0.5}

Table 6.3: Hyperparameter search space for training downstream models. All hyperparameters except those of KE-GCN follow the semantics by scikit-learn.

Downstream evaluation. For entity classification, we report *weighted F1*, as in Jain et al. (2021), aggregated across all classification tasks (denoted EC). For regression, we chose relative squared error (RSE), defined as follows:

$$\text{RSE} = \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \tag{6.1}$$

where N is the number of evaluation examples, y_i are targets to predict, $\hat{y}_i$ are model predictions, and $\bar{y} = \frac{1}{N}\sum_n y_i$, i.e. the mean of targets to predict. We chose RSE because it is interpretable and allows meaningful averaging across the different regression tasks (denoted REG). An RSE value of 1 is equivalent to the performance of a model that predicts the average of the dependent variable in the evaluation data; lower values are better. For each metric, we report the mean and standard deviation over 3 training runs of the downstream model.

Dataset	*Training*	*Selection*	*Downstream tasks* EC (↑)	REG (↓)
FB-237	STD	LP	.873	.447
	MTT	LP	<u>.890</u>	<u>.394</u>
YAGO3-10	STD	LP	.742	.447
	MTT	MTR	<u>.746</u>	<u>.441</u>
WIKI5M	STD*	LP	–	<u>.596</u>
	MTT	MTR	–	.650

Table 6.4: Best performance on test data of downstream tasks with STD and MTT training. Underlined entries show best performance per dataset. The results suggest that none of the pre-training approaches are beneficial across all datasets.

Downstream baselines. We include KE-GCN (Yu et al., 2021a), a recent GNN with state-of-the-art results for graph alignment and entity classification. In contrast to KGEs, this model is directly trained on the downstream task (i.e., no pre-training) and needs to access the KG to perform predictions. For regression tasks, we use a linear layer after the final convolutional layer of KE-GCN, as this led to better performance in our experiments compared to using a single dimensional output in the final convolution layer as done by Huang et al. (2021). We tune hyperparameters using 30 SOBOL trials (as for KGE models); the search space is shown in Table 6.3. For training, evaluation, and model selection, we follow the approach for our downstream models (e.g,. 5-fold CV).

6.2.2 Model Performance

Overall performance. Table 6.4 shows the best downstream performance achieved by models that were pre-trained with STD and MTT training, each with the model selection approach that resulted in the best performance (LP or MTR). For each knowledge graph, column EC reports mean weighted F1 across all classification datasets, and column REG reports mean RSE across all regression datasets.

The results are different for each knowledge graph. On FB-237, pre-training with MTT is more beneficial on both types of downstream tasks. On YAGO3-10, both pre-training approaches provide similar results, while on WIKI5M pre-training with STD is more beneficial. **These results suggest that neither STD nor MTT are generally suitable pre-training approaches for optimizing downstream performance.** For LP, this means that link prediction is too restricted if the goal is using the learned representations for downstream applications. For MTR, this means that capturing more of the graph does not necessarily translate to better downstream performance. We discuss these pre-training approaches as predictive of good downstream performance further in Section 6.2.3. Note that in our study, MTR is computed over the set of tasks used during pre-training (see Table 5.1), and that using a different set of tasks during pre-training and/or model selection may provide different results. We discuss the choice of pre-training tasks for optimizing downstream application in Section 6.2.4.

Model performance. Table 6.5 shows mean performance and standard deviation across all downstream tasks for each KGE model. As before, bold entries show best performance per metric and evaluation method, and underlined entries facilitate performance comparisons across the different training approaches. We report performance for each individual downstream task in Appendix D.

The best overall downstream task performance across all KGE models was achieved by MTT in all cases, and often combined with the LP task for model selection. While the margin compared to STD was sometimes small (e.g., EC on YAGO3-10) and sometimes large (e.g., REG on FB1K-237), STD training, i.e. training only for link prediction, resulted in better downstream performance only a few times compared to MTT. Nevertheless, for a given KGE model, STD training occasionally performed better than MTT (e.g., DistMult and TransE on EC tasks for FB-237 and YAGO3-10, respectively). Given the higher cost of MTT during pre-training, the trade-off for downstream performance improvement may not always be beneficial. Note that using MTT with a different set of pre-training tasks may be not only be more efficient, but may result in better performance. **Ultimately, we conclude that the choice of pre-training objective clearly has an impact on downstream performance, although it is currently unclear how to make this choice in order to maximize downstream performance.** Although we explore this further in Section 6.2.4, our results show that pre-training KGEs

		Training	*Selection*	*Downstream tasks* EC (↑)	REG (↓)
FB-237	ComplEx	STD	LP	.844±.008	.447±.051
		MTT	LP	.858±.005	**.394±.057**
	DistMult	STD	LP	.873±.009	.551±.062
		MTT	LP	.865±.005	.472±.026
	RotatE	STD	LP	.868±.003	.797±.286
		MTT	LP	**.890±.003**	.573±.062
	TransE	STD	LP	.873±.015	.742±.287
		MTT	MTR	.878±.009	.681±.095
	KE-GCN			.829±.526	.501±.001
YAGO3-10	ComplEx	STD	LP	.712±.008	.589±.023
		MTT	MTR	.729±.005	.459±.020
	DistMult	STD	LP	.734±.003	.519±.019
		MTT	LP	**.746±.006**	.472±.029
	RotatE	STD	LP	.701±.002	.696±.018
		MTT	MTR	**.746±.001**	.470±.017
	TransE	STD	LP	.742±.002	.447±.036
		MTT	LP	.723±.004	.441±.029
	KE-GCN			.700±.223	**.398±.008**
WIKI5M	ComplEx	STD	LP	–	.687±.032
		MTT	MTR	–	.720±.023
	TransE	STD	LP	–	**.596±.011**
		MTT	MTR	–	650±.018
	KE-GCN†			–	–

† Not evaluated due to OOM.

Table 6.5: Average performance on test data of all downstream tasks per dataset for models with STD and MTT training, as well as KE-GCN by Yu et al. (2021a). Using STD models almost never results in better downstream performance.

for optimizing their use as KG representations is an open problem worthy of more attention. In this thesis, we provide code and datasets to further explore this issue.

Downstream baseline performance. Table 6.5 also includes our GNN-based baseline KE-GCN (Yu et al., 2021a) that trains directly on the downstream tasks, i.e. it is not a pre-trained model. **When comparing KGE performance with KE-GCN, KGE models outperform KE-GCN almost every time.** The exception is REG on YAGO3-10, where KE-GCN outperforms KGE models. These results suggest that the information captured by KGE models during pre-training is useful for simple downstream models to not only be competitive with, but even outperform, more involved downstream models that train directly on the task.

6.2.3 Impact of Model Selection

Pre-training for downstream applications. Table 6.6 reports the performance of all models on the two main self-supervised pre-training tasks: LP and MTR, and on the two sets of downstream tasks: EC and REG. Each column reports performance of the same set of 8 selected models as in previous sections. To facilitate comparison of relative model performance across tasks, models are sorted in decreasing order. Where applicable, we include KE-GCN.

The result show that neither LP nor MTR are suitable methods for model selection when the goal is to optimize downstream performance. This is clear because the best models for LP or MTR are almost never the best models for EC or REG. In fact, models with weaker performance during pre-training with both STD and MTT often performed competitively in downstream tasks and sometimes even outperformed models with stronger pre-training performance. For example, ComplEx considerably outperformed RotatE and TransE on LP and MTR on both FB-237 and YAGO3-10, but both models outperformed ComplEx on the EC tasks for those datasets. Similar observations can be made about both REG tasks on YAGO3-10. The REG tasks on FB-237 were an exception though; here higher performance during pre-training translated to better performance on downstream tasks, as one would expect. **Generally, these results are problematic, as they suggest that MTR and, perhaps more importantly, LP are often inadequate tasks to guide the choice of the more suitable KGE model class for downstream applications, a problem that needs further exploration in future work.**

	Sorted Performance for each Pre-Training and Downstream Task							
	Graph-structure				*Downstream Tasks*			
	LP (↑)		MTR (↑)		EC (↑)		REG (↓)	
FB-237	Compl. STD	.346	Compl. MTT	.606	RotatE MTT	.890	Compl. MTT	.394
	DistM. STD	.342	RotatE MTT	.598	TransE MTT	.878	Compl. STD	.447
	Compl. MTT	.331	DistM. MTT	.577	TransE STD	.873	DistM. MTT	.472
	TransE STD	.330	TransE MTT	.555	DistM. STD	.873	KE-GCN	.501
	DistM. MTT	.327	RotatE STD	.342	RotatE STD	.868	DistM. STD	.551
	RotatE MTT	.314	TransE STD	.332	DistM. MTT	.865	RotatE MTT	.573
	RotatE STD	.312	Compl. STD	.274	Compl. MTT	.858	TransE MTT	.681
	TransE MTT	.288	DistM. STD	.139	Compl. STD	.844	TransE STD	.742
					KE-GCN	.829	RotatE STD	.797
YAGO3-10	Compl. STD	.550	DistM. MTT	.759	DistM. MTT	.746	KE-GCN	.398
	DistM. STD	.539	Compl. MTT	.749	RotatE MTT	.746	TransE MTT	.441
	Compl. MTT	.538	RotatE MTT	.482	TransE STD	.742	TransE STD	.447
	DistM. MTT	.536	TransE MTT	.457	DistM. STD	.734	Compl. MTT	.459
	TransE STD	.504	RotatE STD	.432	Compl. MTT	.729	RotatE MTT	.470
	RotatE STD	.436	DistM. STD	.429	TransE MTT	.723	DistM. MTT	.472
	RotatE MTT	.427	Compl. STD	.411	Compl. STD	.712	DistM. STD	.519
	TransE MTT	.048	TransE STD	.349	RotatE STD	.701	Compl. STD	.589
					KE-GCN	.700	RotatE STD	.696

Table 6.6: Sorted performance on graph-structure prediction and downstream tasks of KGE models, and KE-GCN (Yu et al., 2021a). Relative model performance given by LP or MTR differs from that given by downstream performance, suggesting that neither LP nor MTR are generally useful for model selection.

		Selection Method			
		EC - Weighted F1 (↑)		*REG - RSE (↓)*	
		LP	MTR	LP	MTR
ComplEx	STD	.844	.858	**.447**	.545
DistMult	STD	**.873**	.836	.551	.686

Table 6.7: Performance on FB-237 downstream tasks for STD training and two model selection approaches: LP and MTR. On both types of tasks, the best performance is obtained by combining STD training with LP model selection.

Further model selection approaches. For completeness, we also explored the impact of further combinations of model selection methods with both STD and MTT training. To explore whether there would be improvements in STD models when selecting them based on performance on the MTR task, Table 6.7 reports downstream performance of some KGE models using STD training combined with either LP or MTR for model selection. **We see that the combination of STD with MTR leads to lower downstream performance almost every time.**

Model selection using downstream information. To explore whether results can improve by using information from downstream tasks to select models, Table 6.8 reports performance on FB-237 of some KGE models using both training approaches with either LP for model selection (which consistently provided better results for these models with both training approaches) or by selecting directly on the metric used to evaluate the downstream task (weighted F1 for entity classification and RSE for regression). **We found that model selection with the downstream task metric provides only marginal benefits for both STD and MTT and can in fact be detrimental, likely due to overfitting on validation data.** This suggests that model selection without information about downstream tasks—i.e., using LP or MTR—may be preferrable to using downstream information. This is convenient, as including downstream information during pre-training or model selection would likely make the resulting representations less general. In addition, it is often the case that the downstream task is not known at the time of pre-training the model (El-Kishky et al., 2022).

Overall, we found that full MTT training with LP for model selection was

		Selection Method			
		EC - Weighted F1 (↑)		*REG - RSE (↓)*	
		LP	Weighted F1	LP	RSE
ComplEx	STD	.844	.850	.447	.437
	MTT	.858	.827	.394	**.393**
DistMult	STD	**.873**	.846	.551	.539
	MTT	.865	.864	.472	.476

Table 6.8: Performance on FB-237 downstream tasks for different KGE model training (STD and MTT) and two model selection approaches: LP and weighted F1 or RSE. Using downstream task data for model selection provides only marginal gains and is sometimes detrimental to downstream performance, likely due to overfitting on validation data.

a suitable choice, but we show in the next section that further improvements are possible by dataset-, model- and task-specific choices of pre-training task.

6.2.4 Impact of Pre-Training Task Selection

Table 6.9 summarizes our results from exploring the impact of pre-training task selection and, in particular, whether all proposed MTT tasks are beneficial. To keep computational costs feasible, we focused on FB-237. We explored performance using MTT with all available pre-training tasks, and MTT without either the LP, REL, DOM, NBE, or NBR pre-training task, as well as without LP+REL or without DOM+NBR. In every case, we selected best models based on their performance on MTR over the corresponding set of pre-training tasks. We report only on the sets of tasks that provided relevant results for our discussion. All combinations of pre-training tasks are reported in Table D.8 in Appendix D.

We observe that the choice of training tasks can have a significant impact on downstream performance and that good choices differ between KGE models and the type of downstream task. For example, compared to full MTT training, using a subset of tasks led to large improvements in almost all cases, except for ComplEx on REG. That is, the best performance in most cases is obtained by removing one of the tasks during training. In particular,

	Training	*Selection*	*Downstream tasks*	
			EC (↑)	REG (↓)
ComplEx	STD	LP	.844±.008	.447±.051
	MTT	MTR	.843±.002	**.412±.037**
	w/o LP	MTR	.870±.002	.512±.044
	w/o NBE	MTR	.856±.002	.562±.038
	w/o LP+REL	MTR	.849±.011	.542±.054
DistMult	STD	LP	.873±.009	.551±.062
	MTT	MTR	.857±.006	.482±.026
	w/o LP	MTR	.861±.008	.522±.067
	w/o DOM	MTR	.849±.002	.478±.027
	w/o NBE	MTR	.844±.002	.524±.047
RotatE	STD	LP	.868±.003	.797±.286
	MTT	MTR	.847±.001	.704±.060
	w/o LP	MTR	.874±.000	.661±.043
	w/o DOM	MTR	**.898±.001**	.593±.078
	w/o NBR	MTR	.863±.007	.552±.035
TransE	STD	LP	.873±.015	.742±.287
	MTT	MTR	.878±.009	.681±.095
	w/o LP	MTR	.870±.000	.486±.027
	w/o NBE	MTR	.884±.002	.463±.032
	w/o NBR	MTR	.857±.007	.458±.024

Table 6.9: Performance on downstream tasks for FB-237 using different pre-training objectives and model selection. Excluding the LP task in pre-training often improves performance.

excluding the LP task during pre-training resulted in better downstream performance compared to STD and full MTT training half of the time, suggesting that including the LP task during pre-training can be detrimental to downstream performance. **These results show that including more tasks during pre-training does not necessarily lead to higher downstream performance, and they also provide more evidence that STD training is not enough for good downstream task performance.**

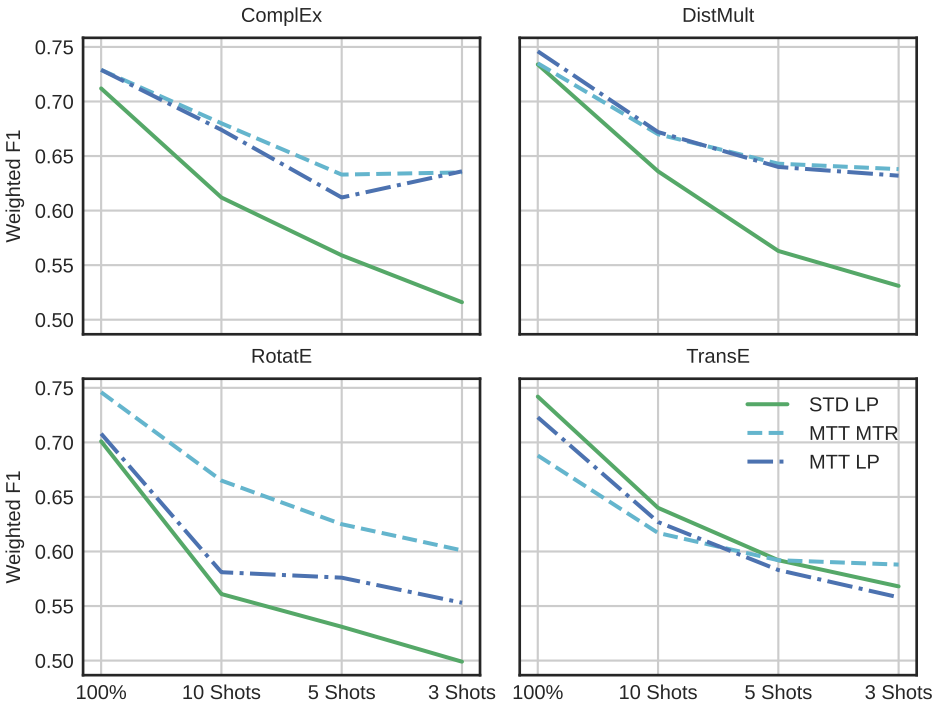


Figure 6.2: Few-shot performance of entity classification tasks for YAGO3-10 (higher is better). Each n-shot training set consists of n sampled positive and negative examples for each class. The gap in performance between MTT and STD models becomes larger as training data becomes less available.

6.2.5 Data Efficiency Tests

As part of our study, we also tested whether KGE models that capture more information during training are more beneficial as pre-trained models when there is a small amount of downstream data available. To this end, we reduced the amount of training data in our downstream datasets in two different ways.

First, we tested models in a few-shot scenario. That is the setting where there are only a handful of examples available for the downstream task. For classification, we sampled n positive and n negative examples per class, where $n \in \{3, 5, 10\}$. Figure 6.2 shows the results for the YAGO3-10 classification tasks (higher is better). We found that, indeed, as less data becomes available, the average performance of STD models over the set of downstream

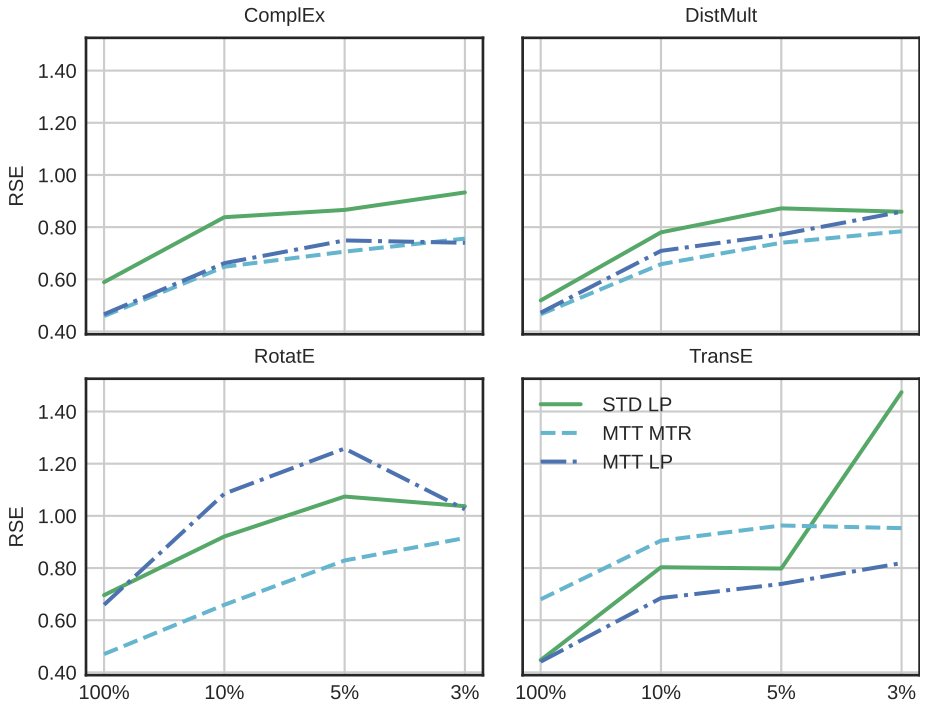


Figure 6.3: Performance of regression tasks for YAGO3-10 with downsampled training sets (lower is better). Each training set was constructed by sampling a percentage of the training set. The gap in performance between MTT and STD models becomes larger as training data becomes less available.

classification tasks becomes considerably lower than when using pre-trained models with MTT. The exception is TransE, where the performance difference is not as significant, but it is still different from the setting with a complete training set. We observed the same pattern in FB-237 (reported in Figure D.1 in Appendix D).

The few-shot scenario applied to regression tasks yielded unsatisfactory models almost every time. We say this because almost all models gave an RSE value considerably above 1. We thus constructed a different scenario with scarce training data. We randomly sampled n% of the training set, where again $n \in \{3, 5, 10\}$. Figure 6.3 shows the results for the YAGO3-10 regression tasks (lower is better). This time, MTT seems to be more beneficial as training data becomes less available for Transe. For the other models, the trend observed with a complete training set is mostly maintained. We do see

more benefits in the FB-237 dataset, reported in Figure D.4 in Appendix D. But these results on regression tasks show that it is not always evident that MTT models are more beneficial with less training data. Still, at no point do models pre-trained with STD become a better choice. In addition, when applying this sampling setting to the classification datasets (reported in Figures D.2 and D.2 in Appendix D), we observed the same pattern as in the few-shot setting, albeit to a lesser extent. **Overall, although not every time, we observed the clear trend that MTT models become more beneficial than STD models as less training data for downstream tasks becomes available, especially for the classification tasks in our tests.**

6.3 Related Work

To our knowledge, there is almost no prior work on studying embedding quality in the context of KGE models.

Some studies focus on understanding KGE performance at a more fine-grained level (Widjaja et al., 2022; Rim et al., 2021; Meilicke et al., 2018). They do so by looking at link prediction performance on different types of relations, e.g. symmetric relations. These studies focus on link prediction performance and do not study the quality of resulting embeddings in downstream applications.

More closely related to our work, Pezeshkpour et al. (2018) proposed KGE models that can handle numerical relations, i.e. relations that associate entities to numerical values, such as a person's age or height. To test this ability, they evaluated their models on regression tasks. We use their datasets in our study. The work from Jain et al. (2021) is the most similar to the work presented in this chapter. They evaluated the pre-trained embeddings from our study in Chapter 3 (i.e. trained on link prediction) on entity classification tasks. We also use their datasets in our study.

In this chapter, we expanded on the work from Jain et al. (2021) by using a larger set of pre-training tasks, as well as more variety and a higher number of downstream tasks. Further, our main focus is on the impact of different pre-training objectives and model selection approaches on downstream task performance.

6.4 Summary

In this chapter, we studied the impact that different pre-training objectives and model selection methods have on downstream task performance. For this purpose, we conducted a large experimental study where we either created, or collected from the literature, 35 different datasets for two different types of downstream tasks: entity classification and regression. We tested the performance of several downstream models that use KGE models as input features to solve these downstream tasks. As pre-trained models, we used KGE models that were trained with the standard link prediction (LP) approach, as well as various forms of the multi-task training (MTT) approach we proposed in Chapter 5.

We found that strong performance during pre-training with both LP and MTT often does not translate to strong performance in downstream applications. In addition, we found that using more tasks during pre-training with MTT is often not the best choice, and that excluding the standard LP task during pre-training often improves downstream performance. When compared to performance of a GCN-based model that trains directly on the downstream task, we found that KGEs often outperform this model, suggesting that KGEs are useful pre-trained models for solving downstream tasks at a reduced cost. In short, although our work makes a step toward understanding the impact of pre-training objectives for KGE models, these results also suggest that more research is needed on the relation between the choice of training objectives and the suitability of resulting KGE models for downstream applications.

CHAPTER

SEVEN

CONCLUSIONS

In this thesis, we proposed new training and evaluation methods for KGE models and conduct several experimental studies that bring to light a few issues with the current state of KGE research. First, we conduct a large-scale experimental study that compares several popular KGE models all under the same training conditions. We find that performance on link prediction, the most commonly studied task for KGE models, is highly sensitive to training conditions, and that new training settings, as opposed to new model architectures, may account for much of the progress reported in recent years. As evidence of this, our results show that, unlike previous results, models that were released many years ago are competitive with, or even outperform, state-of-the-art models when trained with more recent approaches. This highlights the need for future studies to implement their own baselines instead of using previously published results, as this provides a clearer picture of what each experimental component contributes to the reported results.

In addition, we proposed new evaluation methods that highlight the potentially negative impact that the focus on the link prediction task has on KGE research. Namely, the entity-pair ranking protocol that is based on the more general knowledge base completion task, and the multi-task ranking protocol, designed to aggregate ranking performance over any number of tasks. Our results with the entity-pair ranking protocol show that the link prediction task, as commonly defined in the literature, is a constrained form of predicting missing links, and thus not generally indicative of how useful a

KGE model is when adding missing links to an incomplete knowledge graph. Similarly, our results with the multi-task ranking evaluation, in combination with a set of simple graph-structure prediction tasks we proposed, show that link prediction is not indicative of a model's ability to capture the general properties in a knowledge graph, which is one of the main arguments behind the study of KGE models. These results suggest that the focus on link prediction is too narrow, especially if the goal is to learn generally useful representations of knowledge graphs.

Finally, we proposed a generalized training approach that allows the training of KGE models on multiple tasks simultaneously, and tested the impact that different pre-training approaches have on downstream applications. Despite the use of KGE models as pre-trained representations of knowledge graphs, ours is the first study that looks into the relation between different pre-training approaches and model usability in downstream applications. We found that standard link prediction performance is not indicative of good downstream performance, and that while using more tasks during pre-training often yields better results, the best performance is not obtained by pre-training models with as many tasks as possible. These results highlight the need for more research into understanding how to pre-train KGE models, so they provide generally useful representations for downstream applications. To this end, we provide highly modular code and a suite of benchmark datasets as part of the open source framework LibKGE, all of which should be helpful for future research in this direction.

Future Work

As a result of this thesis, there are a number of interesting research directions for future work. We briefly discuss some of them in the following.

Knowledge Base Completion

As discussed in Chapter 4, the knowledge base completion (KBC) task provides a way to more generally assess whether a model can successfully add missing links to an existing knowledge graph. However, model performance on this task is still quite low, especially when compared with rule-based models that clearly outperform KGE models. Low performance on this task

may result in models that add a considerable amount of noise when adding new facts to a knowledge base.

Consequently, and much in the same way that link prediction guided the design of training methods, the KBC task may inspire new types of training approaches that could yield more useful models for knowledge base completion. However, the gap in performance compared to rule-based models as reported in Section 4.3 suggests that this is a challenging research direction. Specific challenges may be the cost of the KBC task, which may also translate to new training approaches, and the impact that the closed world assumption commonly used in KGE research may have when dealing with the entire known graph, as required for this task.

Pre-Training KGE Models

It is common in other fields such as natural language processing to test the quality of pre-trained representations in downstream applications (Mikolov et al., 2013b; Devlin et al., 2019). In addition, it is known that the choice of pre-training objective has an impact on the resulting representations, which is why pre-training objectives are often designed to fit a particular goal (Devlin et al., 2019; Radford et al., 2018). Although we take the first steps in studying similar effects in KGE models, our work highlights the need to focus on this open problem if we aim at using KGE models as pre-trained representations of knowledge graphs. To this end, new types of training objectives may be tested either in isolation or combined with existing ones using our proposed multi-task training approach. Existing approaches that may provide interesting results may be KGE models that are trained on multi-hop query answering (Arakelyan et al., 2020) or other tasks that encourage models to capture different properties in the graph. Similarly, training with supervised objectives may have a significant impact on downstream performance, but perhaps at the expense of obtaining more general representations. Such directions have already been explored in natural language processing (Aribandi et al., 2022; Sanh et al., 2022).

Comparison to Graph Neural Networks

Graph neural networks (GCNs) have recently emerged as a family of models that can perform link prediction (Zhu et al., 2021) and other types of

graph-related tasks, such as entity classification (Yu et al., 2021a) or graph classification (Vashishth et al., 2020). In this work, we compared the use of pre-trained KGE models with a state-of-the-art GCN-based model designed for entity classification. However, given the overlap in tasks with KGE models, a large-scale comparative study of KGE and GCN-based models would be a suitable way to more generally compare and contrast the benefits of each family of approaches. Existing comparative studies have already focused on graph-neural networks for graph classification (Errica et al., 2019) and link prediction on standard graphs (Li et al., 2023), but to our knowledge, no large-scale study focuses on knowledge graph tasks, and includes KGE models as baselines.

BIBLIOGRAPHY

Ralph Abboud, Ismail Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. Boxe: A box embedding model for knowledge base completion. *Advances in Neural Information Processing Systems*, 2020.

Farahnaz Akrami, Lingbing Guo, Wei Hu, and Chengkai Li. Re-evaluating embedding-based knowledge graph completion methods. In *International Conference on Information and Knowledge Management*, 2018.

Marjan Albooyeh, Rishab Goel, and Seyed Mehran Kazemi. Out-of-sample representation learning for knowledge graphs. In *Findings of the Association for Computational Linguistics (EMNLP 2020)*, 2020.

Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Mikhail Galkin, Sahand Sharifzadeh, Asja Fischer, Volker Tresp, and Jens Lehmann. Bringing light into the dark: A large-scale evaluation of knowledge graph embedding models under a unified framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Carl Allen, Ivana Balazevic, and Timothy Hospedales. Interpreting knowlege graph relation representation from word embeddings. In *International Conference on Learning Representations (ICLR)*, 2021.

Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. Complex query answering with neural link predictors. In *International Conference on Learning Representations (ICLR)*, 2020.

Vamsi Aribandi, Yi Tay, Tal Schuster, Jinfeng Rao, Huaixiu Steven Zheng, Sanket Vaibhav Mehta, Honglei Zhuang, Vinh Q Tran, Dara Bahri, Jianmo

Ni, et al. ExT5: Towards extreme multi-task scaling for transfer learning. In *International Conference on Learning Representations (ICLR)*, 2022.

Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambridge, United Kingdom, 2017.

Stephan Baier, Yunpu Ma, and Volker Tresp. Improving visual relationship detection using semantic modeling of scene descriptions. In *International Semantic Web Conference (ISWC)*, 2017.

Ivana Balazevic, Carl Allen, and Timothy Hospedales. TuckER: Tensor factorization for knowledge graph completion. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

Yoshua Bengio. Neural net language models. *Scholarpedia*, 2008.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 2000.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012.

Kurt Bollacker, Robert Cook, and Patrick Tufts. Freebase: A shared database of structured general human knowledge. In *AAAI*, 2007.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *International Conference on Management of Data ACM SIGMOD*, 2008.

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

Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data: Application to word-sense disambiguation. *Machine Learning*, 2013a.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in Neural Information Processing Systems (NIPS)*, 2013b.

Samuel Broscheit, Daniel Ruffinelli, Adrian Kochsiek, Patrick Betz, and Rainer Gemulla. Libkge-a knowledge graph embedding library for reproducible research. In *Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP)*, 2020.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Transactions on Knowledge and Data Engineering*, 1989.

David Chang, Ivana Balažević, Carl Allen, Daniel Chawla, Cynthia Brandt, and Richard Andrew Taylor. Benchmark and best practices for biomedical knowledge graph embeddings. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

Spencer Chang. Scaling Knowledge Access and Retrieval at Airbnb, 2018. URL https://medium.com/airbnb-engineering/scaling-knowledge-access-and-retrieval-at-airbnb-665b6ba21e95. Accessed: 2023-08-17.

Sanxing Chen, Xiaodong Liu, Jianfeng Gao, Jian Jiao, Ruofei Zhang, and Yangfeng Ji. HittER: Hierarchical transformers for knowledge graph embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2021a.

Yao Chen, Jiangang Liu, Zhe Zhang, Shiping Wen, and Wenjun Xiong. Möbiuse: Knowledge graph embedding on möbius ring. *Knowledge-Based Systems*, 2021b.

Yihong Chen, Pasquale Minervini, Sebastian Riedel, and Pontus Stenetorp. Relation prediction as an auxiliary training objective for improving multi-relational graph representations. In *3rd Conference on Automated Knowledge Base Construction*, 2021c.

Fariz Darari, Werner Nutt, Giuseppe Pirrò, and Simon Razniewski. Completeness Management for RDF Data Sources. *ACM Trans. Web*, 2018.

Shib Sankar Dasgupta, Swayambhu Nath Ray, and Partha Talukdar. Hyte: Hyperplane-based temporally aware knowledge graph embedding. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Luc De Raedt. *Logical and relational learning*. Springer Science & Business Media, 2008.

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

Jacob Devlin, Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NA-ACL)*, 2019.

Carl Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. In *International Conference on Computer Vision*, 2017.

John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

Ahmed El-Kishky, Thomas Markovich, Serim Park, Chetan Verma, Baekjin Kim, Ramy Eskander, Yury Malkov, Frank Portman, Sofía Samaniego, Ying Xiao, and Aria Haghighi. Twhin: Embedding the twitter heterogeneous information network for personalized recommendation. In *Conference on Knowledge Discovery and Data Mining ACM SIGKDD*, 2022.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *International Conference on Learning Representations (ICLR)*, 2019.

Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *International Conference on World Wide Web (WWW)*, 2013.

Mikhail Galkin, Max Berrendorf, and Charles Tapley Hoyt. An open challenge for inductive link prediction on knowledge graphs. In *2nd Workshop on Graph Learning Benchmarks (GLB@WWW22)*, 2022.

Alberto García-Durán, Sebastijan Dumancic, and Mathias Niepert. Learning sequence encoders for temporal knowledge graph completion. In *EMNLP*, 2018.

Matt Gardner, Partha Talukdar, Jayant Krishnamurthy, and Tom Mitchell. Incorporating vector space similarity in random walk inference over knowledge bases. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

Kiril Gashteovski, Rainer Gemulla, and Luciano del Corro. MinIE: Minimizing Facts in Open Information Extraction. Association for Computational Linguistics (ACL), 2017.

Aryo Pradipta Gema, Dominik Grabarczyk, Wolf De Wulf, Piyush Borole, Javier Antonio Alfaro, Pasquale Minervini, Antonio Vergari, and Ajitha Rajan. Knowledge graph embeddings in the biomedical domain: Are they useful? a look at link prediction, rule learning, and downstream polypharmacy tasks. *arXiv preprint arXiv:2305.19979*, 2023.

GOC. The Gene Ontology Resource: enriching a GOld mine. *Nucleic Acids Research*, 2021.

GOC. GO FAQs, 2023. URL http://geneontology.org/docs/faq/. Accessed: 2023-03-15.

Rishab Goel, Seyed Mehran Kazemi, Marcus Brubaker, and Pascal Poupart. Diachronic embedding for temporal knowledge graph completion. In *AAAI Conference on Artificial Intelligence*, 2020.

Shu Guo, Quan Wang, Bin Wang, Lihong Wang, and Li Guo. Semantically smooth knowledge graph embedding. In *Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2015.

Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Jointly embedding knowledge graphs and logical rules. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

Ferras Hamad, Isaac Liu, and Xian Xing Zhang. Food Discovery with Uber Eats: Building a Query Understanding Engine, 2018. URL https://www.uber.com/en-DE/blog/uber-eats-query-understanding/. Accessed: 2023-08-17.

Bin He, Di Zhou, Jinghui Xiao, Xin Jiang, Qun Liu, Nicholas Jing Yuan, and Tong Xu. Bert-mk: Integrating graph contextualized knowledge into pre-trained language models. In *Findings of the Association for Computational Linguistics EMNLP*, 2020.

Qi He, Bee-Chung Chen, and Deepak Agarwal. Building The LinkedIn Knowledge Graph, 2016. URL https://engineering.linkedin.com/blog/2016/10/building-the-linkedin-knowledge-graph. Accessed: 2023-08-17.

Geoffrey E Hinton. Distributed representations. 1984.

Pascal Hitzler. A review of the semantic web field. *Communications of the ACM*, 2021.

Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. *Foundations of Semantic Web Technologies*. 2010. URL http://www.semantic-web-book.org/.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. Knowledge Graphs. *ACM Computing Surveys (CSUR)*, 2021.

Han Huang, Leilei Sun, Bowen Du, Chuanren Liu, Weifeng Lv, and Hui Xiong. Representation learning on knowledge graphs for node importance estimation. In *Conference on Knowledge Discovery & Data Mining (ACM SIGKDD)*, 2021.

Ihab F Ilyas, Theodoros Rekatsinas, Vishnu Konda, Jeffrey Pound, Xiaoguang Qi, and Mohamed Soliman. Saga: A platform for continuous construction and serving of knowledge at scale. 2022.

Nitisha Jain, Jan-Christoph Kalo, Wolf-Tilo Balke, and Ralf Krestel. Do embeddings actually capture knowledge graph semantics? In *Extended Semantic Web Conference (ESWC)*, 2021.

Dora Jambor, Komal Teru, Joelle Pineau, and William L Hamilton. Exploring the limits of few-shot link prediction in knowledge graphs. In *Conference of the European Chapter of the Association for Computational Linguistics (E-ACL)*, 2021.

Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and S Yu Philip. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

Zhen Jia, Soumajit Pramanik, Rishiraj Saha Roy, and Gerhard Weikum. Complex temporal question answering on knowledge graphs. In *International Conference on Information & Knowledge Management (CIKM)*, 2021.

Armand Joulin, Laurens van der Maaten, Allan Jabri, and Nicolas Vasilache. Learning visual features from large weakly supervised data. In *European Conference on Computer Vision (ECCV)*, 2016.

Rudolf Kadlec, Ondrej Bajgar, and Jan Kleindienst. Knowledge base completion: Baselines strike back. In *2nd Workshop on Representation Learning for NLP (Rep4NLP@ACL)*, 2017.

Uday Kamath, John Liu, and James Whitaker. *Deep learning for NLP and speech recognition*. Springer, 2019.

Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 2009.

Bhushan Kotnis and Vivi Nastase. Learning knowledge graph embeddings with type regularizer. In *Knowledge Capture Conference*, 2017.

Bhushan Kotnis and Vivi Nastase. Analysis of the impact of negative sampling on link prediction in knowledge graphs. In *1st Workshop on Knowledge Base Construction, Reasoning and Mining (KBCOM@WSDM)*, 2018.

Bhushan Kotnis, Carolin Lawrence, and Mathias Niepert. Answering complex queries in knowledge graphs with bidirectional sequence encoders. In *AAAI Conference on Artificial Intelligence*, 2021.

Arun Krishnan. Making Search Easier: How Amazon's Product Graph is Helping Customers Find Products More Easily. Amazon Blog, August 17 2018. URL https://blog.aboutamazon.com/innovation/making-search-easier.

Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In *International Conference on Machine Learning (ICML)*, 2018.

Ni Lao and William W Cohen. Relational retrieval using a combination of path-constrained random walks. *Machine Learning*, 2010.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 2015.

Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. PyTorch-BigGraph: A Large-scale Graph Embedding System. In *Conference on Systems and Machine Learning (SysML)*, 2019.

Fengying Li, Mingdong Chen, and Rongsheng Dong. Multi-hop Question Answering with Knowledge Graph Embedding in a Similar Semantic Space. In *International Joint Conference on Neural Networks (IJCNN)*, 2022.

Juanhui Li, Harry Shomer, Haitao Mao, Shenglai Zeng, Yao Ma, Neil Shah, Jiliang Tang, and Dawei Yin. Evaluating graph neural networks for link prediction: Current pitfalls and new benchmarking. *arXiv preprint arXiv:2306.10453*, 2023.

Zelong Li, Jianchao Ji, Zuohui Fu, Yingqiang Ge, Shuyuan Xu, Chong Chen, and Yongfeng Zhang. Efficient non-sampling knowledge graph embedding. In *International Conference on World Wide Web (WWW)*, 2021.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI Conference on Artificial Intelligence*, 2015.

Hanxiao Liu, Yuexin Wu, and Yiming Yang. Analogical inference for multi-relational embeddings. In *International Conference on Machine Learning (ICML)*, 2017.

Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are GANs created equal? A large-scale study. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

Jhomara Luzuriaga, Emir Munoz, Henry Rosales-Mendez, and Aidan Hogan. Merging Web Tables for Relation Extraction with Knowledge Graphs. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

Dhruv Mahajan, Ross Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. In *European Conference on Computer Vision (ECCV)*, 2018.

Farzaneh Mahdisoltani, Joanna Biega, and Fabian Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *7th biennial conference on innovative data systems research*, 2014.

Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. Fine-Grained Evaluation of Rule- and Embedding-based Systems for Knowledge Graph Completion. In *International Semantic Web Conference (ISWC)*, 2018.

Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

Christian Meilicke, Patrick Betz, and Heiner Stuckenschmidt. Why a naive way to combine symbolic and latent knowledge base completion works surprisingly well. In *3rd Conference on Automated Knowledge Base Construction*, 2021.

Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations (ICLR)*, 2017.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 2013a.

Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Conference of the North American Chapter of the Association for Computational Linguistics (NA-ACL)*, 2013b.

George A Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 1995.

Sameh Mohamed, Vít Nováček, Pierre-Yves Vandenbussche, and Emir Muñoz. Loss functions in knowledge graph embedding models. In *Workshop on Deep Learning for Knowledge Graphs (DL4KG2019)*, 2019.

Jason Mohoney, Roger Waleffe, Henry Xu, Theodoros Rekatsinas, and Shivaram Venkataraman. Marius: Learning massive graph embeddings on a single machine. In *Symposium on Operating Systems Design and Implementation ({OSDI})*, 2021.

Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. A novel embedding model for knowledge base completion based on convolutional neural network. In *Conference of the North American Chapter of the Association for Computational Linguistics (NA-ACL)*, 2018.

Maximilian Nickel, Volker Tresp, Hans-Peter Kriegel, et al. A three-way model for collective learning on multi-relational data. In *International Conference in Machine Learning (ICML)*, 2011.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE*, 2015.

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

Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. Industry-scale knowledge graphs: Lessons and challenges: Five diverse technology companies show how it's done. *Queue*, 2019.

Peter Orbanz and Daniel M Roy. Bayesian models of graphs, arrays and other exchangeable random structures. *IEEE transactions on pattern analysis and machine intelligence*, 2014.

Peter Orbanz and Yee Whye Teh. Bayesian nonparametric models. *Encyclopedia of machine learning*, 2010.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 2022.

Alberto Paccanaro and Geoffrey E Hinton. Learning hierarchical structures with linear relational embedding. *Advances in neural information processing systems*, 2001.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.

Thomas Pellissier Tanon, Denny Vrandečić, Sebastian Schaffert, Thomas Steiner, and Lydia Pintscher. From Freebase to Wikidata: The Great Migration. In *International Conference on World Wide Web (WWW)*, 2016.

Boya Peng, Yejin Huh, Xiao Ling, and Michele Banko. Improving Knowledge Base Construction from Robust Infobox Extraction. In *Conference of the North American Chapter of the Association for Computational Linguistics (NA-ACL)*, 2019.

Pouya Pezeshkpour, Liyan Chen, and Sameer Singh. Embedding multimodal relational data for knowledge base completion. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. In *Empirical Methods in NaturalLanguage Processing (EMNLP)*, 2017.

Wiem Ben Rim, Carolin Lawrence, Kiril Gashteovski, Mathias Niepert, and Naoaki Okazaki. Behavioral testing of knowledge graph embedding models for link prediction. In *3rd Conference on Automated Knowledge Base Construction*, 2021.

Petar Ristoski and Heiko Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In *International Semantic Web Conference (ISWC)*, 2016.

Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *Conference of the North American Chapter of the Association for Computational Linguistics (NA-ACL)*, 2015.

Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2021.

Daniel Ruffinelli and Rainer Gemulla. Beyond Link Prediction: On Pre-Training Knowledge Graph Embeddings. 2023. *Under Submission*.

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 (ICLR)*, 2020.

Tara Safavi and Danai Koutra. CoDEx: A Comprehensive Knowledge Graph Completion Benchmark. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Farnood Salehi, Robert Bamler, and Stephan Mandt. Probabilistic knowledge graph embeddings. In *1st Symposium on Advances in Approximate Bayesian Inference (AABI)*, 2018.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Teven Le Scao, Arun Raja, et al. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations (ICLR)*, 2022.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *Extended Conference in Semantic Web (ESWC)*, 2018.

Chao Shang, Yun Tang, Jing Huang, Jinbo Bi, Xiaodong He, and Bowen Zhou. End-to-end structure-aware convolutional networks for knowledge base completion. In *AAAI Conference on Artificial Intelligence*, 2019.

Anshumali Shrivastava and Ping Li. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Saurabh Shrivastava. Bring Rich Knowledge of People, Places, Things and Local Businesses to your Apps, 2017. URL https://blogs.bing.com/search-quality-insights/2017-07/bring-rich-knowledge-of-people-places-things-and-local-businesses-to-your-apps. Accessed: 2023-08-17.

Ajit P Singh and Geoffrey J Gordon. Relational learning via collective matrix factorization. In *International Conference on Knowledge Discovery and Data Mining ACM SIGKDD*, 2008.

Amit Singhal. Introducing the Knowledge Graph: Things, not Strings, 2012. URL https://blog.google/products/search/introducing-knowledge-graph-things-not/. Accessed: 2023-08-17.

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

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. YAGO: A Core of Semantic Knowledge. In *International Conference on World Wide Web (WWW)*, 2007.

Kai Sun, Yifan Ethan Xu, Hanwen Zha, Yue Liu, and Xin Luna Dong. Head-to-tail: How knowledgeable are large language models (llm)? a.k.a. will llms replace knowledge graphs? *arXiv preprint arXiv:2308.10168*, 2023.

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

Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. A re-evaluation of knowledge graph completion methods. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

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 (CVSC@ACL)*, 2015.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro,

Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

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

Théo Trouillon, Éric Gaussier, Christopher R Dance, and Guillaume Bouchard. On inductive abilities of latent factor models for relational learning. *Journal of Artificial Intelligence Research*, 2019.

Theo Van Veen. Wikidata. *Information technology and libraries*, 2019.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2020.

Denny Vrandečić and Markus Krötzsch. Wikidata: A Free Collaborative Knowledgebase. *Communications of the ACM*, 2014.

W3C. SPARQL 1.1 Query Language, 2013. URL https://www.w3.org/TR/sparql11-query/. Accessed: 2023-03-15.

W3C. RDF 1.1 Concepts and Abstract Syntax, 2014. URL https://www.w3.org/TR/rdf11-concepts/. Accessed: 2023-03-15.

Brian Walsh, Sameh K Mohamed, and Vít Nováček. BioKG: A Knowledge Graph for Relational Learning on Biological Data. In *International Conference on Information & Knowledge Management*, 2020.

Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Dkn: Deep knowledge-aware network for news recommendation. In *International Conference of the World Wide Web (WWW)*, 2018a.

Jie Wang, Zhanqiu Zhang, Zhihao Shi, Jianyu Cai, Shuiwang Ji, and Feng Wu. Duality-Induced Regularizer for Semantic Matching Knowledge Graph Embeddings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022a.

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 2017.

Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. KEPLER: A unified model for knowledge embedding and pre-trained language representation. *Transactions of the Association for Computational Linguistics*, 2021.

Xintao Wang, Qianyu He, Jiaqing Liang, and Yanghua Xiao. Language models as knowledge embeddings. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2022b.

Yanjie Wang, Rainer Gemulla, and Hui Li. On multi-relational link prediction with bilinear models. In *AAAI Conference on Artificial Intelligence*, 2018b.

Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, Samuel Broscheit, and Christian Meilicke. On Evaluating Embedding Models for Knowledge Base Completion. In *4th Workshop on Representation Learning for NLP (Rep4NLP@ACL)*, 2019. Received Outstanding Paper Award.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI Conference on Artificial Intelligence*, 2014.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 2022.

Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge base completion via search-based question answering. In *International Conference on World Wide Web (WWW)*, 2014.

Haris Widjaja, Kiril Gashteovski, Wiem Ben Rim, Pengfei Liu, Christopher Malon, Daniel Ruffinelli, Carolin Lawrence, and Graham Neubig. KGxBoard: Explainable and Interactive Leaderboard for Evaluation of Knowledge Graph Completion Models. In *Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP)*, 2022.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A Comprehensive Survey on Graph Neural Networks. *IEEE transactions on neural networks and learning systems*, 2020.

Han Xiao, Minlie Huang, Lian Meng, and Xiaoyan Zhu. Ssp: semantic space projection for knowledge graph embedding with text descriptions. In *AAAI Conference on Artificial Intelligence*, 2017.

Qizhe Xie, Xuezhe Ma, Zihang Dai, and Eduard Hovy. An Interpretable Knowledge Transfer Model for Knowledge Base Completion. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. Representation learning of knowledge graphs with entity descriptions. In *AAAI Conference on Artificial Intelligence*, 2016.

Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *International Conference on Learning Representations (ICLR)*, 2015.

Donghan Yu, Yiming Yang, Ruohong Zhang, and Yuexin Wu. Knowledge embedding based graph convolutional network. In *International Conference on the World Wide Web (WWW)*, 2021a.

Jinxing Yu, Yunfeng Cai, Mingming Sun, and Ping Li. Mquade: a unified model for knowledge fact embedding. In *International Conference on World Wide Web (WWW)*, 2021b.

Mengqi Zhang, Yuwei Xia, Qiang Liu, Shu Wu, and Liang Wang. Learning latent relations for temporal knowledge graph reasoning. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023.

Wen Zhang, Bibek Paudel, Wei Zhang, Abraham Bernstein, and Huajun Chen. Interaction embeddings for prediction and explanation in knowledge graphs. In *ACM International Conference on Web Search and Data Mining (WSDM)*, 2019a.

Zhanqiu Zhang, Jie Wang, Jieping Ye, and Feng Wu. Rethinking graph convolutional networks in knowledge graph completion. In *International Conference on World Wide Web (WWW)*, 2022.

Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. Ernie: Enhanced language representation with informative entities. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019b.

Yu Zhao, Sheng Gao, Patrick Gallinari, and Jun Guo. Zero-shot embedding for unseen entities in knowledge graph. *IEICE TRANSACTIONS on Information and Systems*, 2017.

Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. DGL-KE: Training Knowledge Graph Embeddings at Scale. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 2020.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. 2021.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 2021.

LIST OF ALGORITHMS

LIST OF FIGURES

LIST OF TABLES

APPENDICES

A Additional Material for Chapter 3

		RESCAL	TransE	DistMult	ComplEx	ConvE	RotatE
FB15K-237	Learning rate	0.00074	0.12197	0.15954	0.18255	0.00428	0.22032
	Sched. patience	1	6	6	7	9	6
	Ent. reg. weight	4.70^{-11}	1.40^{-07}	1.41^{-09}	6.70^{-09}	4.30^{-15}	1.01^{-06}
	Rel. reg. weight	7.34^{-13}	9.25^{-17}	4.10^{-15}	4.13^{-14}	1.12^{-14}	9.90^{-18}
	Ent. emb. dropout	0.43	0.02	0.42	0.50	0.00	0.00
	Rel. emb. dropout	0.16	0.00	0.41	0.23	0.12	0.00
	Proj. drop. (ConvE)	–	–	–	–	0.50	–
	Ft. map drop. (ConvE)	–	–	–	–	0.49	–
WNRR	Learning rate	0.00085	0.10666	0.33127	0.52558	0.00162	0.20986
	Sched. patience	8	6	7	5	1	6
	Ent. reg. weight	1.37^{-14}	1.51^{-07}	1.25^{-12}	4.52^{-06}	1.08^{-08}	1.01^{-06}
	Rel. reg. weight	2.57^{-15}	2.92^{-18}	1.53^{-14}	4.19^{-10}	9.52^{-11}	1.57^{-17}
	Ent. emb. dropout	0.00	0.00	0.37	0.36	0.00	0.00
	Rel. emb. dropout	0.24	0.00	0.50	0.31	0.23	0.00
	Proj. drop. (ConvE)	–	–	–	–	0.15	–
	Ft. map drop. (ConvE)	–	–	–	–	0.33	–

Table A.1: Hyperparameters of best performing models found after quasi-random hyperparameter optimization and Bayesian optimization. All other hyperparameters are the same as in Tables A.3 (FB15K-237) and A.4 (WNRR).

		RESCAL	TransE	DistMult	ComplEx	ConvE	RotatE
FB15K	Learning rate	0.07879	0.03905	0.00054	0.04934	0.11317	0.38031
	Sched. patience	7	5	6	8	3	4
	Ent. reg. weight	2.32^{-06}	9.60^{-11}	2.34^{-13}	5.99^{-06}	5.50^{-18}	4.57^{-09}
	Rel. reg. weight	2.42^{-03}	9.53^{-18}	7.85^{-08}	6.83^{-17}	2.41^{-06}	1.53^{-15}
	Ent. emb. dropout	0.30	0.02	0.00	0.15	0.16	0.00
	Rel. emb. dropout	0.21	0.00	0.11	0.00	0.44	0.00
	Proj. drop. (ConvE)	–	–	–	–	0.07	–
	Ft. map drop. (ConvE)	–	–	–	–	0.16	–
WN18	Learning rate	0.16402	0.00300	0.01491	0.09452	0.00100	0.46028
	Sched. patience	6	2	6	3	6	4
	Ent. reg. weight	5.12^{-09}	5.98^{-19}	7.96^{-16}	1.70^{-02}	4.81^{-16}	1.07^{-08}
	Rel. reg. weight	2.53^{-08}	7.62^{-17}	7.38^{-19}	3.72^{-08}	6.32^{-14}	5.95^{-16}
	Ent. emb. dropout	0.47	0.07	0.00	0.09	0.00	0.00
	Rel. emb. dropout	0.39	0.00	0.05	0.00	0.00	0.00
	Proj. drop. (ConvE)	–	–	–	–	0.18	–
	Ft. map drop. (ConvE)	–	–	–	–	0.47	–

Table A.2: Hyperparameters of best performing models found after quasi-random hyperparameter optimization and Bayesian optimization. All other hyperparameters are the same as in Tables A.5 (FB15K) and A.6 (WN18).

		RESCAL	TransE	DistMult	ComplEx	ConvE	RotatE
FB15K-237	Mean MRR	*36.1*	*33.5*	*35.0*	*35.3*	*34.3*	*34.0*
	Embedding size	128 *(-0.5)*	512 *(-2.8)*	256 *(-0.2)*	256 *(-0.3)*	256 *(-0.4)*	256 *(-2.0)*
	Training type	1vsAll *(-0.8)*	NegSamp –	NegSamp *(-0.2)*	NegSamp *(-0.3)*	1vsAll *(-0.4)*	NegSamp –
	Reciprocal	No *(-0.5)*	Yes *(-3.0)*	Yes *(-0.3)*	Yes *(-0.3)*	Yes –	Yes *(-2.0)*
	Samples sub (NegSamp)	–	2	557	557	–	25
	Samples obj (NegSamp)	–	56	367	367	–	292
	Lab. Smooth. (KvsAll)	–	–	–	–	–	–
	Loss	CE *(-0.9)*	CE *(-2.8)*	CE *(-3.1)*	CE *(-3.8)*	CE *(-0.4)*	BCE *(-2.3)*
	Margin (MR)	–	–	–	–	–	–
	L_p-norm (TransE)	–	L2	–	–	–	–
	Optimizer	Adam *(-0.5)*	Adagrad *(-2.8)*	Adagrad *(-0.2)*	Adagrad *(-0.5)*	Adagrad *(-1.5)*	Adagrad *(-2.7)*
	Batch size	512 *(-0.5)*	128 *(-2.8)*	1024 *(-0.2)*	1024 *(-0.3)*	1024 *(-0.4)*	128 *(-2.3)*
	Learning rate	0.00063	0.04122	0.14118	0.14118	0.00373	0.22032
	Sched. patience	1 *(-0.5)*	6 *(-2.8)*	9 *(-0.2)*	9 *(-0.3)*	5 *(-0.4)*	6 *(-2.0)*
	L_p reg.	L2 *(-0.5)*	L2 *(-2.8)*	L3 *(-0.2)*	L3 *(-0.3)*	L3 *(-0.4)*	L2 *(-2.5)*
	Ent. emb. weight	5.82^{-12}	1.32^{-07}	1.55^{-10}	1.55^{-10}	1.55^{-11}	1.01^{-06}
	Rel. emb. weight	6.77^{-11}	3.72^{-18}	3.93^{-15}	3.93^{-15}	7.91^{-12}	9.90^{-18}
	Freq. weight	Yes *(-0.5)*	No *(-2.8)*	Yes *(-0.3)*	Yes *(-0.3)*	Yes *(-1.5)*	No *(-2.0)*
	Emb. norm. (TransE)						
	Entity	–	No	–	–	–	–
	Relation	–	No	–	–	–	–
	Dropout						
	Ent. emb.	0.37	0.00	0.46	0.46	0.00	0.00
	Rel. emb.	0.28	0.00	0.36	0.36	0.10	0.00
	Proj. (ConvE)	–	–	–	–	0.19	–
	Feat. map (ConvE)	–	–	–	–	0.49	–
	Emb. init.	Normal *(-0.8)*	XvNorm *(-2.8)*	Unif. *(-0.2)*	Unif. *(-0.5)*	XvNorm *(-0.4)*	XvNorm *(-2.0)*
	Std. dev. (Normal)	0.80620	–	–	–	–	–
	Interval (Unif)	–	–	[-0.85, 0.85]	[-0.85, 0.85]	–	–

Table A.3: Hyperparameters of best models found with random search on FB15K-237. For each hyperparameter, we show in parenthesis the reduction in MRR for the best configuration that does not use this value.

		RESCAL	TransE	DistMult	ComplEx	ConvE	RotatE
WNRR	Mean MRR	*46.8*	*23.0*	*45.4*	*47.6*	*44.3*	*47.7*
	Embedding size	128 *(-1.0)*	512 *(-0.9)*	512 *(-1.1)*	128 *(-1.0)*	512 *(-1.2)*	256 *(-3.7)*
	Training type	KvsAll *(-1.0)*	NegSamp –	KvsAll *(-1.1)*	1vsAll *(-1.0)*	KvsAll *(-1.2)*	NegSamp –
	Reciprocal	Yes *(-1.0)*	Yes *(-2.1)*	Yes *(-1.1)*	No *(-1.0)*	Yes –	Yes *(-1.9)*
	Samples sub (NegSamp)	–	2	–	–	–	25
	Samples obj (NegSamp)	–	56	–	–	–	292
	Lab. Smooth. (KvsAll)	0.30	–	0.21	–	-0.29	–
	Loss	CE *(-2.0)*	CE *(-3.4)*	CE *(-2.4)*	CE *(-3.5)*	CE *(-1.4)*	BCE *(-2.4)*
	Margin (MR)	–	–	–	–	–	–
	L_p-norm (TransE)	–	L2	–	–	–	–
	Optimizer	Adam *(-1.2)*	Adag. *(-2.6)*	Adag. *(-1.5)*	Adag. *(-1.5)*	Adam *(-1.4)*	Adag. *(-1.9)*
	Batch size	128 *(-1.0)*	128 *(-0.9)*	1024 *(-1.1)*	512 *(-1.0)*	1024 *(-1.3)*	128 *(-2.4)*
	Learning rate	0.00160	0.04122	0.25575	0.50338	0.00160	0.22032
	Sched. patience	8 *(-1.0)*	6 *(-0.9)*	6 *(-1.1)*	7 *(-1.0)*	1 *(-1.2)*	6 *(-1.9)*
	L_p reg.	L3 *(-1.2)*	L2 *(-0.9)*	L3 *(-1.1)*	L2 *(-1.0)*	L1 *(-1.2)*	L2 *(-1.9)*
	Ent. emb. weight	3.82^{-20}	1.32^{-07}	1.34^{-10}	1.48^{-18}	3.70^{-12}	1.01^{-06}
	Rel. emb. weight	7.47^{-05}	3.72^{-18}	6.38^{-16}	1.44^{-18}	6.56^{-10}	9.90^{-18}
	Freq. weigh	No *(-1.0)*	No *(-0.9)*	Yes *(-1.1)*	No *(-1.0)*	No *(-1.4)*	No *(-1.9)*
	Emb. norm. (TransE)						
	Ent.	–	No	–	–	–	–
	Rel.	–	No	–	–	–	–
	Dropout						
	Ent. emb.	0.00	0.00	0.12	0.05	0.00	0.00
	Rel. emb.	0.00	0.00	0.36	0.44	0.23	0.00
	Proj. (ConvE)	–	–	–	–	0.09	–
	Feat. map (ConvE)	–	–	–	–	0.42	–
	Emb. init.	Unif. *(-1.0)*	XvNorm *(-2.1)*	Unif. *(-1.3)*	Unif. *(-1.5)*	XvNorm *(-1.4)*	XvNorm *(-1.9)*
	Std. dev. (Normal)	–	–	–	–	–	–
	Interval (Unif)	[-0.31, 0.31]	–	[-0.81, 0.81]	[-0.31, 0.31]	–	–

Table A.4: Hyperparameters of best models found with random search on WNRR. For each hyperparameter, we show in parenthesis the reduction in MRR for the best configuration that does not use this value.

		RESCAL	TransE	DistMult	ComplEx	ConvE	RotatE
FB15K	Mean MRR	*64.6*	*68.2*	*84.0*	*83.8*	*82.1*	*78.2*
	Embedding size	256 *(-3.4)*	512 *(-18.1)*	512 *(-1.2)*	256 *(0.0)*	512 *(-5.6)*	256 *(-15.7)*
	Training type	1vsAll *(-8.3)*	NegSamp –	1vsAll *(-5.9)*	1vsAll *(-7.2)*	KvsAll *(-0.2)*	NegSamp –
	Reciprocal	No *(-3.4)*	Yes *(-18.1)*	Yes *(-1.2)*	No *(0.0)*	Yes –	Yes *(-15.7)*
	Samples sub (NegSamp)	–	11	–	–	–	49
	Samples obj (NegSamp)	–	3	–	–	–	361
	Lab. Smooth. (KvsAll)	–	–	–	–	0.25	–
	Loss	CE *(-7.3)*	BCE *(-15.4)*	CE *(-3.1)*	CE *(-1.7)*	BCE *(-0.2)*	CE *(-8.7)*
	Margin (MR)	–	–	–	–	–	–
	L_p-norm (TransE)	–	L1	–	–	–	–
	Optimizer	Adagrad *(-4.0)*	Adagrad *(-23.3)*	Adam *(-1.2)*	Adagrad *(0.0)*	Adagrad *(-0.2)*	Adagrad *(-15.7)*
	Batch size	512 *(-3.7)*	256 *(-15.4)*	1024 *(-1.2)*	256 *(0.0)*	512 *(-0.4)*	512 *(-8.7)*
	Learning rate	0.01958	0.00826	0.00034	0.06949	0.17384	0.46028
	Sched. patience	4 *(-3.4)*	9 *(-15.4)*	7 *(-1.2)*	8 *(0.0)*	4 *(-0.2)*	4 *(-8.7)*
	L_p reg.	L2 *(-4.1)*	L1 *(-15.4)*	L2 *(-2.0)*	L2 *(-1.4)*	L2 *(-0.2)*	L3 *(-15.7)*
	Ent. emb. weight	8.06^{-11}	1.36^{-11}	9.58^{-13}	9.56^{-07}	5.14^{-20}	1.07^{-08}
	Rel. emb. weight	7.19^{-18}	9.91^{-20}	2.29^{-02}	2.56^{-17}	2.49^{-06}	5.95^{-16}
	Freq. weight	Yes *(-3.4)*	Yes *(-15.4)*	Yes *(-1.2)*	No *(0.0)*	Yes *(-0.2)*	No *(-15.7)*
	Emb. norm. (TransE)						
	Ent.	–	L2	–	–	–	–
	Rel.	–	L2	–	–	–	–
	Dropout						
	Ent. emb.	0.16	0.03	0.08	0.00	0.16	0.00
	Rel. emb.	0.09	0.00	0.06	0.00	0.46	0.00
	Proj. (ConvE)	–	–	–	–	0.03	–
	Feat. map (ConvE)	–	–	–	–	0.16	–
	Emb. init.	Normal *(-3.4)*	Normal *(-15.4)*	XvNorm *(-1.2)*	XvUnif *(0.0)*	XvNorm *(-0.4)*	Normal *(-8.7)*
	Std. dev. (Normal)	0.00020	0.00003	–	–	–	0.12247
	Interval (Unif)	–	–	–	–	–	–

Table A.5: Hyperparameters of best models found with random search on FB15K. For each hyperparameter, we show in parenthesis the reduction in MRR for the best configuration that does not use this value.

		RESCAL	TransE	DistMult	ComplEx	ConvE	RotatE
WN18	Mean MRR	*94.7*	*60.9*	*93.9*	*95.0*	*94.6*	*94.6*
	Embedding size	256 *(-0.3)*	256 *(-1.4)*	512 *(0.1)*	512 *(-0.1)*	128 *(-0.0)*	256 *(-0.8)*
	Training type	1vsAll *(-0.3)*	NegSamp –	1vsAll *(-4.4)*	KvsAll *(-0.1)*	1vsAll *(-0.1)*	NegSamp –
	Reciprocal	Yes *(-0.3)*	Yes *(-1.4)*	Yes *(-9.8)*	Yes *(-0.1)*	Yes –	Yes *(-0.8)*
	Samples sub (NegSamp)	–	87	–	–	–	49
	Samples obj (NegSamp)	–	26	–	–	–	361
	Lab. Smooth. (KvsAll)	–	–	–	0.21	–	–
	Loss	CE *(-1.5)*	CE *(-8.0)*	CE *(-9.8)*	CE *(-0.5)*	CE *(-0.1)*	CE *(-0.5)*
	Margin (MR)	–	–	–	–	–	–
	L_p-norm (TransE)	–	L1	–	–	–	–
	Optimizer	Adagrad *(-0.3)*	Adagrad *(-1.4)*	Adagrad *(-1.1)*	Adagrad *(-0.2)*	Adam *(-0.1)*	Adagrad *(-0.8)*
	Batch size	256 *(-0.3)*	512 *(-1.4)*	128 *(0.1)*	1024 *(-0.1)*	1024 *(-0.0)*	512 *(-0.5)*
	Learning rate	0.06949	0.00299	0.01068	0.25575	0.00089	0.46028
	Sched. patience	8 *(-0.3)*	2 *(-1.4)*	6 *(0.1)*	6 *(-0.1)*	4 *(-0.0)*	4 *(-0.5)*
	L_p reg.	L2 *(-0.3)*	L3 *(-1.4)*	L3 *(0.1)*	L3 *(-0.1)*	L2 *(-0.0)*	L3 *(-0.7)*
	Ent. emb. weight	9.56^{-07}	6.67^{-20}	4.08^{-17}	1.34^{-10}	8.63^{-17}	1.07^{-08}
	Rel. emb. weight	2.56^{-17}	1.68^{-15}	8.78^{-16}	6.38^{-16}	2.51^{-14}	5.95^{-16}
	Freq. weight	No *(-0.3)*	Yes *(-8.0)*	No *(0.1)*	Yes *(-0.1)*	No *(-0.1)*	No *(-0.8)*
	Emb. norm. (TransE)						
	Ent.	–	No	–	–	–	–
	Rel.	–	L2	–	–	–	–
	Dropout						
	Ent. emb.	0.00	0.16	0.00	0.12	0.00	0.00
	Rel. emb.	0.00	0.00	0.16	0.36	0.00	0.00
	Proj. (ConvE)	–	–	–	–	0.09	–
	Feat. map (ConvE)	–	–	–	–	0.46	–
	Emb. init.	XvUnif *(-0.3)*	XvNorm *(-2.1)*	Normal *(-0.4)*	Unif. *(-0.1)*	XvNorm *(-0.1)*	Normal *(-0.5)*
	Std. dev. (Normal)	–	–	0.00322	–	–	0.12247
	Interval (Unif)	–	–	–	[-0.81, 0.81]	–	–

Table A.6: Hyperparameters of best models found with random search on WN18. For each hyperparameter, we show in parenthesis the reduction in MRR for the best configuration that does not use this value.

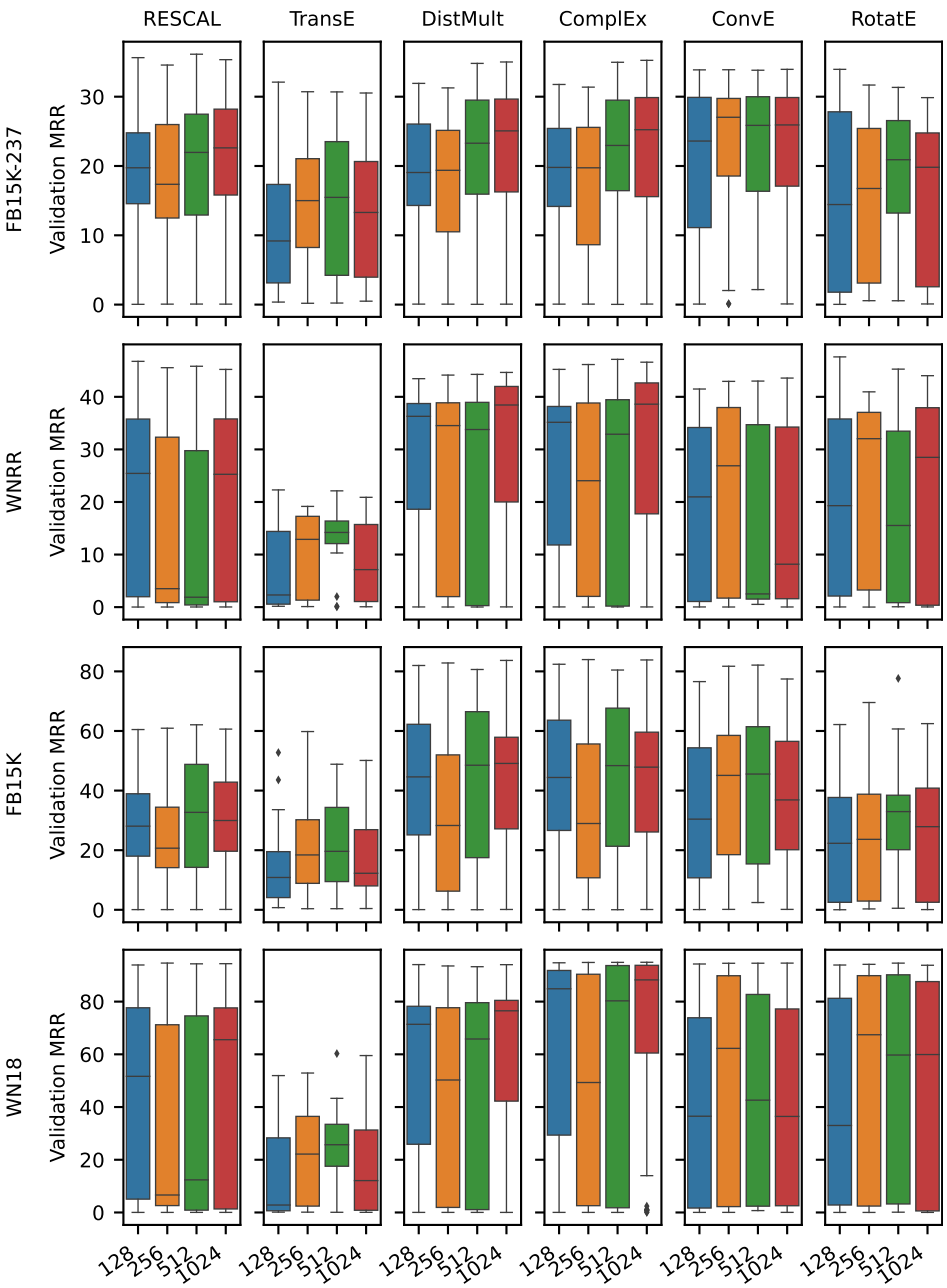


Figure A.1: Distribution of filtered MRR (on validation data, quasi-random search only) for different batch sizes (top row: FB15K-237, bottom row: WNRR)

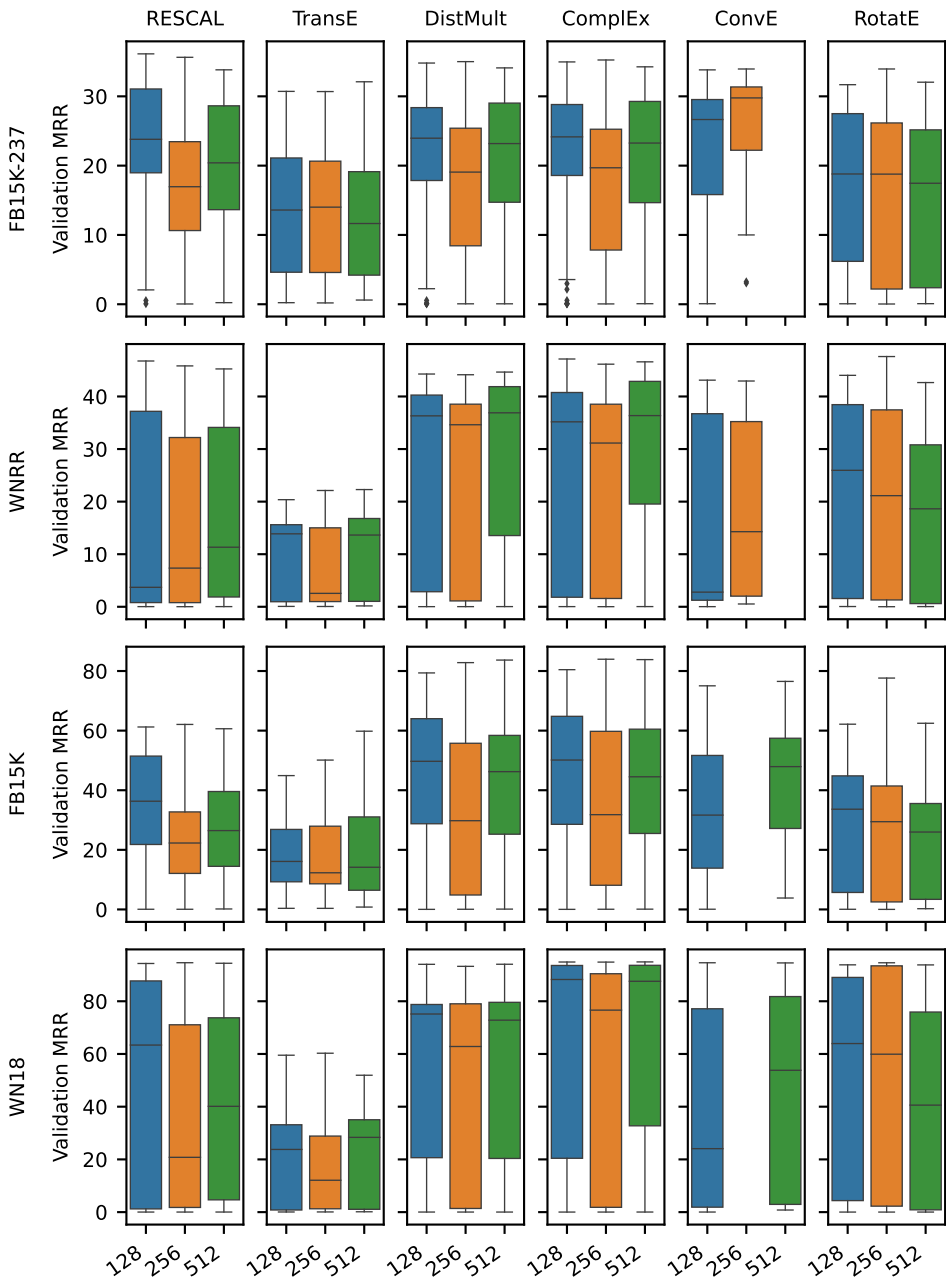


Figure A.2: Distribution of filtered MRR (on validation data, quasi-random search only) for different embedding sizes (top row: FB15K-237, bottom row: WNRR)

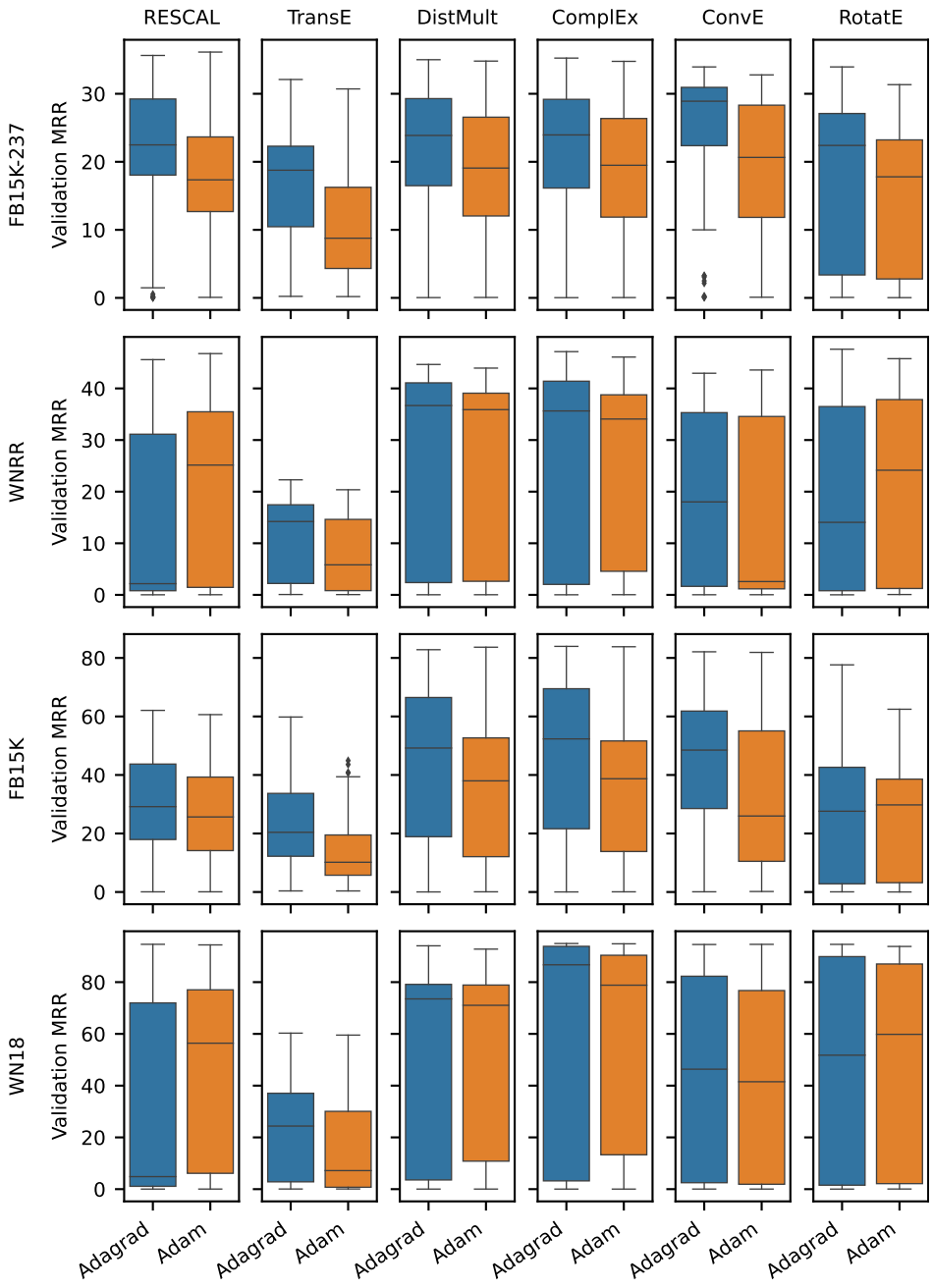


Figure A.3: Distribution of filtered MRR (on validation data, quasi-random search only) for different optimizers (top row: FB15K-237, bottom row: WNRR)

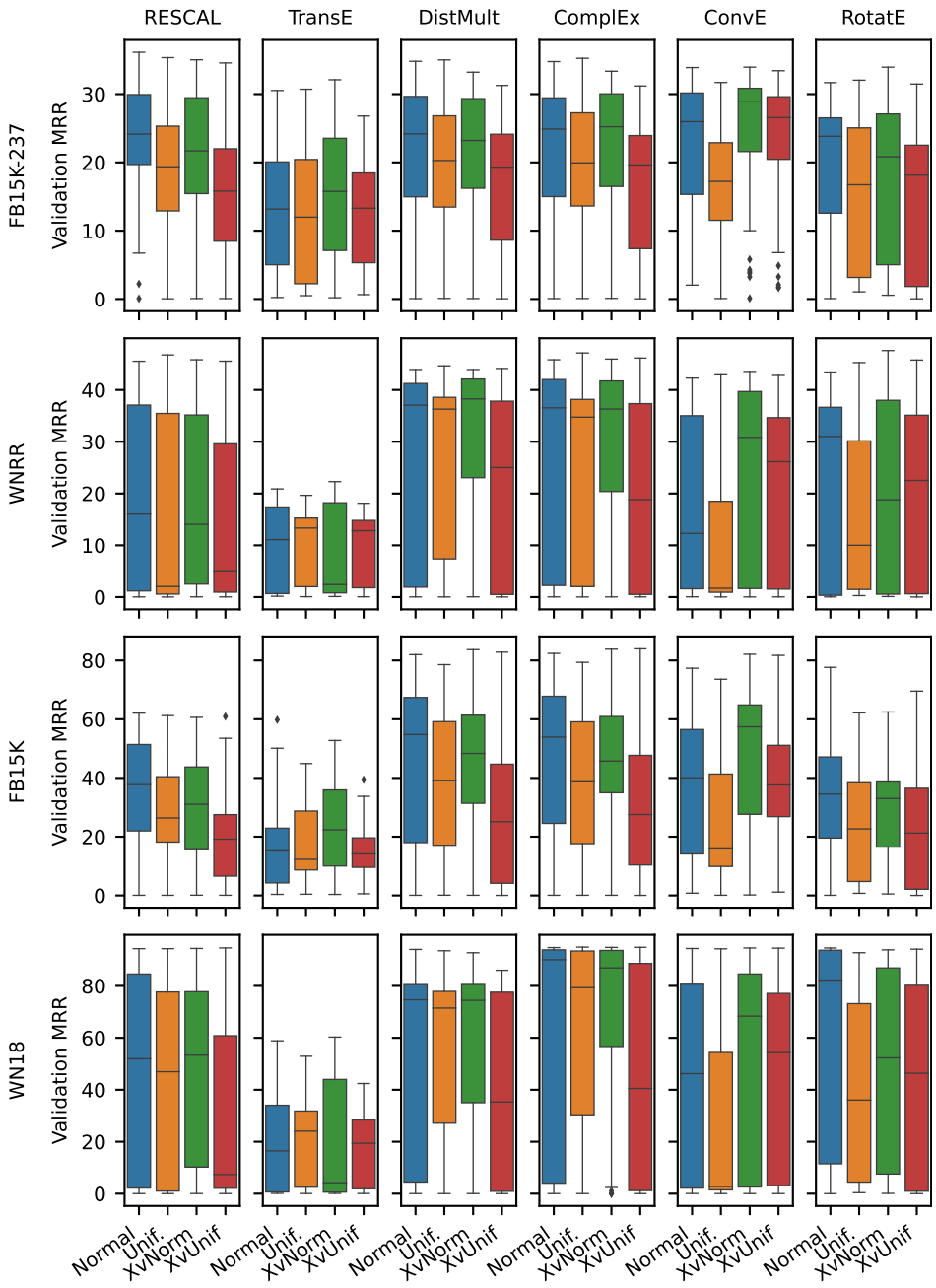


Figure A.4: Distribution of filtered MRR (on validation data, quasi-random search only) for different initializers (top row: FB15K-237, bottom row: WNRR)

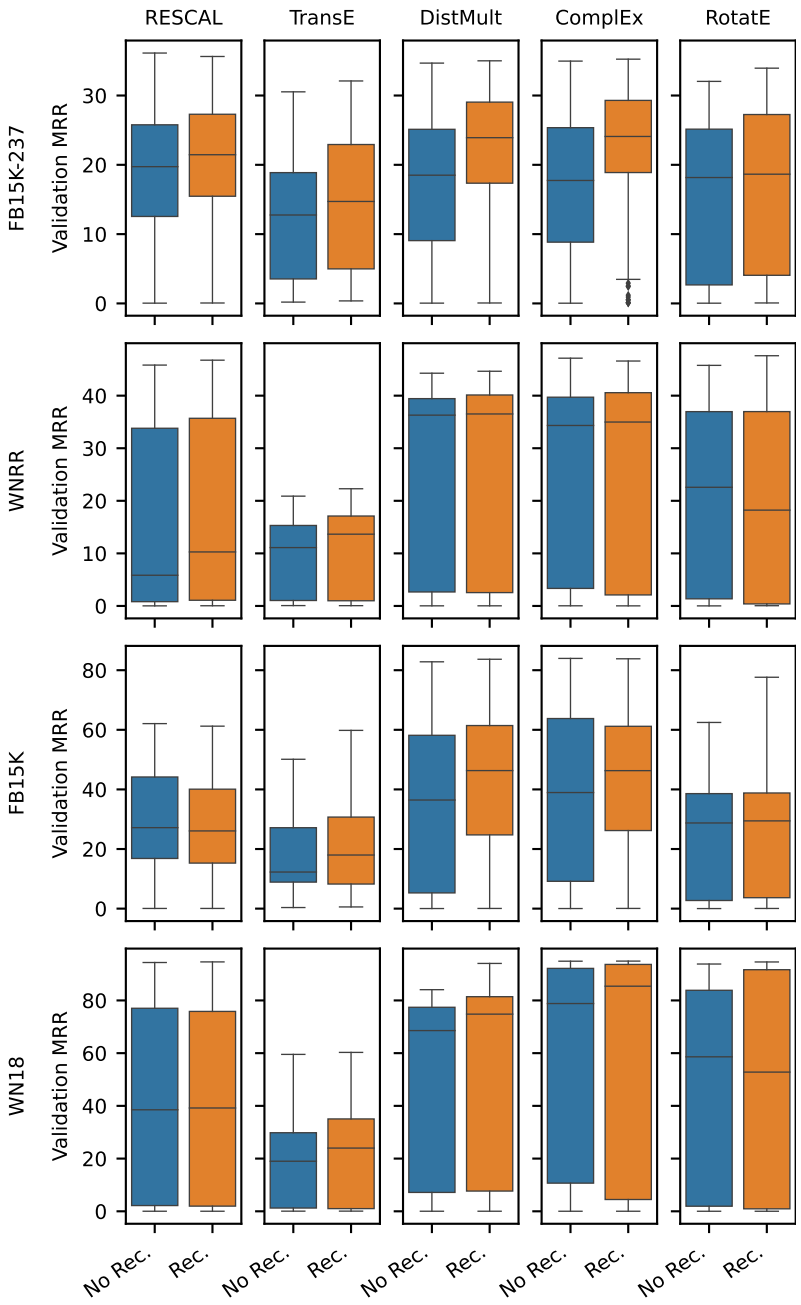


Figure A.5: Distribution of filtered MRR (on validation data, quasi-random search only) with and without reciprocal relations (top row: FB15K-237, bottom row: WNRR)

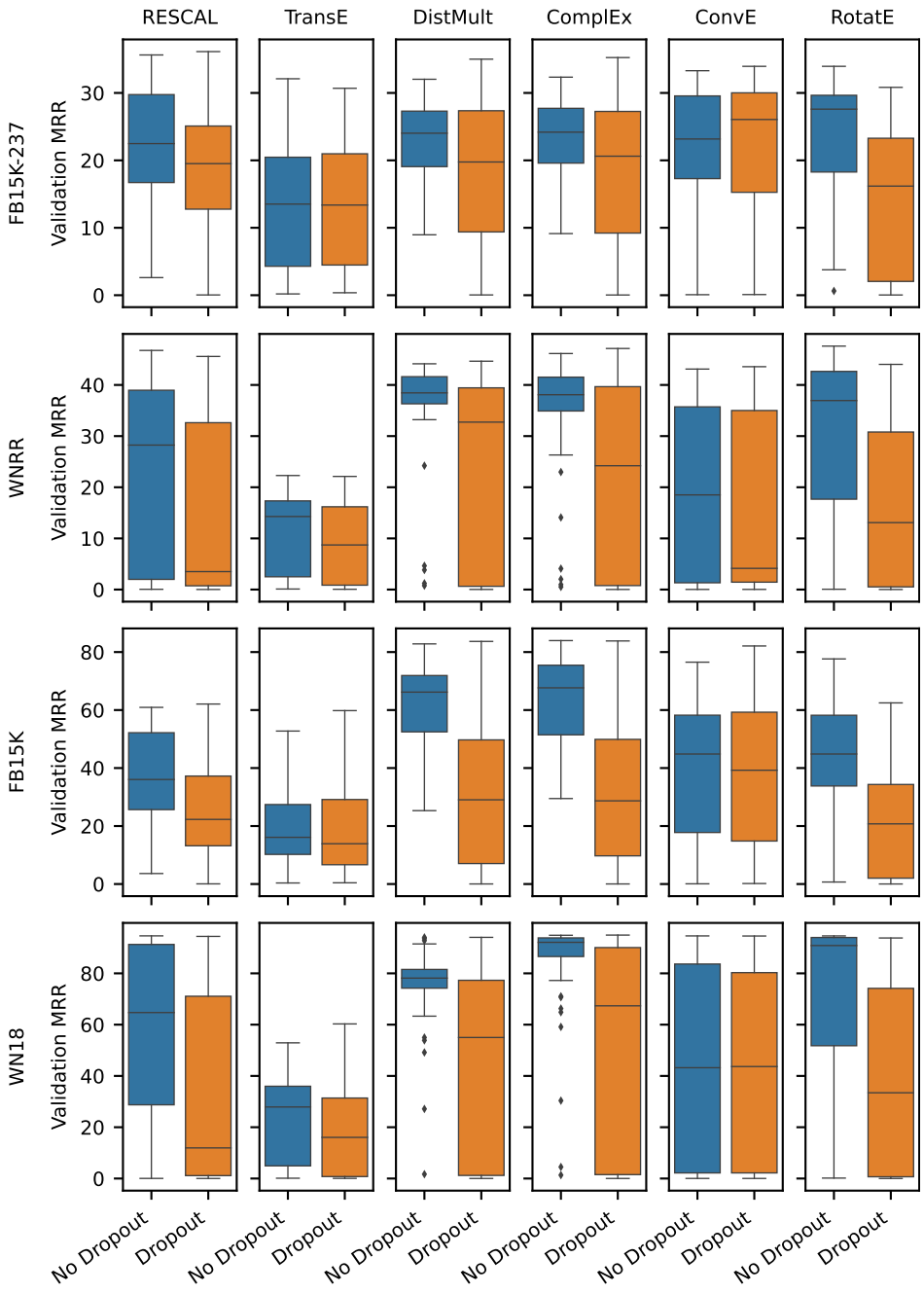


Figure A.6: Distribution of filtered MRR (on validation data, quasi-random search only) with and without dropout (top row: FB15K-237, bottom row: WNRR)

B Additional Material for Chapter 4

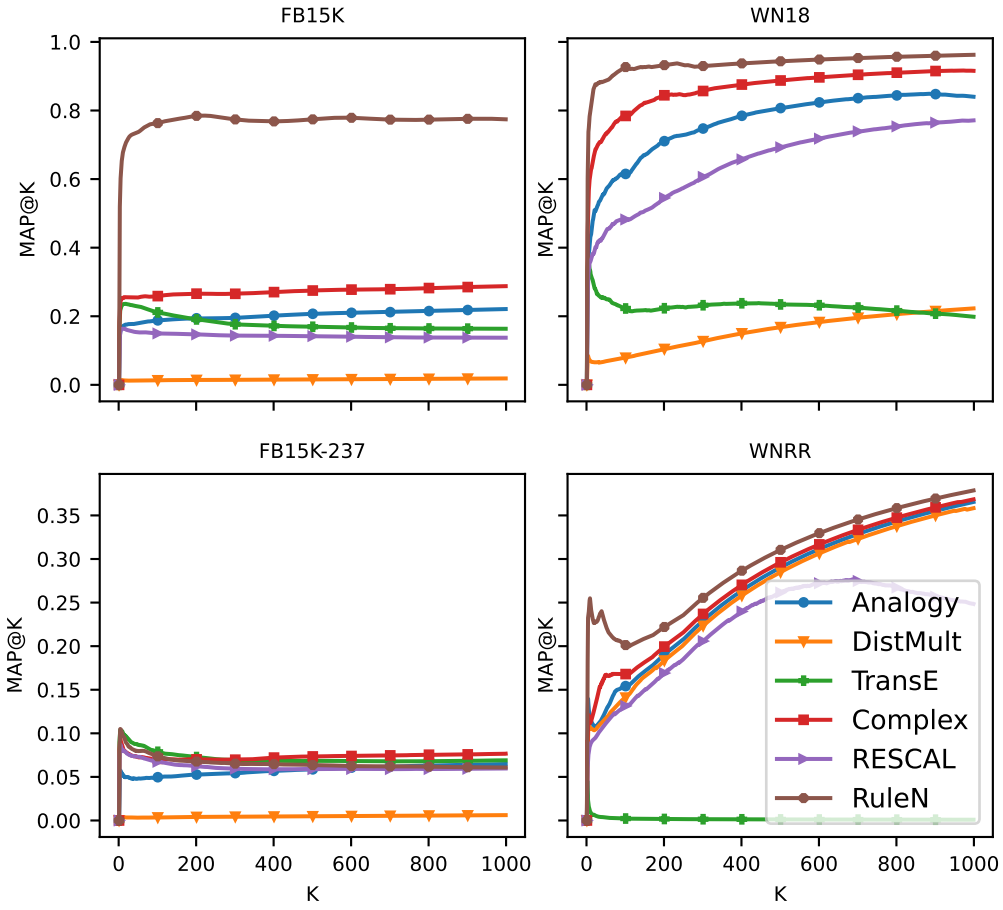


Figure B.1: MAP@K with PR as a function of K.

C Additional Material for Chapter 5

	Training	*Selection*	*Graph-structure prediction (↑)*					
			LP	REL	DOM	NBE	NBR	MTR
ComplEx	STD	LP	**.346**	.805	.423	.016	.046	.274
	MTT	MTR	.331	**.977**	.773	**.210**	.925	**.606**
	w/o LP	MTR	.154	.972	.831	.200	.932	.579
	w/o REL	MTR	.322	.831	.831	.159	.927	.590
	w/o DOM	MTR	.327	.966	.713	.198	.915	.586
	w/o NBE	MTR	.315	.958	**.850**	.005	.936	.575
	w/o NBR	MTR	.325	.967	.795	.199	.874	.595
	w/o LP+REL	MTR	.001	.009	.843	.177	**.939**	.436
	w/o DOM+NBR	MTR	.330	.970	.074	.199	.107	.266
DistMult	STD	LP	.342	.388	.045	.009	.036	.139
	MTT	MTR	.327	.939	.780	.142	.879	.577
	w/o LP	MTR	.159	.954	.826	.087	.937	.553
	w/o REL	MTR	.323	.857	.827	.057	.932	.571
	w/o DOM	MTR	.323	.948	.703	.106	.914	.560
	w/o NBE	MTR	.316	.928	.848	.003	.937	.571
	w/o NBR	MTR	.325	.956	.801	.112	.775	.554
	w/o LP+REL	MTR	.000	.019	.837	.108	.937	.420
	w/o DOM+NBR	MTR	.307	.955	.136	.147	.279	.299
RotatE	STD	LP	.312	.919	.581	.051	.136	.342
	MTT	MTR	.314	.964	.813	.160	.922	.598
	w/o LP	MTR	.204	.914	.842	.126	.928	.568
	w/o REL	MTR	.272	.887	.846	.137	.924	.583
	w/o DOM	MTR	.319	.965	.661	.170	.883	.559
	w/o NBE	MTR	.301	.960	.813	.003	.912	.558
	w/o NBR	MTR	.318	.964	.710	.168	.673	.522
	w/o LP+REL	MTR	.012	.031	.842	.124	.916	.424
	w/o DOM+NBR	MTR	.322	.945	.016	.166	.019	.221
TransE	STD	LP	.330	.900	.624	.038	.054	.332
	MTT	MTR	.288	.960	.708	.112	.911	.555
	w/o LP	MTR	.271	.968	.781	.138	.901	.572
	w/o REL	MTR	.307	.944	.698	.124	.906	.557
	w/o DOM	MTR	.325	.965	.626	.126	.879	.542
	w/o NBE	MTR	.330	.966	.801	.012	.904	.562
	w/o NBR	MTR	.329	.966	.723	.125	.790	.545
	w/o LP+REL	MTR	.149	.930	.821	.116	.924	.550
	w/o DOM+NBR	MTR	.312	.962	.360	.129	.580	.414

Table C.1: Performance on test data of graph-structure prediction and downstream tasks for FB-237 of STD with LP model selection and various forms of multi-task training, all using MTR for model selection.

D Additional Material for Chapter 6

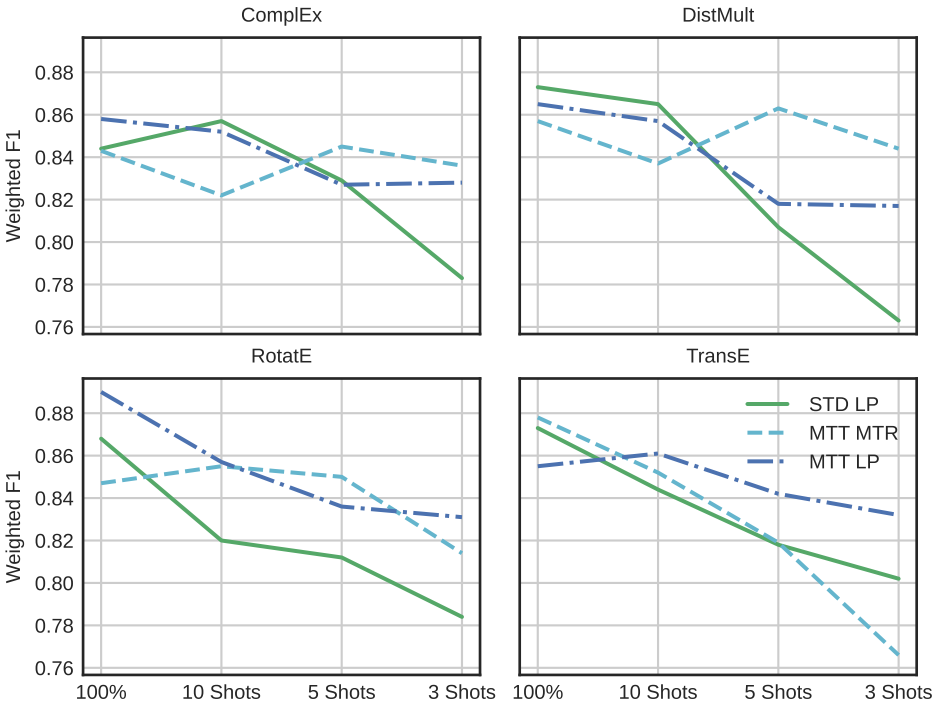


Figure D.1: Few-shot performance of entity classification tasks for FB15K-237 (higher is better). Each n-shot training set consists of n sampled positive and negative examples for each class.

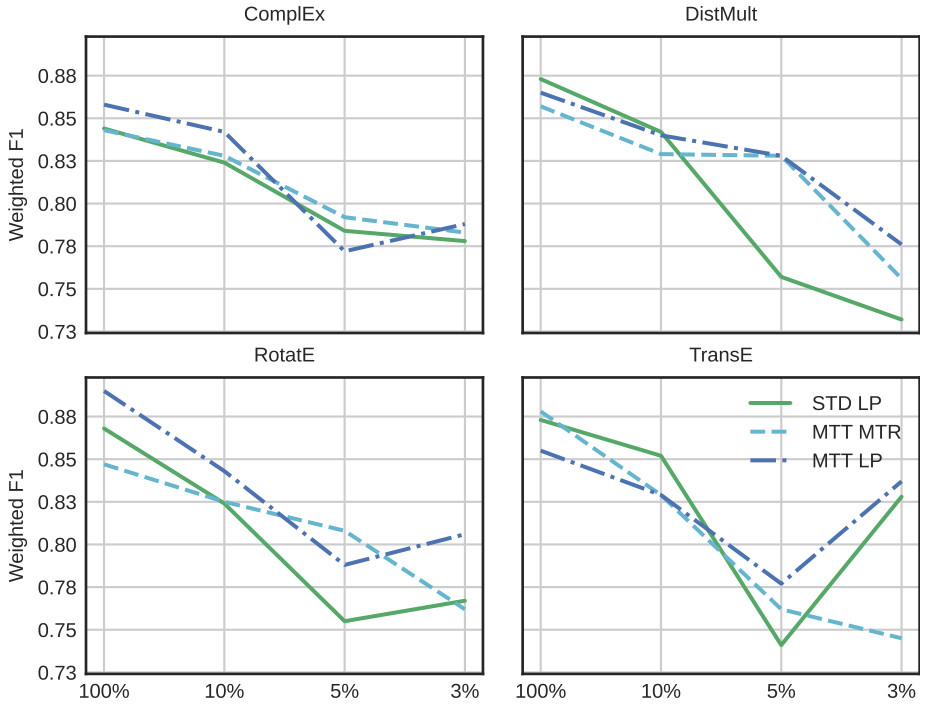


Figure D.2: Performance on entity classification for FB15K-237 with down-sampled training sets (higher is better). Each training set was constructed by sampling (stratified) a percentage of the training set.

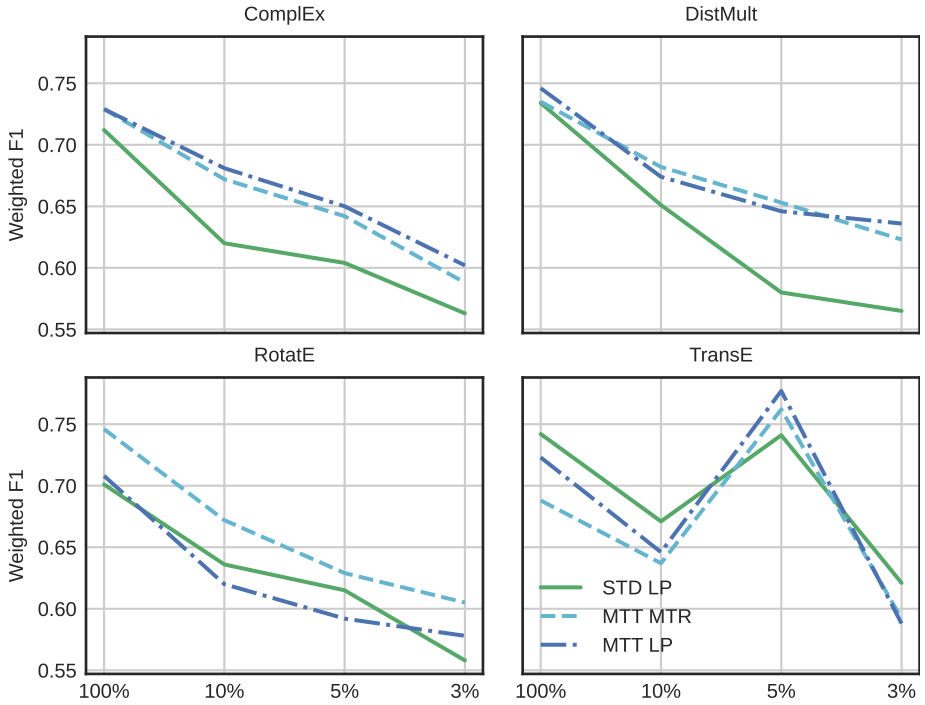


Figure D.3: Performance on entity classification for YAGO3-10 with down-sampled training sets (higher is better). Each training set was constructed by sampling (stratified) a percentage of the training set.

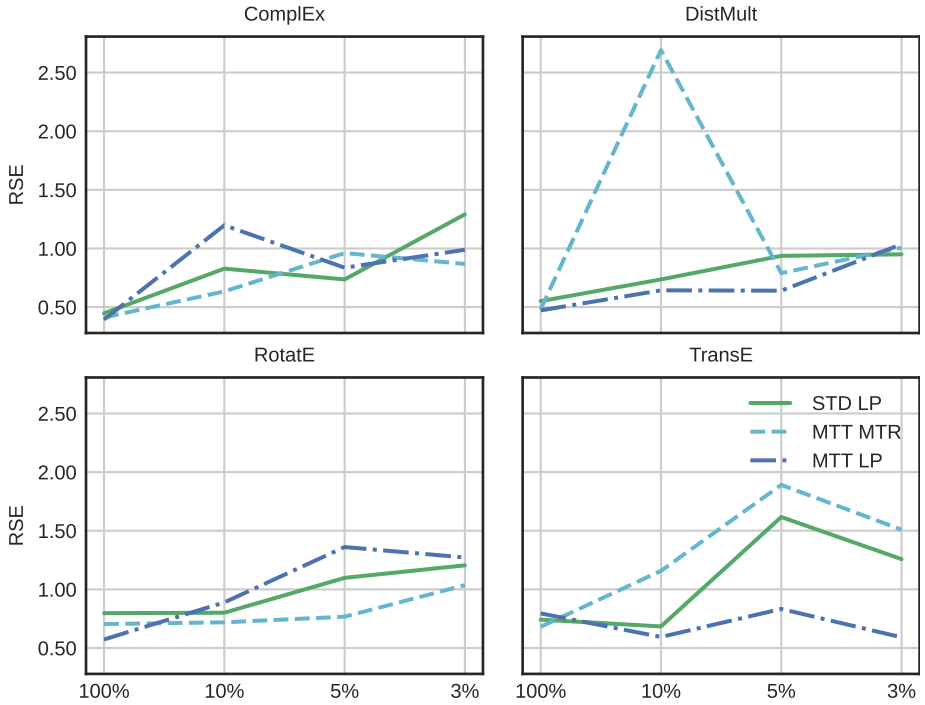


Figure D.4: Performance of regression tasks for FB15K-237 with downsampled training sets (lower is better). Each training set was constructed by sampling a percentage of the training set.

		FB15K-237			
		Entity Classification (Weighted F1 - higher is better)			
		Type	Profession	Organization	Writer
ComplEx	STD+LP	.986±.001	.808±.011	.921±.021	.661±.000
	MTT+LP	.986±.000	.820±.005	.946±.003	.682±.012
	MTT+MTR	.986±.000	.802±.004	.944±.003	.641±.000
DistMult	STD+LP	.984±.000	.811±.007	.912±.009	.785±.020
	MTT+LP	.987±.000	.810±.016	.974±.002	.690±.000
	MTT+MTR	.986±.000	.785±.006	.890±.000	.768±.018
RotatE	STD+LP	.985±.000	.797±.000	.908±.013	.781±.000
	MTT+LP	.989±.001	.807±.000	.934±.012	.828±.000
	MTT+MTR	.989±.000	.810±.000	.931±.003	.658±.000
TransE	STD+LP	.984±.001	.791±.005	.913±.032	.806±.021
	MTT+LP	.987±.000	.805±.006	.946±.009	.681±.014
	MTT+MTR	.987±.000	.796±.000	.942±.000	.789±.034
KE-GCN		.988±.000	.738±.000	.906±.002	.685±.020

Table D.1: Weighted F1 on test data of downstream classifiers (MLP, Logistic Regression, KNN and Random Forest) that use pre-trained KGE embeddings as input to solve entity classification tasks about entities in FB15K-237; and KE-GCN (Yu et al., 2021a), a GCN that trains directly on the downstream data. Datasets are sorted by decreasing size of the training set from left to right.

		YAGO3-10							
		Entity Classification (Weighted F1 - higher is better)							
		Type	Player	Profession	Writer	Scientist	Organization	Artist	Waterbody
ComplEx	STD+LP	.994±.000	.918±.001	.753±.004	.575±.006	.518±.013	.789±.005	.480±.018	.673±.015
	MTT+LP	.997±.000	.919±.002	.790±.002	.619±.006	.553±.011	.877±.003	.466±.013	.614±.000
	MTT+MTR	.996±.000	.914±.001	.776±.000	.617±.009	.556±.007	.871±.005	.491±.021	.614±.000
DistMult	STD+LP	.994±.000	.919±.001	.764±.003	.577±.000	.529±.003	.814±.011	.535±.007	.738±.000
	MTT+LP	.996±.000	.919±.002	.789±.002	.634±.019	.556±.003	.890±.010	.495±.010	.691±.000
	MTT+MTR	.996±.000	.918±.002	.776±.000	.622±.006	.539±.009	.876±.005	.462±.006	.691±.000
RotatE	STD+LP	.973±.001	.914±.000	.706±.002	.611±.000	.545±.000	.734±.014	.530±.000	.593±.000
	MTT+LP	.990±.001	.913±.001	.733±.000	.605±.000	.469±.009	.793±.005	.413±.000	.751±.000
	MTT+MTR	.994±.000	.919±.001	.768±.000	.643±.000	.576±.000	.830±.011	.534±.000	.707±.000
TransE	STD+LP	.993±.000	.919±.001	.762±.000	.623±.000	.630±.000	.833±.000	.507±.015	.670±.000
	MTT+LP	.991±.000	.912±.000	.728±.005	.583±.000	.603±.000	.804±.011	.506±.007	.654±.006
	MTT+MTR	.992±.000	.892±.000	.750±.000	.580±.000	.401±.012	.809±.003	.464±.015	.614±.012
KE-GCN		.996±.000	.896±.001	.709±.000	.582±.005	.610±.006	.853±.006	.463±.014	.488±.014

Table D.2: Weighted F1 on test data of downstream classifiers (MLP, Logistic Regression, KNN and Random Forest) that use pre-trained KGE embeddings as input to solve entity classification tasks about entities in YAGO3-10; and KE-GCN (Yu et al., 2021a), a GCN that trains directly on the downstream data. Datasets are sorted by decreasing size of the training set from left to right.

		FB15K-237				
		Regression (RSE - lower is better)				
		Node Imp.	Birth Year	Latitude	Longitude	Person Height
ComplEx	STD+LP	.870±.048	.601±.239	.172±.013	.089±.010	.678±.010
	MTT+LP	.918±.142	.477±.190	.145±.015	.066±.008	.661±.011
	MTT+MTR	.909±.086	.214±.050	.143±.009	.096±.008	.678±.000
DistMult	STD+LP	.807±.023	.844±.042	.182±.031	.088±.005	.669±.003
	MTT+LP	.788±.006	.827±.065	.143±.001	.083±.013	.651±.009
	MTT+MTR	.802±.049	.701±.052	.232±.053	.070±.006	.691±.000
RotatE	STD+LP	.913±.000	.872±.027	.498±.057	.279±.003	.657±.000
	MTT+LP	.834±.016	.797±.069	.313±.014	.173±.003	.813±.136
	MTT+MTR	.856±.003	.811±.005	.411±.022	.225±.096	.847±.000
TransE	STD+LP	.886±.035	.836±.041	.170±.022	.084±.006	.722±.003
	MTT+LP	.833±.018	.812±.012	.078±.011	.061±.003	.769±.009
	MTT+MTR	.897±.044	.655±.053	.088±.005	.052±.006	.824±.000
KE-GCN		.804±.005	.376±.035	.218±.023	.113±.003	.748±.002

Table D.3: Part 1: Relative squared error (RSE) on test data of downstream models (MLP and Linear Regression) that use pre-trained KGE embeddings as input to solve regression tasks about entities in FB15K-237; and KE-GCN (Yu et al., 2021a), a GCN that trains directly on the downstream data. Models with RSE above 1 are considered unsatisfactory. Datasets are sorted by decreasing size of the training set from left to right.

		FB15K-237 *Regression (RSE - lower is better)*				
		Size Area	Population	Film Year	Date Founded	Film Rating
ComplEx	STD+LP	.234±.018	.442±.071	.156±.016	.494±.042	.736±.046
	MTT+LP	.046±.026	.260±.064	.138±.007	.431±.047	.795±.058
	MTT+MTR	.049±.021	.493±.097	.126±.003	.605±.033	.804±.065
DistMult	STD+LP	.412±.318	.914±.093	.152±.003	.627±.036	.813±.062
	MTT+LP	.435±.046	.503±.004	.134±.012	.429±.045	.728±.063
	MTT+MTR	.025±.008	.540±.030	.146±.005	.718±.012	.894±.043
RotatE	STD+LP	.700±.223	.463±.429	.176±.004	.618±.024	.792±.089
	MTT+LP	.708±.112	.537±.035	.146±.008	.514±.055	.897±.168
	MTT+MTR	.440±.190	.710±.158	.157±.010	.631±.060	.949±.056
TransE	STD+LP	.326±.075	.906±.574	.153±.019	.499±.046	.839±.045
	MTT+LP	.041±.744	.227±.730	.141±.004	.300±.013	.690±.031
	MTT+MTR	.833±.684	.675±.109	.130±.012	.708±.022	.946±.018
KE-GCN		.754±.0180	.664±.051	.144±.008	.498±.034	.691±.009

Table D.4: Part 2: Relative squared error (RSE) on test data of downstream models (MLP and Linear Regression) that use pre-trained KGE embeddings as input to solve regression tasks about entities in FB15K-237; and KE-GCN (Yu et al., 2021a), a GCN that trains directly on the downstream data. Models with RSE above 1 are considered unsatisfactory. Datasets are sorted by decreasing size of the training set from left to right.

		YAGO3-10 *Regression (RSE - lower is better)*				
		Born on Date	Created on Date	Died on Date	Destroyed on Date	Happened on Date
ComplEx	STD+LP	.519±.001	.672±.033	.555±.014	.872±.060	.324±.006
	MTT+LP	.345±.025	.603±.009	.377±.005	.709±.009	.296±.036
	MTT+MTR	.363±.010	.643±.016	.406±.023	.605±.029	.277±.023
DistMult	STD+LP	.432±.013	.612±.024	.466±.025	.773±.004	.311±.030
	MTT+LP	.345±.023	.565±.015	.416±.023	.724±.044	.312±.040
	MTT+MTR	.352±.006	.648±.016	.438±.035	.677±.024	.214±.022
RotatE	STD+LP	.689±.027	.800±.009	.849±.000	.913±.000	.227±.055
	MTT+LP	.538±.006	.717±.008	.657±.018	.886±.031	.497±.233
	MTT+MTR	.421±.016	.706±.012	.468±.003	.616±.043	.137±.013
TransE	STD+LP	.422±.018	.647±.008	.351±.037	.513±.057	.300±.059
	MTT+LP	.371±.006	.725±.022	.434±.009	.573±.081	.100±.027
	MTT+MTR	.494±.017	.777±.000	.521±.038	.942±.048	.666±.024
KE-GCN		.256±.009	.611±.008	.299±.011	.657±.045	.167±.001

Table D.5: Relative squared error (RSE) on test data of downstream models (MLP and Linear Regression) that use pre-trained KGE embeddings as input to solve regression tasks about entities in YAGO3-10; and KE-GCN (Yu et al., 2021a), a GCN that trains directly on the downstream data. Models with RSE above 1 are considered unsatisfactory. Datasets are sorted by decreasing size of the training set from left to right.

		WIKIDATA5M			
		Regression (RSE - lower is better)			
		Date of Birth	Album Pub.	Asteroid Mag.	River Length
ComplEx	STD+LP	.475±.003	.760±.009	.436±.014	.559±.022
	MTT+LP	.481±.006	.844±.009	.519±.026	.540±.007
	MTT+MTR	.468±.010	.813±.006	.518±.014	.659±.025
TransE	STD+LP	.373±.002	.555±.004	.377±.015	.444±.016
	MTT+LP	.434±.007	.669±.003	.439±.013	.433±.029
	MTT+MTR	.455±.005	.667±.010	.455±.021	.418±.021

Table D.6: Part 1: Relative squared error (RSE) on test data of downstream models (MLP and Linear Regression) that use pre-trained KGE embeddings as input to solve regression tasks about entities in WIKIDATA5M; Models with RSE above 1 are considered unsatisfactory. Datasets are sorted by decreasing size of the training set from left to right.

		WIKIDATA5M			
		Regression (RSE - lower is better)			
		Airport Elev.	Season Start	Population	Munic. Area
ComplEx	STD+LP	.849±.007	.596±.002	.019±.197	.801±.000
	MTT+LP	.917±.000	.695±.014	.785±.139	.867±.000
	MTT+MTR	.928±.000	.657±.040	.841±.086	.877±.000
TransE	STD+LP	.734±.019	.546±.029	.928±.000	.811±.000
	MTT+LP	.894±.037	.654±.024	.739±.087	.825±.000
	MTT+MTR	.873±.000	.610±.011	.896±.080	.825±.000

Table D.7: Part 2: Relative squared error (RSE) on test data of downstream models (MLP and Linear Regression) that use pre-trained KGE embeddings as input to solve regression tasks about entities in WIKIDATA5M; Models with RSE above 1 are considered unsatisfactory. Datasets are sorted by decreasing size of the training set from left to right.

	Training	*Selection*	*Downstream tasks* EC (↑)	REG (↓)
ComplEx	STD	LP	.844±.008	.447±.051
	MTT	MTR	.843±.002	**.412±.037**
	w/o LP	MTR	.870±.002	.512±.044
	w/o REL	MTR	.851±.005	.486±.035
	w/o DOM	MTR	.851±.003	.479±.029
	w/o NBE	MTR	.856±.002	.562±.038
	w/o NBR	MTR	.858±.000	.459±.062
	w/o LP+REL	MTR	.849±.011	.542±.054
	w/o DOM+NBR	MTR	.856±.001	.415±.029
DistMult	STD	LP	.873±.009	.551±.062
	MTT	MTR	.857±.006	.482±.026
	w/o LP	MTR	.861±.008	.522±.067
	w/o REL	MTR	.868±.008	.536±.077
	w/o DOM	MTR	.849±.002	.478±.027
	w/o NBE	MTR	.844±.002	.524±.047
	w/o NBR	MTR	.859±.002	.493±.043
	w/o LP+REL	MTR	.856±.001	.572±.085
	w/o DOM+NBR	MTR	.839±.001	.545±.060
RotatE	STD	LP	.868±.003	.797±.286
	MTT	MTR	.847±.001	.704±.060
	w/o LP	MTR	.874±.000	.661±.043
	w/o REL	MTR	.862±.003	.692±.079
	w/o DOM	MTR	**.898±.001**	.593±.078
	w/o NBE	MTR	.862±.003	.558±.050
	w/o NBR	MTR	.863±.007	.552±.035
	w/o LP+REL	MTR	.864±.001	.743±.123
	w/o DOM+NBR	MTR	.854±.001	.809±.249
TransE	STD	LP	.873±.015	.742±.287
	MTT	MTR	.878±.009	.681±.095
	w/o LP	MTR	.870±.000	.486±.027
	w/o REL	MTR	.856±.001	.622±.061
	w/o DOM	MTR	.863±.000	.539±.052
	w/o NBE	MTR	.884±.002	.463±.032
	w/o NBR	MTR	.857±.007	.458±.024
	w/o LP+REL	MTR	.860±.001	.594±.032
	w/o DOM+NBR	MTR	.864±.001	.497±.057

Table D.8: Performance on downstream tasks for FB-237 using different pre-training objectives and model selection. Excluding the LP task in pre-training often improves performance.