Informatics

Recursive Rule Injection in Knowledge Graphs

Exploiting Logical Knowledge in Machine Learning

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Felix Karl Michael Wagner, BSc
Matrikelnummer 01429339

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Prof. Dr. Emanuel Sallinger
Mitwirkung: Dipl.-Ing. Markus Nissl
Dipl.-Ing. Aleksandar Pavlovic

Wien, 10. November 2021

Felix Karl Michael Wagner   Emanuel Sallinger

Informatics

Recursive Rule Injection in Knowledge Graphs

Exploiting Logical Knowledge in Machine Learning

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Felix Karl Michael Wagner, BSc
Registration Number 01429339

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr. Emanuel Sallinger
Assistance: Dipl.-Ing. Markus Nissl
Dipl.-Ing. Aleksandar Pavlovic

Vienna, 10th November, 2021

Felix Karl Michael Wagner   Emanuel Sallinger

Erklärung zur Verfassung der Arbeit

Felix Karl Michael Wagner, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 10. November 2021

Felix Karl Michael Wagner

Acknowledgements

First and foremost, I am extremely grateful to my supervisor Prof. Dr. Emanuel Sallinger for his incredible support, patience, and guidance throughout this thesis. I would also like to thank Dipl-Ing. Markus Nissl, and Dipl-Ing. Aleksandar Pavlovic, for their expertise, insightful comments, and fast feedback.

Finally, I would like to thank my friends and family for helping me laugh away my stress.

Without all the support, I would not have been able to finish my master's degree.

Kurzfassung

Knowledge Graphs (KGs) sind ein vielversprechendes Forschungsgebiet in der Künstlichen Intelligenz und werden verwendet, um Wissen zu repräsentieren, zu verwalten und zu verarbeiten. Eine Stärke von ihnen ist es, symbolisches Wissen zu repräsentieren. Jedoch kann die Manipulation von diesem Wissen spezielle Probleme verursachen. Aus diesem Grund wandelt das Forschungsgebiet der Knowledge Graph Embeddings (KGEs) das symbolische Wissen der KGs in den sub-symbolisches Raum um.

Allerdings basieren KGs häufig auf fehlerhaften und nicht kompletten Datensätzen. Das Trainieren der KGEs mit unvollständigen Daten ist eine Herausforderung. Der Verlust der Erklärbarkeit im sub-symbolischen Raum stellt ein weiteres Problem dar. Um genau diese zwei Probleme zu beheben, wurden Embedding Modelle entwickelt, welche teilweise erklärbar sind. Diese Modelle erlauben es, wertvolles Hintergrundwissen, welches im KG in Form von logischen Regeln bereits enthalten ist, in den Lernprozess der KGEs zu integrieren.

Bisher wurden einige Ansätze zur Einarbeitung von Hintergrundwissen veröffentlicht. Es gibt eine Vielfalt von logischen Regelarten, jedoch unterstützen derzeitige Ansätze nur einen Teil von diesen Arten. Eine sehr wichtige und nicht unterstützte Art, welche fundamentale Möglichkeiten zum logischen Schließen in Graphen erlaubt, sind rekursive logische Regeln. Diese Diplomarbeit hat eine Methode entwickelt, welche es ermöglicht rekursives logisches Wissen mit KGEs zu vereinigen. Deshalb wurde ein existierendes KGE Modell adaptiert, welches die vorgeschlagene Methode unterstützt. Zusätzlich wurde ein neues KGE Modell im hyperbolischen Raum eingeführt, welches ebenfalls diese Methode unterstützt. Die Verwendung von nichteuklidischen Räumen als Repräsentationsraum bietet einige Vorteile, wie zum Beispiel die Möglichkeit gewisse Daten mit einer zugrundeliegenden Struktur, welche nicht im euklidischen Raum erfasst werden können, zu repräsentieren. Dieses Forschungsgebiet findet in den letzten Jahren große Beachtung.

Die Modelle wurden implementiert und mithilfe von DBPedia und synthetisch generierten Datensätzen evaluiert. Die Evaluation zeigt, dass die Einarbeitung von rekursiven Regeln die Performanz der Modelle für alle Datensätze für nichteuklidische sowie euklidische Modelle verbessert. Zusätzlich konnte unser Ansatz ein hyperbolisches State of the Art Modell in einer Metrik übertreffen.

Abstract

Knowledge Graphs (KGs) are one of the most significant fields of research in Artificial Intelligence (AI) today. KGs, in broad terms, represent, manage and process knowledge. They are exceptionally effective at representing symbolic knowledge. However, utilising symbolic knowledge may pose unique difficulties. To enable effective manipulation of this knowledge, the field of Knowledge Graph Embeddings (KGEs) maps KGs into sub-symbolic space.

However, KGs frequently encounter noisy and incomplete data. This fact represents a challenge for KGEs, which are trained on this sparse and noisy data. Another limitation of current KGE models is that explainability in latent sub-symbolic space is usually lost. Therefore, KGE models were developed that allow for some interpretation, which additionally allows the injection of precious knowledge in the form of logical rules that the KG already provides.

There are already some approaches that support the injection of background knowledge, but they are few in number. Existing approaches, however, do not support all types of logical rules, including recursive rules. Recursive rules represent the fundamental means of reasoning in graphs and are thus an essential rule type. Therefore, this thesis develops and evaluates a method to inject recursive logical knowledge into KGEs. As a result, an existing KGE approach is adapted to support the proposed method. Additionally, we propose a novel KGE model in hyperbolic space that supports the method to inject recursive logical knowledge. Changing the representation space from Euclidean to non-Euclidean provides several benefits. Non-Euclidean geometry has the advantage of being able to express certain data with an underlying structure that cannot be represented in Euclidean space. This area has received considerable interest in recent years.

These models are implemented and validated using datasets retrieved from DBPedia and generated synthetically. The evaluation demonstrates that injecting recursive rules increases the performance of Euclidean and non-Euclidean KGEs across all datasets. Additionally, our approach outperforms a state-of-the-art hyperbolic model in one particular metric.

Contents

CHAPTER 1

Introduction

Combining the strengths of symbolic AI and Statistical AI – Machine Learning (ML) – has the potential to revolutionise AI methodologies. This fusion can create better-performing systems requiring considerably less training data and provide understandable ML model decisions. Furthermore, the promising research field of KGs is inherently multi-disciplinary. Therefore, it combines many research areas, including ML and symbolic AI, where symbols/names are used for representation. As a result, KGs are a suitable candidate to develop augmentation methods of ML with symbolic AI methods.

One of the ML areas within KGs are KGEs, representing KGs as low-dimensional vector values. They can be used to detect missing information in KGs. On the other side, one of the strengths of KGs is their compatibility with symbolic reasoning using logical rules. Logical rules contain rich background information and are understandable. Incorporating them into KGEs is exactly the combination mentioned above of symbolic and sub-symbolic knowledge, but the number of existing methods is sparse.

An essential type of rules in logic represent recursive rules, which can be seen in many real-world examples, such as company ownership or Recommender Systems [BFGS19, CFG$^+$19]. However, to the best of our knowledge, there is no research regarding the injection of recursive rules into KGEs, which would allow us to solve the aforementioned fusion of KGEs and symbolic reasoning for recursive rules.

This thesis investigates this open challenge and develops a method to inject recursive rules into KGEs. This method trains vector embeddings by following the logical properties of recursive rules. The effectiveness of this method will be evaluated and analysed.

1.1 Motivation and Problem Statement

KGs are a rapidly growing research field and among the key trends in AI. In 2012 Google announced the Google Knowledge Graph [Sin12]. This announcement coined the term

"Knowledge Graph" and drew much attention to the concept [Pau17]. However, despite the considerable advancements in this research field and the high number of created KGs, there are still many definitions of the notion. In broad terms, one can say that a KG is used to represent, manage and process knowledge [BSV20b]. In Chapter 2 a definition of a KG for this thesis is presented. In this chapter, we assume that a KG is a graph that consists of certain entities (nodes) and relations (edges) that connect them.

After giving a broad definition of the notion, we introduce the benefits and challenges of the KG research field. KGs are very effective in representing symbolic knowledge, but manipulating this knowledge can be hard. Additionally, KGs often experience incomplete and noisy data [AM13, NMTG16] which affects the performance of reasoning tasks on KGs. Therefore, the research field of KGEs has emerged, which has the goal to transition KGs (entities and relations) into low-dimensional continuous vector space and still capture the semantic meanings [WMWG17]. KGEs can be used to compute scores of unobserved knowledge that indicate the possibility of the facts being true. Therefore, KGEs are useful for *Knowledge Graph Completion* tasks. They identify relevant knowledge using the computed scores and add them to the incomplete KG. Furthermore, KGEs are not only useful for Knowledge Graph Completion tasks but have a wide variety of application fields, including *relation extraction*, *entity classification* and *entity resolution*. They are also advantageous for faster reasoning over a large KG after the creation of the embedding [WMWG17].

A wide variety of proposed KGE models are using different methods, ranging from exploiting different spaces (Euclidean, complex, hyperbolic), sophisticated operations (translations, isometries), Neural Networks (NNs) to Graph Neural Networks [JPC$^+$21]. Besides this significant amount of different approaches, some works concentrated on enhancing those KGE models by incorporating additional background information. This additional background information can have a wide variety of forms; some examples are: textual information, image information, path information, temporal information, and logical rules [WMWG17, BRC$^+$20]. Incorporating additional information can yield a much better representation than just relying and training on often highly incomplete KGs. The quote from Demeester et al. [DRR16] supports this: "Recent research on combining rules with learned vector representations has been important for new developments in the field of knowledge base completion". Thus, combining additional domain expert knowledge with the actual KG to create better embedding representations is a promising direction in the field of KGEs.

Although there are approaches to inject logical rules into KGEs, the literature and the support of injectable logical rules are quite sparse [DRR16, GWW$^+$16, MCM$^+$17, FRP18, GWW$^+$18, ACLS20]. Logical rules contain rich background information and provide a high level of explainability. These two reasons represent limitations, yet current embedding models lack the capabilities to provide explainable embeddings. Already the following simple example shows the motivation to inject logical rules: Assuming we know that if a city is the capital of a country, it must be located in this country. The following logical rule states that: $\forall x, y : \mathit{CapitalOf}(x, y) \rightarrow \mathit{LocatedIn}(x, y)$. If we could train the embedding in a way that the vector values follow this logical rule and are consistent with it, the problem of predicting missing links for certain *LocatedIn* relations is solved. Furthermore, we add some explainability to the embedding models since we can translate the logical rule into an understandable human sentence and have more insights into the vector values.

However, injecting logical rules into KGEs poses several complicated challenges:

- Translating logical properties into vector space and assuring that the vector values follow this translation after training.
- Developing the method itself to inject logical rules into the training process. This step poses the question of whether to develop a method that consistently enforces the background knowledge or allows exceptions.
- Proposing a method that supports complicated rule types or a broad range of types.

As described above, these challenges have been solved for specific rule types, e.g. *implication rules*, *symmetry rules* or *anti-symmetry rules* [ACLS20, FRP18, GWW$^+$18, DRR16]. However, none of them supports recursive rules, an essential type when reasoning over KG, which per definition have a recursive structure. Recursive rules provide rich background information and successfully incorporating recursive rules has the potential to improve the performance of the KGEs significantly.

In fact, previous methods fail to capture recursive rules because they focus on incorporating rather basic and common rule types. Furthermore, the majority of the models supports injecting single logical rules independently but neglect the analysis of jointly capturing them. Since recursive rules represent a more complex rule type, previous works do not discuss an analysis or methods to inject recursive rules.

1.2 Aim of the Work

We have discussed the importance of incorporating additional background knowledge into KGEs. In this chapter we present the necessary outcomes to develop a KGE model that supports recursive rule injection. The aim of this work is to:

- *Identify and analyse state-of-the-art KGE models that support logical rule injection.*

  An analysis of existing state-of-the-art models that support the injection of logical rules and the compatibility of various logical rule types should be discussed. This analysis will be used for development of our approach.

- *Develop a method to inject recursive rules into KGE models.*

  The method should capture the semantics of the recursive rules and guide the learning process of the vector values according to it.

- *Propose a KGE model compatible with the proposed method to inject recursive logical rules.*

  It is critical that the proposed method is supported by our proposed KGE model. Not all KGE models are compatible with our proposed technique due to the inability to enforce certain numerical constraints.

- *Implement the KGE model and evaluate the effectiveness of the recursive rule injection.*

  The KGE model and the developed method need to be implemented and evaluated on suitable data.

These findings serve as the foundation for this thesis and constitute its main contributions. The following research questions are posed in the thesis as a result of the findings. The thesis poses two research questions, with the second one being divided into two sub-questions.

Research Question 1 *What is an appropriate method to inject recursive logical rules into KGEs?*

To address this research issue, we must first conduct a literature review to discover state-of-the-art models. In Section 4.1 we propose our own method for injecting logical rules based on certain commonly used techniques from state-of-the-art models. Moreover, Section 4.2 and Section 4.3 propose two KGE models to inject recursive logical rules. Having proposed a suitable method does not guarantee that the resulting output will exhibit the anticipated improvements. Therefore, the next research question concerns the evaluation and is split into two sub-questions, the first studying the overall performance and the second, an analysis of the specialities of recursive rules.

Research Question 2.1 *Does the proposed method to inject recursive rules yield better overall performance compared to the same method without injection?*

Research Question 2.2 *Does the recursion depth affect the performance of the proposed injection method?*

Chapter 5 discusses these two research questions in more detail. In order to evaluate the effectiveness of the method, we formulate a Knowledge Graph Completion problem, more concretely a link prediction task – Chapter 2 and Chapter 5 give a more detailed explanation on the evaluation procedure – a typical evaluation task for state-of-the art KGE models. For the evaluation, we use two datasets, one extracted from DBPedia and another synthetically generated one, and use the standard metrics, i.e., Mean Reciprocal Rank (MRR) and Hits at K (H@K), to measure the effectiveness of our approaches. Section 5.2 discusses the characteristics of the used datasets. Our synthetically generated datasets enable us to control the recursion depth, which allows us to evaluate the effect of the recursion depth.

1.3 Methodological Approach

This thesis uses the following methodological approach:

1. **Literature review**

   In this step, a literature review is conducted. The literature review should focus on the area of KG, KGEs and logical knowledge in KG. One goal is to identify existing KGE models and classify state-of-the-art methods. Furthermore, KGE models that support logical rule injection are identified, and the supported rule types are compared. One of our proposed approaches is based on hyperbolic geometry. Therefore, the literature review focuses on gathering standard literature on non-Euclidean geometry.

2. **Develop an algorithm to inject recursive rules**

   Based on the literature review, common techniques to inject logical rules into KGEs are gathered. The next step is to develop an algorithm to inject recursive logical rules. The standard techniques we identified during the literature review should provide a reference point to developing this algorithm.

3. **Propose KGE model that supports the developed method to inject recursive rules**

   In this step, an existing state-of-the-art model is modified to support the previously developed algorithm to inject recursive logical rules. Moreover, a new KGE model in hyperbolic space is proposed that also supports the developed algorithm.

4. **Implementation**

   The developed algorithm, along with the proposed KGE models are implemented. Additionally, a synthetic data generator is implemented for evaluating the experiments.

5. **Evaluation and interpretation**

   The hypotheses for the research questions are postulated, and the experiments to investigate those are constructed. Afterwards, we run the experiments on a datasets from DBPedia and the synthetic data generator. Lastly, the results are interpreted.

1.4 Main Contributions

This thesis exploits logical knowledge in ML. In this section, we summarise the main contributions of this thesis.

First, we propose a general method to inject recursive logical rules into KGEs by transforming recursive rules into a set of implications based on recursion depths. To the best of our knowledge, there does not exist a KGE model that supports recursive rule injection. We utilise this method to extend the existing KGE model SimplE$^+$ [FRP18] to support recursive rule injection. Furthermore, we propose a KGE model in hyperbolic space that supports the injection of implications and, in turn, our proposed method to incorporate recursive logical rules. We have implemented both models.

Furthermore, this thesis evaluates the performance of the models using synthetically generated datasets and a dataset queried from DBpedia. In addition, we discuss the implementation of the synthetic data generator. We present the achieved results and compare them to state-of-the-art KGE models. The evaluation shows that the injection of recursive logical rules does yield a performance gain for both models. Our proposed method outperforms a state-of-the-art KGE model in one metric. Moreover, the hyperbolic model performs better in general. Lastly, this thesis presents a discussion about the influence of the recursion depth regarding the performance.

In conclusion, this thesis develops and evaluates a method to incorporate recursive logical background knowledge in KGEs.

1.5 Structure of the Work

This thesis is structured as follows:

Background. Chapter 2 summarises important concepts of KGs, KGEs and non-Euclidean geometry needed for the remaining thesis. This chapter starts by giving a formal definition of a KG. Afterwards, KGEs are introduced. Lastly, an introduction to non-Euclidean geometry is presented.

Related work. Chapter 3 presents current state-of-the-art KGE models and identifies models that support logical rule injection. Furthermore, an overview of the supported rule types is given.

Proposed approach. Chapter 4 introduces our developed algorithm to inject recursive logical rules. Moreover, we discuss how this algorithm is integrated into an existing KGE model. Additionally, we propose a new KGE model in hyperbolic space that is also compatible with this algorithm.

Evaluation. Chapter 5 describes the data we used to evaluate our approaches along with the implementation of the synthetic data generator. Furthermore, the experiments and results are discussed and analysed.

Conclusion. In the last Chapter 6, we conclude this thesis by answering the proposed research questions. Lastly, future research directions are presented.

CHAPTER 2

Background

KGs combine multiple research fields within AI and have a broad application field. This chapter introduces KGs using a running example that will be referenced throughout this thesis. It will demonstrate the key concepts and discuss real-world application fields in the financial sector. Since there does not exist a commonly accepted definition of a KG, this chapter presents the definition we are using for this thesis. This thesis focuses on the combination of logical knowledge and KGE methods. Therefore, background on logical knowledge and logical reasoning in KGs is discussed in Section 2.3. The concepts of KGE models are presented in Section 2.4. Moreover, using non-Euclidean space as representation space in ML yielded promising results in recent years, and one of our proposed approaches is based on hyperbolic geometry. Therefore, the key concepts of non-Euclidean geometry are introduced in Section 2.5.

2.1 Running Example

This section introduces a running example that helps to better illustrate the concepts in the following sections. KGs gained much interest in a large number of business sectors. Especially in the financial domain, there are a lot of interesting application areas for KGs [BFGS19]. The paper [BFGS19] discusses three applications in the financial sector: *company ownership*, *detection of close links* and *detection of family-owned businesses*. Specifically, *company ownership graphs* offer promising research and applications fields [BBG+20].

A *company ownership graph* is a directed graph representing companies/shareholders as entities, and the edges represent the amount of amount of shares a shareholder owns from another entity in the graph. Those networks have complex structures, and therefore it is hard to analyse the control structures of companies. Direct ownership relations are relatively easy to spot, but once there are very long and intertwined control structures, the reasoning process gets complicated [BBC+20]. As stated in [BBC+20] those long and intertwined control chains are present in real-world ownership graphs. The paper [BBC+20] gives the following definition of the company control problem:

Definition 2.1.1 (Company Control) *"A company **x** controls a company **y**, if: (i) **x** directly owns more than 50% of **y**; or, (ii) **x** controls a set of companies that jointly (i.e., summing their shares), and possibly together with **x**, own more than 50% of **y**."[BBC+20]*

The reasoning over company ownership graphs is essential to unwind those control structures and prevent, for example, hidden hostile takeovers of companies. In [BBC+20] the authors discuss reasoning methods against hostile takeovers of *strategic relevant* companies (e.g. transport, military or energy sector). Those companies should remain in control of trusted shareholders. If knowledge about the persons involved is available in the ownership graph, it can also be used for the detection of family-owned businesses, as discussed in [BFGS19].

Therefore, we can see that those company ownership graphs can be used in various financial application fields and are useful for economic research. This motivates to focus this thesis on a simplified form of company ownership graphs, namely *company subsidiary* graphs. From a legal perspective, the definition of ownership and, as a result, the exact definition of a subsidiary depends on the legal structure, country and other factors [BB01]. However, in most articles and websites, a subsidiary (daughter company) is defined as a company that is owned by another company (the parent company), where ownership is defined similarly to Definition 2.1.1.

In [Vij06], the author uses the following definitions. A *subsidiary corporation* is a "corporation in which a parent corporation has a controlling share." [Vij06]. *Controlling share* is defined as owning more than 50% of the shares. Additionally, the author uses the definition of an *affiliate* that is "a corporation that is related to another corporation by shareholdings or other means of control; a subsidiary, parent or corporation" [Vij06]. Furthermore, [Vij06] gives the explanation that (strictly speaking) one should use the term *parent-subsidiary structure* when ownership exceeds 50% and *affiliated-firm structure* when the company holds only minority interests. However, in [Vij06] the two terms are used interchangeably since the economics of the problems in the work of the author are similar. For this thesis we use these definitions and also use the term *parent-subsidiary* for *affiliated-firm structure*. Another reason for using these two terms interchangeably is that those ownership structures change over time. For example, one year, a company is considered as the *parent*. A year later, the company only has 49% of shares but is still related to the daughter company by shareholdings or other means of control. Furthermore, if we would differentiate between *parent-subsidiary* and *affiliated-firm structure* and only investigate *parent-subsidiary* structures, we would restrict each child node in the graph to only have one parent node. This restriction represents a hierarchical structure. However, if we extend *parent-subsidiary* structures with *affiliated-firm* structures, we can evaluate our embedding methods on more complex KG structures and also unwind more complex company structures. Therefore, we define a company subsidiary graph for this thesis as:

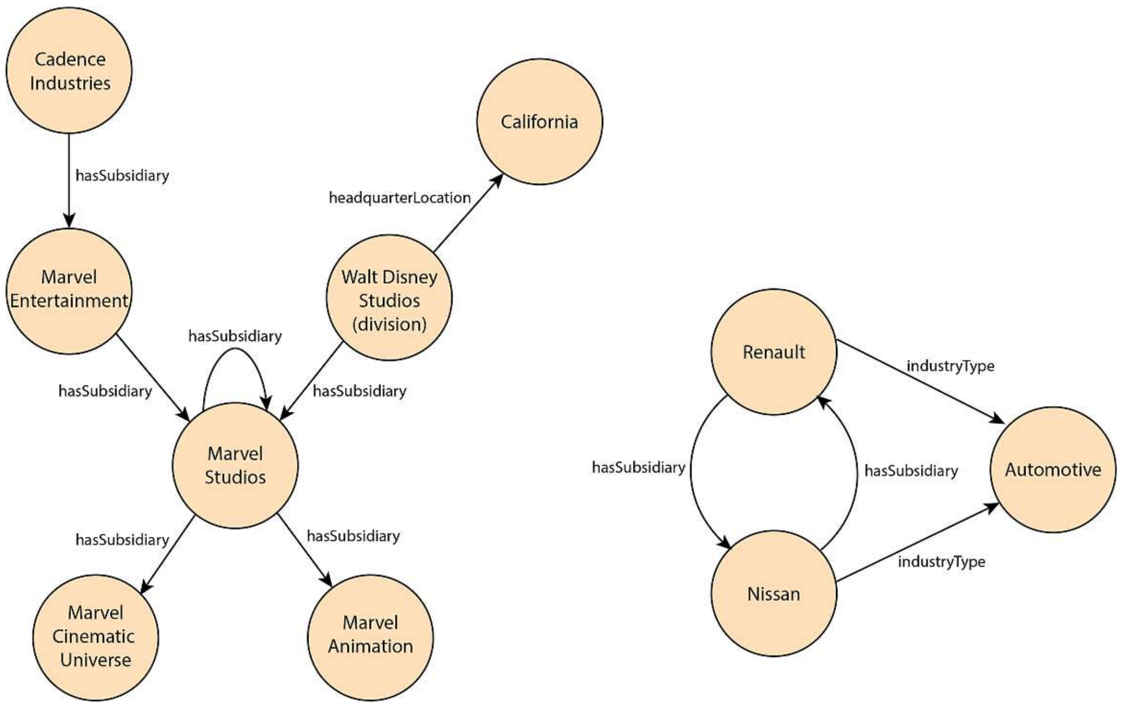


Figure 2.1: Company Subsidiary Graph

Definition 2.1.2 (Company Subsidiary Graph) *A company subsidiary graph is a directed graph, where the nodes represent companies and a directed edge from node **x** to node **y** indicates that **x** is related to **y** by shareholdings or other means of control. Node **x** is called parent and node **y** is a **subsidiary** of **x**.*

Those company subsidiary graphs can be used for similar application fields like ownership graphs to unwind complex company structures. For example, this thesis uses KGEs to predict parent-subsidiary relations that are non existing in the data. The difference to company ownership graphs is that we do not have knowledge about the number of shares and an edge in a company subsidiary graph represents some means of control. Figure 2.1 shows our toy company subsidiary KG. We extracted the running example from DBPedia[1]. It is a small subgraph from our dataset that is discussed in more detail in Section 5.2 along with the corresponding SPARQL Query 5.1. Additionally, we queried the headquarter location *California* of *WaltDisneyStudios*(*division*) and the industry type *Automotive* manually. These additions and the toy KG itself help to better demonstrate the concepts in the following sections.

The running example contains some interesting characteristics that are important for the next sections. We highlight them in this paragraph.

[1] `https://www.dbpedia.org/`

- **Relations**: The KG contains relations between entities (companies, persons and industry types), e.g. *MarvelStudios* is a subsidiary of *MarvelEntertainment*.
- **Cyclic Relationships:** The KG has *cycles* (i.e. *Renault* is a subsidiary of *Nissan* and *Nissan* is a subsidiary of *Renault*).
- **Self-loops:** The Running Example also contains *self-loops* (i.e. *MarvelStudios* is a subsidiary of itself).
- **Subsidiaries of Subsidiaries:** This characteristic is presented by the following example: *MarvelStudios* is a subsidiary of *MarvelEntertainment* which is a subsidiary of *CadenceIndustries* (therefore *MarvelStudios* is a subsidiary of a subsidiary of *CadenceIndustries*). As a result, we have relations over multiple nodes that are not directly present in the data.
- **Multiple Types of Relations:** Besides company subsidiary relations, the Running Example also contains information about the location of the headquarter of the companies, e.g. the headquarter of *WaltDisneyStudios*(*division*) is located in *California*. Moreover, the KG contains knowledge about the industry type of companies (e.g. *Renault* is in the *Automotive* industry type).

From the characteristics, we see that the running example is *cyclic*. Furthermore, to compute the subsidiaries of subsidiaries relations over multiple entities, we use *recursion* in the following sections. Lastly, together with the proposed KGE models to predict parent-subsidiary relations that are non existing in the data, the industry types and headquarter location knowledge can be used to analyse locations of related company structures or industry types that they cover.

2.2 Knowledge Graphs

KGs are one of the recent and promising trends in AI. In the last years, they did not only attract the attention of the academic field but also of the industry [BSV20b]. The term KG gained much interest after Google released this term along with Google's idea of this concept in a blog post in 2012 [Sin12]. There is a wide variety of definitions of KGs in the literature, but there is no common accepted formal definition. Some works define KGs as Knowledge Bases (KBs) in a graph structure, or heterogeneous graphs [BSV20b].

Since this thesis concentrates on KGEs, we follow the majority of the KGEs literature and define a KG similar to [BSV20b, JPC+21]:

Definition 2.2.1 (Knowledge Graph) *A Knowledge Graph $\mathcal{KG}$ is a quadruple $\mathcal{KG} = \{\mathcal{E}, \mathcal{R}, \mathcal{F}, \Sigma\}$, where $\mathcal{E} = \{e_1, ..., e_n\}$ is a set of entities, $\mathcal{R} = \{r_1, ..., r_n\}$ is a set of relations that link entities, $\Sigma = \{k_1, ..., k_n\}$ is a set of logical rules and $\mathcal{F} = \{f_1, ..., f_n\}$ represents a set of facts. Facts are denoted as triples and are a subset of $(h, r, t) \in \mathcal{F} \subseteq$*

$(\mathcal{E} \times \mathcal{R} \times \mathcal{E})$. *Rules are first-order sentences of the following form* $\varphi \rightarrow \psi$, *where* ψ *is a first-order atom and* φ *a conjunctions of atoms.*

The following section introduces the notion of atoms formally. Furthermore, we added the set Σ to the existing definition, which represents the *logical knowledge* about the data, in the form of logical rules. This knowledge is used for the injection. Our definition represents entities and their relationships. We illustrate this definition using our Running Example 2.1:

Example 1. *Our company subsidiary KG consists of the following sets (the set of logical rules is described in the next Section 2.3 in Example 5). Note that we abbreviate the hasSubsidiary relation as hS:*

$$\begin{aligned}\mathcal{E} = \{&\mathit{CadenceIndustries}, \mathit{MarvelEntertainment}, \mathit{WaltDisneyStudios}(\mathit{division}),\\ &\mathit{MarvelStudios}, \mathit{MarvelCinematicUniverse}, \mathit{MarvelAnimation}, \mathit{Renault}\\ &\mathit{Nissan}, \mathit{Automotive}, \mathit{California}\}\end{aligned}$$

$$\mathcal{R} = \{\mathit{hasSubsidiary}, \mathit{keyPersonOf}, \mathit{industryType}\}$$

$$\begin{aligned}\mathcal{F} = \{&(\mathit{CadenceIndustries}, \mathit{hS}, \mathit{MarvelEntertainment}),\\ &(\mathit{MarvelEntertainment}, \mathit{hS}, \mathit{MarvelStudios}),\\ &(\mathit{MarvelStudios}, \mathit{hS}, \mathit{MarvelStudios}),\\ &(\mathit{MarvelStudios}, \mathit{hS}, \mathit{MarvelCinematicUniverse}),\\ &(\mathit{MarvelStudios}, \mathit{hS}, \mathit{MarvelAnimation}),\\ &(\mathit{WaltDisneyStudios}(\mathit{division}), \mathit{hS}, \mathit{MarvelStudios})\\ &(\mathit{Renault}, \mathit{hS}, \mathit{Nissan}), (\mathit{Nissan}, \mathit{hS}, \mathit{Renault}),\\ &(\mathit{WaltDisneyStudios}(\mathit{division}), \mathit{headquarterLocation}, \mathit{California}),\\ &(\mathit{Renault}, \mathit{industryType}, \mathit{Automotive}), (\mathit{Nissan}, \mathit{industryType}, \mathit{Automotive})\}\end{aligned}$$

From this definition and the running example, we can see that KGs are powerful in representing symbolic knowledge. Furthermore, they are not only limited to the financial domain but can be applied in various domains, as one can see from Google's KG [Sin12]. Additionally, from the running example, we can see that one can extend them with additional knowledge (e.g. industry type of a company). The example above is rather logic-based, and also, the next Section 2.3 discusses how logical knowledge can be used in KGs. However, this thesis puts a focus on KGE methods and how logical knowledge can be incorporated into embeddings. KGEs map symbolic knowledge into subsymbolic space, and as a result, they usually represent entities and relations as numerical vectors.

In fact, embedding approaches would assign each symbolic entity from Example 1, a vector representation. These vectors can be used for various reasoning tasks. KGEs are discussed in more detail in Section 2.4. Nevertheless, this symbolic example helps to better demonstrate and discuss the logical reasoning part of this thesis in the following sections and, finally, the combination of logical knowledge and KGE models.

2.3 Logical Knowledge in Knowledge Graphs

This section discusses the importance of logical knowledge in KGs. Logical knowledge enables logic-based reasoning over KGs that can be used to infer unobserved information, (e.g. create new links between entities), find inconsistencies or integrate new knowledge [BSV20a]. Logical knowledge can be expressed in various forms, but this thesis concentrates on logical knowledge in the form of *logical rules* and the following concepts. First, we introduce the notation. Note that we use the same notation and definitions as in [BGS18] but adapt it to our KG Definition 2.2.1.

Notation. In [BGS18], the authors introduce the Vadalog system. Vadalog is based on the logic-based programming language Datalog and, in particular on Warded Datalog$^{\pm}$. Datalog$^{\pm}$ generalises Datalog rules with existential quantification in the rule heads. Vadalog can perform complex logical reasoning tasks in KGs [BGS18]. Therefore, the following definitions are related to Warded Datalog$^{\pm}$.

A *relational schema* **S** is a finite set that contains predicates (relations) with their corresponding arity [BGS18]. This corresponds to our set of relations $\mathcal{R}$ in Definition 2.2.1 since we defined facts as triples, all $r \in \mathcal{R}$ are binary relations. Furthermore, we introduce two disjoint countably infinite sets, $\mathcal{C}$ and $\mathcal{V}$. $\mathcal{C}$ represents a set of *constants* and corresponds to the set of entities $\mathcal{E}$ of Definition 2.2.1. $\mathcal{V}$ is the set of *variables*. A *term* can be either a variable or a constant. The arguments of a relation are referred to as *tuple*. Additionally, an *atom* over the schema **S** has the form $r(\bar{t})$, where r represents a predicate of arity $n > 0$ and $\bar{t}$ is a n-tuple of terms [BGS18]. Again, in Definition 2.2.1 we restricted relations to arity two and therefore atoms correspond to facts of the set $\mathcal{F}$ in Definition 2.2.1. As in [BGS18] we will use the terms atom, triple and fact interchangeably. To give a better illustration we will provide a small example:

Example 2. *$hasSubsidiary(Renault, Nissan)$ represents an atom from the Running Example's KG 2.1.*
In triple (fact) notation, this atom is defined as: $(Renault, hasSubsidiary, Nissan)$.

2.3.1 Logical Reasoning in Knowledge Graphs

The key component of logical reasoning in KGs are *logical rules*. A very expressive and important type of rules for inferring new knowledge are implication rules. They encode 'if-then' relationships.

Example 3.

$$\boxed{\mathit{headquarterLocation}(x, y) \rightarrow \mathit{locatedIn}(x, y)}$$

*This rule states that **if** the headquarter of company x is located in y, **then** the company x is located in y. In our Running Example 2.1 we can infer that WaltDisneyStudios(division) is located in California, since the headquarter WaltDisneyStudios(division) is located in California.*

The left-hand side of the rule is referred to as *body* (if-part) and the right-hand side is called *head* (then-part) [AHV95, BGS18]. The simplest form of a *recursive* rule is if an atom appears in both head and tail. These rules are called *self-recursive* rules. Additionally, if the recursive relation appears only once in the body, it is a *linear* recursion and *non-linear* if it appears multiple times [GHLZ13]. Recursion is a fundamental concept to enable complex reasoning tasks in the KG and also allow navigation [BGS18]. Recursion enables the exploration of paths of arbitrary length in the graph. Therefore, it allows traversing through the graph and, for example, visit each node. As a result, recursion provides navigational capabilities.

Furthermore, two key concepts enable object creation and express rich knowledge, namely tuple-generating dependencies (TGDs) and equality-generating dependencies (EGDs) [Sal13]. In First-order logic (FOL) notation TGDs have the following form:

$$\forall \overline{x} (\varphi(\overline{x}) \rightarrow \exists \overline{y} \psi(\overline{x}, \overline{y})),$$

where φ and ψ are conjunctions of atoms with constants and variables [Sal13]. An example of a TGD is:

Example 4.

$$\boxed{\forall x (\mathit{company}(x) \rightarrow \exists i \ \mathit{industryType}(x, i))}$$

This rule states that for every company x, there exists an industry type i. Suppose we allow unary predicates in our Running Example 2.1 and define every entity that represents a company as a company. In that case, we can see that there exists the industry type Automotive for Renault and Nissan. We could also infer new facts, e.g. for MarvelAnimation exists the industry type Entertainment. After applying our proposed embedding method to discover subsidiary relations that are non existing in the data and together with the above TGD, we could query the industry type of a parent company and all its direct and indirect subsidiaries. This knowledge can be used to analyse the industry types this company structure is involved in.

In contrast, EGDs have the following form [Sal13]:

$$\forall \overline{x}(\varphi(\overline{x}) \rightarrow x_i = x_j)$$

Combining these three key concepts (TGDs, EGDs and recursion) enables expressive logical reasoning in KGs [BSV20a]. As in [BGS18] we omit the $\forall$ quantifier and replace conjunctions $\wedge$ with a comma in the following chapters.

As the title of this thesis suggests, the main focus of this work lies on recursive rules. We will concentrate on non-linear recursive rules with two atoms in the body. Recursive rules of the following form:

$$\boxed{r(x,z), r(z,y) \rightarrow r(x,y)}$$

This rule actually expresses the **transitive closure** of the graph. It represents a special form of the composition rule type. The different rule types are discussed in Chapter 3. Our proposed embedding methods try to inject knowledge of recursive rules of the above form that compute the transitive closure. To better illustrate the idea, we will use a small subgraph of the Running Example 2.1. In the following example, we assume a set of facts $\mathcal{F}$ (selected subset of the running example) and a set of logical rules Σ (recursive rules in this case). We apply these rules on the set of facts to infer new facts and add them to the existing set.

Example 5.

$$\begin{aligned}\mathcal{F} = \{&(\mathit{CadenceIndustries}, \mathit{hasSubsidiary}, \mathit{MarvelEntertainment}),\\ &(\mathit{MarvelEntertainment}, \mathit{hasSubsidiary}, \mathit{MarvelStudios}),\\ &(\mathit{MarvelStudios}, \mathit{hasSubsidiary}, \mathit{MarvelAnimation})\}\end{aligned}$$

The set Σ contains the following rules:

$$\boxed{\begin{array}{l} 1.\ \mathit{hasSubsidiary}(x,y) \rightarrow \mathit{hasSubsidiary}(x,y) \\ 2.\ \mathit{hasSubsidiary}(x,z), \mathit{hasSubsidiary}(z,y) \rightarrow \mathit{hasSubsidiary}(x,y) \end{array}}$$

From these recursive rules, we can infer the following new facts:

$$\begin{aligned}\mathcal{F}^{new} = \{&(\mathit{CadenceIndustries}, \mathit{hasSubsidiary}, \mathit{MarvelStudios})\\ &(\mathit{MarvelEntertainment}, \mathit{hasSubsidiary}, \mathit{MarvelAnimation})\\ &(\mathit{CadenceIndustries}, \mathit{hasSubsidiary}, \mathit{MarvelAnimation})\}\end{aligned}$$

Example 5 shows how recursive rules are applied for logical reasoning on KGs. In Chapter 4, we discuss how our approach injects recursive rules from Example 5 into KGE models.

2.4 Knowledge Graph Embeddings

This section introduces the motivation for KGEs and its key concepts. KGEs gained a lot of interest during the last years and are used for several tasks, such as *Knowledge Graph Completion*, *Entity Resolution*, *Entity Classification* and *Recommendation Systems* [BSV20a].

2.4.1 Motivation

KGs often experience incomplete and noisy data [AM13, NMTG16]. Incomplete and noisy data affects the performance of reasoning tasks on KGs. In order to resolve this issue, the field of Knowledge Graph Completion identifies relevant knowledge and adds *facts* to the KG. There are many subtasks within Knowledge Graph Completion, but this thesis concentrates on one of the most important ones, namely *link prediction* [BSV20a].

Link Prediction. Link prediction is the task of predicting relations between entities. We can distinguish between two forms [WMWG17]:

- **Relation Prediction:** This task predicts missing relations between a given head entity h and a given tail entity t. This is denoted as: $(h, ?, t)$.
- **Entity Prediction:** This task predicts either missing head entities, given a relation r and a tail entity t or predict missing tail entities, given r and a head entity h. This task is denoted as $(?, r, t)$ and $(h, r, ?)$ for the latter case.

2.4.2 Knowledge Graph Embedding General Approach

As mentioned in the beginning, one of the most prominent methods to solve Knowledge Graph Completion tasks are KGEs. The key idea of KGEs is to map *symbolic* entities and relations of a KG into low-dimensional continuous vector space. This mapping should preserve the KG's structure in vector space and simplify computation and reasoning tasks. Figure 2.2 demonstrates this key idea by using a left subgraph of the Running Example 2.1. In general, the embedding method consists of four steps [WMWG17, JPC+21]:

1. Selection of a representation space for entities and relations (e.g. Euclidean space, complex vector space or hyperbolic space).
2. Map entities and relations to selected representation space. Usually, entities and relations are represented as vectors and initialised randomly from uniform distribution [WMWG17].
3. Define a **scoring** function $s(\mathbf{h}, \mathbf{r}, \mathbf{t})$ that measures the plausibility of a fact. True facts should have higher scores than false facts. Furthermore, define a **loss** function $\mathcal{L}$ that includes the scoring function and represents the objective of the optimisation.

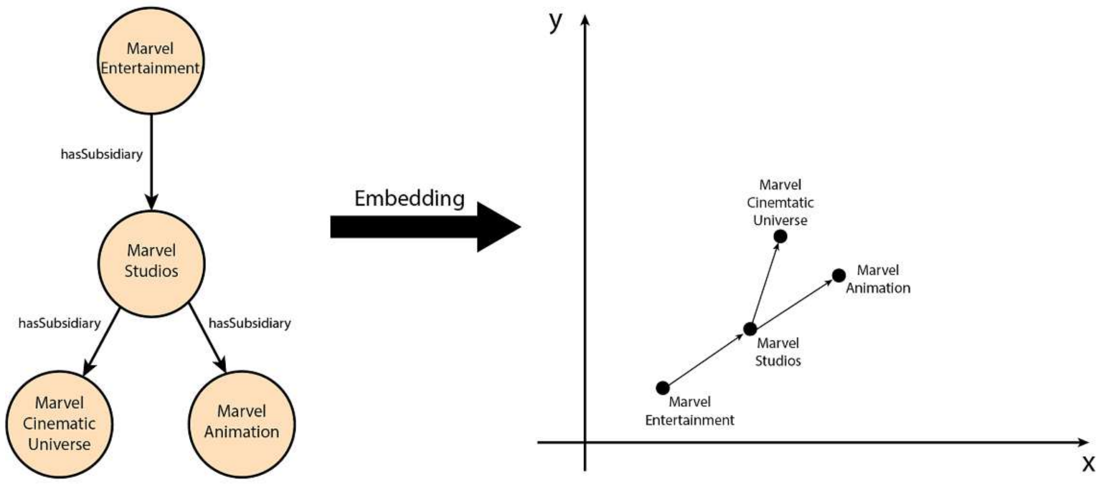


Figure 2.2: General idea of Knowledge Graph Embeddings

4. Solve the optimisation problem by minimising the loss function and in turn optimise the plausibility of the observed facts. Typically, the optimisation problem is solved by Stochastic Gradient Descent (SGD) [RM51] in minibatch mode.

The work [WMWG17] presents an algorithm for those steps. Next, we present an adapted version in Algorithm 2.1 that summarises the four steps. In this algorithm, we generate n negative examples per one positive example [TWR$^+$16b]. A more detailed explanation of the above steps is presented in the following paragraphs, but first, a rough classification of KGE methods is given. They can be divided into three categories as described in [BSV20a]:

- **Translational and Rotational Models:** These models usually model entities as points in vector space and relations as translations or rotations, respectively. Furthermore, they exploit distances or degrees as plausibility score and use them as scoring function. TransE [BUGD$^+$13] is one of the most prominent translational models.
- **Semantic Matching Models:** In contrast to translational and rotational models, semantic matching models utilise similarity-based scoring functions. Therefore, they use element-wise multiplications between entities and relations as plausibility values. An early model for this class is the RESCAL [NTK11] model.
- **Neural Network-Based Models:** These models are built on top of NNs. They are based on a multi layered-based learning approach, consisting of an encoding phase for calculating the vectors and a scoring step for evaluating the plausibilities. NTN [SCMN13] is one of the early methods in this category.

Algorithm 2.1: Learning Knowledge Graphs Embedding

```
Input: Set of positive facts $\mathcal{F}^+$, set of entities $\mathcal{E}$, set of relations $\mathcal{R}$, $n$ number of
       negative facts per positive fact
Output: Optimised entitiy and relation embeddings
1  Select representation space $\mathcal{U}$ ;
2  Initialize e ← randomly from uniform distribution for each $e \in \mathcal{E}$ ;
3  Initialize r ← randomly from uniform distribution for each $r \in \mathcal{R}$ ;
4  Loop
5      $\mathcal{S} \leftarrow$ minibatch sampled from $\mathcal{F}^+$ ;
6      $\mathcal{B}^+ \leftarrow \emptyset$ ; // Initialize positive facts in current batch
7      $\mathcal{B}^- \leftarrow \emptyset$ ; // Initialize negative facts in current batch
8      for $(h, r, t) \in \mathcal{S}$ do
9          $\mathcal{N} \leftarrow$ generate $n$ negative facts ;
10         $\mathcal{B}^+ \leftarrow \mathcal{B}^+ \cup (h, r, t)$ ;
11         $\mathcal{B}^- \leftarrow \mathcal{B}^- \cup \mathcal{N}$ ;
12     end
13     Update embeddings w.r.t. the gradients of the loss function $\mathcal{L}$, e.g.
         $\sum_{f \in \mathcal{B}^+ \cup \mathcal{B}^-} \log(1 + \exp(-y_{hrt} s(\mathbf{h}, \mathbf{r}, \mathbf{t})))$ ;
14 EndLoop
```

This thesis will concentrate on Translational/Rotational and Semantic Matching models. A more detailed discussion and explanation of KGEs can be found in the following works [NMTG16, WMWG17, BSV20a, JPC+21].

2.4.3 Training the Embedding model

This subsection discusses Algorithm 2.1 to learn KGEs in more detail. Throughout this thesis, we use lower-case bold letters to represent entities and relation embeddings, e.g. $\mathbf{e}$ and $\mathbf{r}$. In contrast, the symbolic entities and relations are denoted as lower-case non-bold letters, e.g. e and r.

Open and Closed World Assumption. There are two different assumptions under which one can train embeddings: the Open World Assumption (OWA) and the Closed World Assumption (CWA). The CWA states that all non-observed facts in a KG are false. In contrast, OWA indicates that all observed facts in the KG are true and non-observed facts can be either missing or false. These two assumptions differ in the embedding training process, especially in the selection of the loss function. More models are trained under the OWA [NMTG16]. Therefore, this subsection focuses on training embeddings under OWA.

Learning Algorithm. Algorithm 2.1 takes as input the facts of the KG, denoted as $\mathcal{F}^+$, the set of entities $\mathcal{E}$ and the set of relations $\mathcal{R}$ of the KG and additionally the number of negative facts n per positive examples (the next paragraph discusses this in more detail). First, we need to select a representation space $\mathcal{U}$ on Line 1, as described in the previous subsection. Afterwards, each symbolic entity $e \in \mathcal{E}$ and each symbolic relation $r \in \mathcal{R}$ gets assigned a random embedding value in the representation space $\mathcal{U}$ on Lines 2 and 3 (e.g. vector representation in Euclidean space).

After that, we loop over the facts $\mathcal{F}^+$ on Line 4 and generate a minibatch for SGD optimisation on Line 5. Then, since training is based on negative and positive facts, we create a set of positive facts $\mathcal{B}^+$ and a set of negative facts $\mathcal{B}^-$ for the current batch on Lines 6 and 7. Then, for each triple in the minibatch, we generate n negative facts (Line 9) and assign them to the corresponding positive and negative fact batches $\mathcal{B}^+$ and $\mathcal{B}^-$ on Lines 10 and 11. Lastly, we optimise the loss function (described in the following paragraphs) using SGD on Line 13.

Algorithm 2.1 outputs the learned embedding values of relations and entities.

Positive and negative facts. Usually, KG only represent positive facts since they do not store negative facts. Training solely on positive facts might result in overfitting [JPC+21]. Therefore, the following work [BUGD+13] proposed a method to generate negative facts. Observed facts in the KG are assumed to be positive and denoted as $f^+ = (h, r, t) \in \mathcal{F}^+$. To construct negative facts, either the head or tail entitiy of positive facts is replaced by a random entitiy. They are represented as $f^- = (h', r, t') \in \mathcal{F}^-$, where [BUGD+13]:

$$\mathcal{F}^- = \{(h', r, t) | h' \in \mathcal{E}\} \cup \{(h, r, t') | t' \in \mathcal{E}\}$$

In Algorithm 2.1 Line 9 generates these negative facts. The algorithm generates n negative facts per positive fact, [TWR+16b] showed that the number of negative examples per positive fact can influence the model's performance.

Scoring Functions. To better illustrate the loss functions in the next paragraph, the scoring function of TransE [BUGD+13] is presented as an example. TransE models relations r as translations from the head h to the tail t and therefore the following should hold: $h + r \approx t$. As a result, the scoring function models the distance and is defined as follows [BUGD+13]:

$$s(\mathbf{h}, \mathbf{r}, \mathbf{t}) = -\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|$$

On Line 13 in Algorithm 2.1 we can see that the scoring function is integrated into the optimisation step of the loss function.

Loss Functions. This paragraph introduces different loss functions. The model parameters (usually entity and relation vector representations) are denoted as Θ. Different loss functions are used to train embedding models. In this paragraph, we will introduce three major loss functions. In order to learn the embeddings, we can minimise one of the following loss functions. Equation 2.1 shows the negative log-likelihood of logistic models [JPC+21]:

$$\min_{\Theta} \sum_{f \in \mathcal{F}^+ \cup \mathcal{F}^-} \log(1 + \exp(-y_{hrt} s(\mathbf{h}, \mathbf{r}, \mathbf{t}))), \tag{2.1}$$

$$\text{where } y_{hrt} = \begin{cases} -1 & \text{if } (h, r, t) \in \mathcal{F}^+ \\ 1 & \text{otherwise.} \end{cases}$$

Algorithm 2.1 minimises the negative log-likelihood loss function with SGD in minibatch mode on Line 13.

In some models, the sigmoid function σ is applied on the scoring function: $\sigma(s(\mathbf{h}, \mathbf{r}, \mathbf{t}))$. The reason for this is to restrict the output to a probability in the range $[0, 1]$. Usually, Bernoulli negative log-likelihood is minimised for this case. For example, the SimplE$^+$ [FRP18] model uses this loss function that is shown in Equation 2.2:

$$\min_{\Theta} - \sum_{f \in \mathcal{F}^+} \log(s(\mathbf{h}, \mathbf{r}, \mathbf{t})) - \sum_{f \in \mathcal{F}^-} \log(1 - s(\mathbf{h}', \mathbf{r}, \mathbf{t}')) \tag{2.2}$$

SimplE$^+$ is discussed in more detail in Chapter 3 .

Besides these two loss functions, one can minimise the pairwise ranking loss as shown in Equation 2.3. TransE [BUGD+13] exploits this loss function. The goal of this loss is to rank positive facts higher than negative ones, where the margin γ separates them [WMWG17]:

$$\min_{\Theta} \sum_{f^+ \in \mathcal{F}^+} \sum_{f^- \in \mathcal{F}^-} \max(0, \gamma - s(\mathbf{h}, \mathbf{r}, \mathbf{t}) + s(\mathbf{h}', \mathbf{r}, \mathbf{t}')) \tag{2.3}$$

Most of the models also add regularization terms to the loss function, like $L2$ regularization: $\lambda \|\Theta\|_2^2$, where λ is a *regularization hyperparameter* [WMWG17]. Regularization terms are used to prevent overfitting by biasing the parameters closer to the origin, add information or impose certain constraints [DFO20a].

Incorporating additional information. This section discussed the training procedure of embeddings that utilises only facts during training. Nevertheless, we can also exploit auxiliary information and *inject* it into the training process. This incorporation of additional background knowledge should increase the model's performance and make it potentially more explainable. Another advantage would be that it also tackles the problem of noisy and incomplete data as discussed in Subsection 2.4.1 since the additional information should compensate for missing facts. This auxiliary information can range from textual information, images, ontologies to logical rules [WMWG17, BSV20a]. This thesis concentrates on logical rules as additional background knowledge, and Chapter 3 presents state-of-the-art models that inject logical rules.

2.5 Non-Euclidean Geometry

In this section, we introduce the basic concepts of *Non-Euclidean geometry*. The first Subsection 2.5.1 discusses *Euclidean geometry* along with Euclid's famous fifth postulate. This postulate plays an important role in defining Non-Euclidean geometries. Section 2.5.2 introduces key concepts of *Differential Geometry* that are needed for *Hyperbolic Geometry*. Afterwards, Section 2.5.3 defines one specific hyperbolic model, namely the Poincaré ball.

2.5.1 Euclidean Geometry

In this subsection, we review the basic concepts of *Non-Euclidean* geometry. First, we start by introducing the well-known *Euclidean* geometry. Euclid formulated five *postulates* (axioms) in his books (the *Elements* [EH56]) on which he based his plane geometry [RG11]:

1. "A straight line segment can be drawn joining any two points."
2. "Any straight line segment can be extended indefinitely in a straight line."
3. "Given any straight line segment, a circle can be drawn having the segment as radius and one endpoint as centre."
4. "All right angles are congruent."
5. "If two lines are drawn which intersect a third in such a way that the sum of the inner angles on one side is less than two right angles, then the two lines inevitably must intersect each other on that side if extended far enough." [RG11]

The last postulate is also known as *parallel postulate*. In other words, it states that if there is a line l and a point $\mathbf{A}$ that does not lie on this line, there exists *exactly one* line going through $\mathbf{A}$ that is parallel to l. Mathematicians tried to prove Euclid's fifth postulate from the other four postulates but were unsuccessful in finding a proof. Therefore, they started to construct geometries by replacing the parallel postulate with alternate versions. As a result, the two Non-Euclidean geometries with their adapted parallel postulate were introduced:

- **Hyperbolic Geometry:** There exist *indefinitely many* lines going through $\mathbf{A}$ that are parallel to l.
- **Spherical/Elliptical Geometry:** There exists *no* line going through $\mathbf{A}$ that is parallel to l.

Figure 2.3 shows the adapted fifth postulate for hyperbolic geometry. All three lines going through point $\mathbf{A}$ are parallel to line l [RG11].

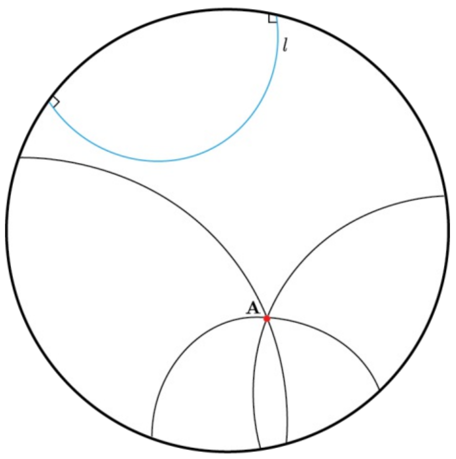
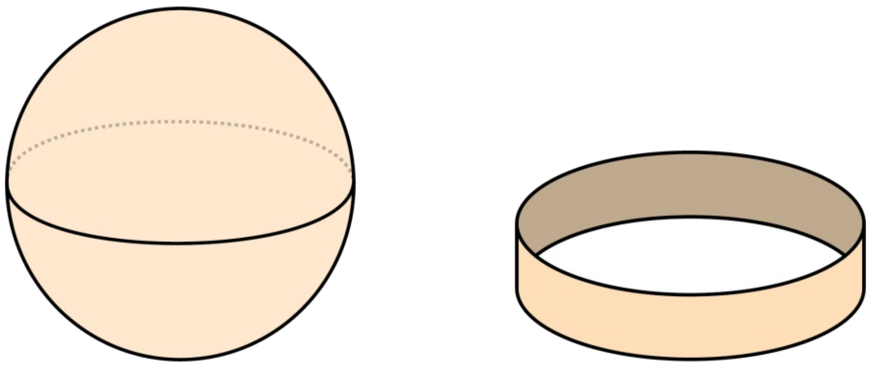


Figure 2.3: Three parallel lines to l, going through point **A** in hyperbolic Poincaré disk model



(a) Sphere $\mathbb{S}^2$ (b) Cylinder without top and bottom disk

Figure 2.4: Examples of 2-manifolds

2.5.2 Key Concepts of Differential Geometry

Next, we introduce the key concepts of *Differential Geometry* that are needed to develop the hyperbolic embedding model. We present an intuitive explanation of the concepts.

Manifold. First, we need to define the notion of a *manifold*. A n-dimensional manifold $\mathcal{M}$ is a topological space that locally represents the Euclidean space $\mathbb{R}^n$ near any of its points. For example, 2-dimensional manifolds represent *surfaces*, since the $\mathbb{R}^2$ space is a plane. Intuitively, if we zoom in enough, the neighbourhood of each point on a 2-manifold looks like a plane. Figure 2.4 shows two examples of 2-manifolds, the sphere $\mathbb{S}^2$ and the cylinder without top and bottom disk. Other examples would be paraboloids and hyperboloids. In many applications of manifold theory, the computation of curvatures and volumes are needed. These computations are done by differentiation and integration.

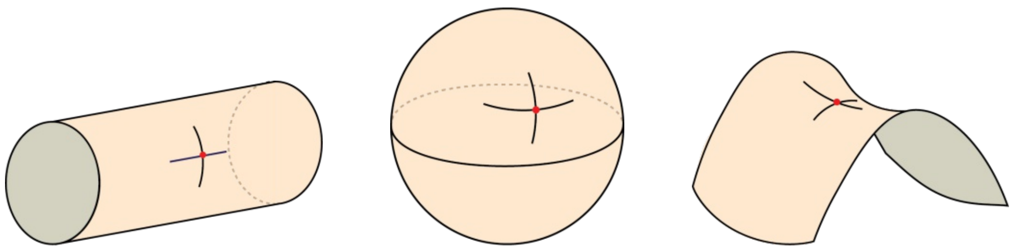


Figure 2.5: Surfaces with different curvatures. Idea of figure taken from [HNA$^+$97]

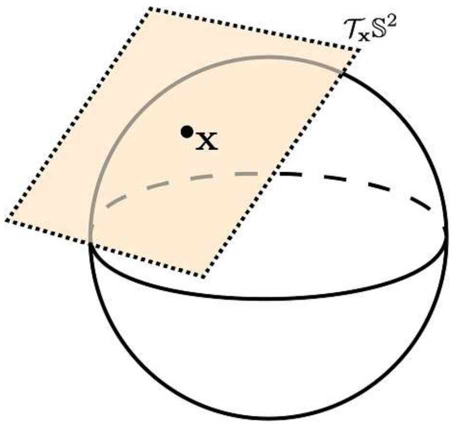


Figure 2.6: Tangent space $\mathcal{T}_\mathbf{x}\mathbb{S}^2$ of point $\mathbf{x}$ on $\mathbb{S}^2$. Idea of figure taken from [Lee13]

Therefore, the ideas of calculus need to be transferred to a manifold. Without giving a formal definition, a *smooth manifold* is a manifold on which one can do calculus. We need this notion for the definition of the Non-Euclidean models [Lee13, Wei14].

Curvature. Another important concept is *curvature* which measures how an object deviates from a flat plane. Figure 2.5 illustrates different curvatures. At a high level, a surface has zero curvature at a point if at least one of the *principal curvatures* (the two lines intersecting the red point in Figure 2.5) is zero. It has positive curvature if the two principal curvatures bend in the same direction and negative curvature if they bend in different directions [HNA$^+$97].

Tangent Space. The next notion that is essential for the hyperbolic embedding model is the *tangent space*. Intuitively, if there is a n-dimensional manifold $\mathcal{M}$ embedded in $\mathbb{R}^{n+1}$, “the tangent space $\mathcal{T}_\mathbf{x}\mathcal{M}$ at point $\mathbf{x}$ on $\mathbb{M}$ is a n-dimensional hyperplane in $\mathbb{R}^{n+1}$ that best approximates $\mathcal{M}$ around $\mathbf{x}$” [CYRL19]. The tangent space contains all possible directions of curves that pass through the point $\mathbf{x}$. Figure 2.6 illustrates an example of the tangent space $\mathcal{T}_\mathbf{x}\mathbb{S}^2$ of point $\mathbf{x}$ on $\mathbb{S}^2$ [CYRL19].

Riemannian Metric. The next step is to equip smooth manifolds with a *Riemannian metric* which enables the computation of lengths and angles on the manifold. A Riemannian metric $\mathbf{g} = (g_{\mathbf{x}})_{\mathbf{x} \in \mathcal{M}}$ determines an inner product $\langle \cdot, \cdot \rangle_x := g_x(\cdot, \cdot)$ on each tangent space $\mathcal{T}_{\mathbf{x}}\mathcal{M}$. The Riemannian metric defines how to compute the length of smooth curves on the manifold. Given a curve from a to b as continuous function $\gamma : [a, b] \rightarrow \mathcal{M}$, the length is computed by integration:

$$L(\gamma) = \int_a^b \sqrt{g_{\gamma(t)}(\gamma'(t), \gamma'(t))} dt$$

As a result, the computation of curve lengths on manifolds enables the measurement of distances on manifolds [CBG20]. In contrast, a 'normal' *metric* measures the distance between two points. For example, in Euclidean space, if we use the dot product as inner product, we get the following distance:

$$d(\mathbf{x}, \mathbf{y}) := \sqrt{\langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle}$$

between $\mathbf{x}$ and $\mathbf{y}$. The following mapping is a metric [DFO20b]:

$$(\mathbf{x}, \mathbf{y}) \mapsto d(\mathbf{x}, \mathbf{y})$$

Riemannian Manifold. After defining a smooth manifold and a Riemannian metric, we can define a *Riemannian manifold* [CYRL19]:

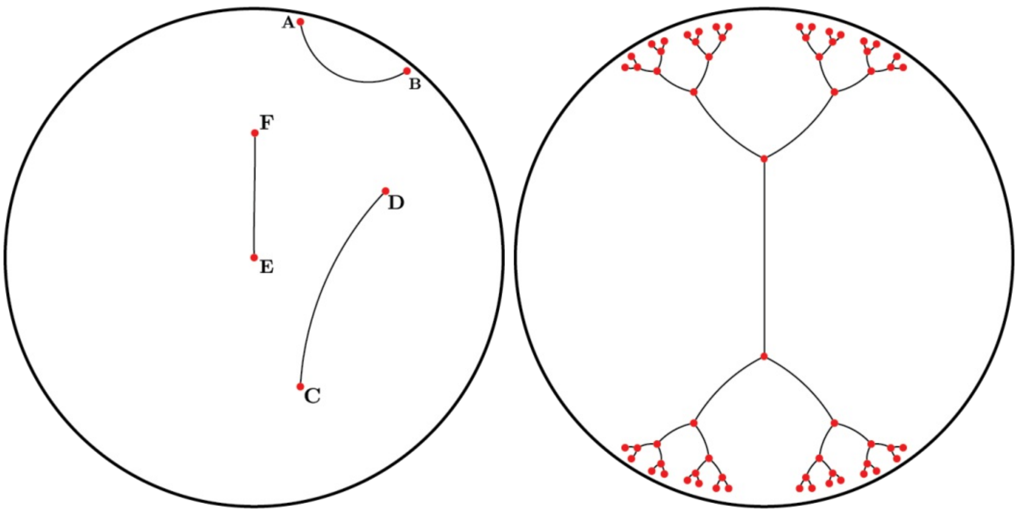
Definition 2.5.1 (Riemannian manifold) *A Riemannian manifold is a pair* $(\mathcal{M}, \mathbf{g})$, *where* $\mathcal{M}$ *is a smooth manifold and* $\mathbf{g}$ *is a Riemannian metric.*

2.5.3 Hyperbolic Geometry

In general, hyperbolic geometry studies space with constant **negative** curvature. There exist several models of hyperbolic space, but for this thesis, we concentrate on the *Poincaré disk* and its generalised form to higher dimensions, the *Poincaré ball*. In these models, n-dimensional hyperbolic space is modelled in an n-dimensional ball or on a disk (for the 2-dimensional case). Figure 2.7 shows the Poincaré disk model.

The space grows exponentially towards the edge of the disk. In this model, straight lines are represented as arcs that are bent towards the centre of the disk and are perpendicular to the disk's edge (see Figure 2.3). Furthermore, because of the exponential growth towards the edge, the shortest paths (geodesics) between two points appear curved since it is shorter to go into the direction of the centre than to directly go straight to the second point (see Figure 2.7a) [CCD17].

This exponential growth also motivates the usage of hyperbolic space as representation space in ML. Hyperbolic space is well-suited to model hierarchical data, such as trees. In trees, the number of leaves grows exponentially, and this exponential growth is exactly represented in hyperbolic space. Figure 2.7b shows a binary tree embedded in the



(a) Geodesics (shortest paths) (b) Embedded binary tree

Figure 2.7: Poincaré disk model. Idea of figure taken from [NK17, CYRL19]

Poincaré disk model. If we interpret this Figure in the Euclidean space perspective, we can see that the distances between leaves get smaller and smaller. There is no way to embed this tree in a distance preserving way in Euclidean space, except increasing the number of dimensions. A higher number of dimensions increases the number of parameters and, as a result, can cause runtime, complexity and memory issues [NK17].

Poincaré Ball Model. As mentioned in the beginning of this section, we use the d-dimensional Poincaré ball with negative curvature $-c, c > 0$. The model is a Riemannian manifold $(\mathbb{B}^{d,c}, \mathbf{g})$:

$$\mathbb{B}^{d,c} = \left\{ \mathbf{x} \in \mathbb{R}^d : \|\mathbf{x}\|^2 < \frac{1}{c} \right\},$$

$$\mathbf{g} = \left(\frac{2}{1 - c\|\mathbf{x}\|^2} \right)^2 \mathbf{I}_d$$

where $\|\cdot\|$ denotes the L2-norm and I_d is the identity matrix. The tangent space $\mathcal{T}_x\mathbb{B}^{d,c}$ for the Poincaré ball $\mathbb{B}^{d,c}$ is a d-dimensional Euclidean space ($\mathcal{T}_x\mathbb{B}^{d,c} = \mathbb{R}^n$) [BAH19b]. This is a very important concept for the development of the embedding model since we perform Euclidean operations and optimisations on the tangent space as in [CWJ+20].

Exponential and Logarithmic Maps. In order to map from the tangent space $\mathcal{T}_x\mathbb{B}^{d,c}$ to $\mathbb{B}^{d,c}$ we use the *exponential map*: $\exp_{\mathbf{x}}^c : \mathcal{T}_x\mathbb{B}^{d,c} \to \mathbb{B}^{d,c}$. For the other direction, we use the *logarithmic map*: $\log_{\mathbf{x}}^c : \mathbb{B}^{d,c} \to \mathcal{T}_x\mathbb{B}^{d,c}$ [CWJ+20]. Closed-form expression for the *exponential map* and *logarithmic map* at the center of the ball are defined as follows: [GBH18b]:

$$\exp_{\mathbf{0}}^c(\mathbf{v}) = \tanh(\sqrt{c}\|\mathbf{v}\|)\frac{\mathbf{v}}{\sqrt{c}\|\mathbf{v}\|} \tag{2.4}$$

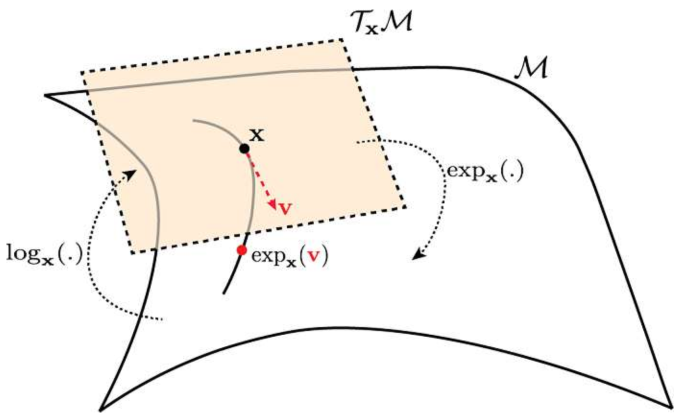


Figure 2.8: Tangent space $\mathcal{T}_{\mathbf{x}}\mathcal{M}$ on point $\mathbf{x}$ on the manifold $\mathcal{M}$. And the exponential map $\exp_{\mathbf{x}}(\mathbf{v})$ which maps $\mathbf{v}$ from the tangent space to the manifold. Idea of figure taken from [CWJ+20]

$$\log_{\mathbf{0}}^{c}(\mathbf{y}) = \operatorname{arctanh}(\sqrt{c}\|\mathbf{y}\|)\frac{\mathbf{y}}{\sqrt{c}\|\mathbf{y}\|} \tag{2.5}$$

Figure 2.8 illustrates this mapping between a manifold and a tangent space on point $\mathbf{x}$.

Vector Addition. The last concept we introduce in this subsection is vector addition in hyperbolic space. Adding two vectors in the Poincaré ball might lead to a point outside of the ball. Therefore, standard vector addition is not possible in hyperbolic space, but *Möbius addition* was proposed to tackle this problem. It is analogue to Euclidean vector addition and defined as follows [Ung01, CWJ+20]:

$$\mathbf{x} \oplus_c \mathbf{y} = \frac{(1 + 2c\langle \mathbf{x}, \mathbf{y}\rangle + c\|\mathbf{y}\|^2)\mathbf{x} + (1 - c\|\mathbf{x}\|^2)\mathbf{y}}{1 + 2c\langle \mathbf{x}, \mathbf{y}\rangle + c^2\|\mathbf{x}\|^2\|\mathbf{y}\|^2} \tag{2.6}$$

2.5.4 Spherical Geometry

This subsection discusses the concepts of spherical geometry. While hyperbolic space has constant negative curvature, spherical geometry has positive curvature. In Figure 2.5b positive curvature is presented. A common realization of this geometry is on the surface of a hypersphere $\mathbb{S}_K$ [SGB20]. Therefore, the manifold (of the hypersphere) is defined as [SGB20]:

$$\mathbb{S}_K^d = \left\{ \mathbf{x} \in \mathbb{R}^{n+1} : \langle \mathbf{x}, \mathbf{x}\rangle_2 = \frac{1}{c} \right\}, \text{for } c > 0$$

In spherical space, geodesics are great circles on the hypersphere [WHPD14]. A great circle is the intersection of a plane that goes through the center of the sphere [Wei02]. Figure 2.9 shows two great circles. Therefore, any two lines meet in two points. Furthermore, there are no parallel lines and as a result, Euclidean's fifth postulate is rejected as in the hyperbolic space [Wei21]. Moreover, the distance function can be defined as [SGB20]:

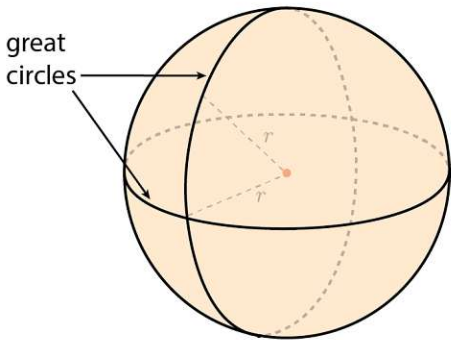


Figure 2.9: Two great circles on $\mathbb{S}^2$ with radius r

$$d^c_{\mathbb{S}}(\mathbf{x}, \mathbf{y}) = \frac{1}{\sqrt{c}} \arccos(c\langle \mathbf{x}, \mathbf{y} \rangle_2)$$

2.5.5 Combining multiple spaces

While hyperbolic space is well suited for modelling hierarchical structures, spherical space gained much popularity to embed cyclical data due to its performance [WHPD14, MHW+19, ZTJ+21]. Recently, approaches have been proposed to combine multiple spaces to model data [GSGR19, SGB20, ZTJ+21]. The reason for using multiple representation spaces is because usually, data has multiple structures; therefore, one space can not ideally capture all structures [ZTJ+21]. For example, if a data set contains hierarchical and cyclical structures, a combination of hyperbolic and spherical space might improve the performance of modelling this type of data.

Cyclic data could be embedded around a great circle on the sphere that could increase the model's performance[2]. It is essential to mention that these approaches to combine multiple spaces do not support the injection of rules. This research area is important to research which spaces are suitable to represent specific properties.

2.6 Summary

We conclude the introduction of the relevant concepts needed to present the proposed approaches and related work models. This chapter introduced a running example and real-world application fields. Furthermore, a formal definition of a KG was presented. Afterwards, we discussed the logical part of KGs and also the ML side of KGs by introducing the key concepts of KGEs. Additionally, we discussed the benefits of using

[2]https://dawn.cs.stanford.edu/2019/10/10/noneuclidean/

non-Euclidean geometry as representation space for embeddings. An introduction to hyperbolic geometry for modelling hierarchical data and spherical spaces to model cyclic data was presented. The next chapter presents state-of-the-art embedding models and additionally current approaches to inject logical knowledge into embeddings.

CHAPTER 3

Related Work

This chapter presents state-of-the-art KGE approaches. First, we present models from three classes of embeddings: Semantic Matching Models, Translational/Rotational Models and Hyperbolic Models. Furthermore, we discuss the hyperbolic approach of Chami et al. [CWJ+20] in more detail since one of our proposed approaches is based on some key ideas from this approach. Next, in Section 3.2 recent models that support logical rule injection are presented. We discuss the embedding approaches along with their ideas to inject logical rules. Moreover, we put a focus on the SimplE$^+$ [FRP18] model since our proposed Euclidean approach is based on it. There are commonly used logical rule patterns in KGE literature to discuss the representation and injection capabilities of the models. In Table 3.1 the most prominent ones are summarised [ACLS20]. To better discuss the embedding models, we use the patterns from Table 3.1 throughout this chapter.

3.1 Knowledge Graph Embeddings

This section presents current state-of-the-art KGE models. We divide them into Translational/Rotational Models and Semantic Matching Models as presented in Chapter 2 and discuss the chosen representation space of the models. Additionally, hyperbolic embedding models are presented.

3.1.1 Semantic Matching Models

As discussed in Chapter 2, Semantic Matching Models utilize similarity-based scoring functions. One of the earliest Semantic Matching Models is RESCAL [NTK11]. In RESCAL each relation r is modelled as matrix $\mathbf{M}_r$ and the scoring function is defined as [NTK11]:

$$s(\mathbf{h}, \mathbf{r}, \mathbf{t}) = \mathbf{h}^\intercal \mathbf{M}_r \mathbf{t}$$

Table 3.1: Common logical rule patterns[1]

Logical Rule Patterns	**Notation**
Symmetry (Sym)	$r_1(x,y) \rightarrow r_1(y,x)$
Anti-symmetry (A-Sym)	$r_1(x,y) \rightarrow \neg r_1(y,x)$
Inversion (Inv)	$r_1(x,y) \leftrightarrow r_2(y,x)$
Composition (Comp)	$r_1(x,y), r_2(y,z) \rightarrow r_3(x,z)$
Implication/Hierarchy (Imp)	$r_1(x,y) \rightarrow r_2(x,y)$
Non-linear Recursion (Rec)	$r_1(x,y), r_1(y,z) \rightarrow r_1(x,z)$

The paper [YYH+15] introduces the DistMult model, which is a special case of RESCAL using diagonal relation matrices. However, DistMult can not model asymmetric relations [BAH19b]. Therefore, the ComplEx model was proposed in [TWR+16a]. ComplEx is able to capture asymmetric relations [BAH19b]. ComplEx moves DistMult to complex vector space. From this approach, we can already see that changing the representation space can provide advantages in expressiveness. Another recent Semantic Matching Model is TuckER [BAH19a] that achieved promising evaluation results. This approach utilizes Tucker decomposition. Tucker decomposition was proposed in [Tuc64] and decomposes a tensor into a set of matrices, and one small core tensor [BAH19a].

3.1.2 Translational and Rotational Models

In this type of model, entities are usually modelled as points and relations as translations or rotations. The distance functions or degrees between the translated/rotated head and tail entities are typically used as a scoring function. The seminal model in this category is TransE [BUGD+13]. In Chapter 2.4.3 we discussed the scoring function of TransE. Multiple extensions to address expressiveness problems have been proposed, e.g. TransH [WZFC14], which solves TransE's problems of representing 1-to-N, N- to-1, and N-to-N relations, TransR [LLS+15], that addresses TransE's representation issues to capture multiple aspects of an entity and various relations focusing on different aspects or TransSparse [JLHZ16], which simplifies TransR's complexity. Instead of translations, the RotatE model [SDNT19] is based on rotations in complex vector space. This approach allows capturing logical properties like symmetry, inversion, composition, or anti-symmetry [CWJ+20]. Again, there is an improvement in expressiveness due to a change of the representation space. RotatE has the following scoring function [JPC+21]:

$$||\mathbf{h} \circ \mathbf{r} - \mathbf{t}|| \qquad \mathbf{h}, \mathbf{t} \in \mathbb{C}^d,\ \mathbf{r} \in \mathbb{C}^d,$$

where $\circ$ is the Hadmard (element-wise) product.

[1]For the relations we have: $r_1 \neq r_2 \neq r_3$ [ACLS20]

3.1.3 Hyperbolic Embeddings

As discussed in Section 2.5, hyperbolic space can naturally represent trees very well. Therefore, using hyperbolic space to embed hierarchical data gained much interest in recent years. Nickel and Kiela [NK17] proposed a model to embed the *transitive closure* of the WordNet noun hierarchy data set. As a hyperbolic model, they chose the Poincaré ball model. Their evaluation showed that low-dimensional hyperbolic embeddings can significantly increase the performance compared to higher-dimensional Euclidean embeddings.

Furthermore, in [CWJ$^+$20] the authors also showed that low dimensional hyperbolic embeddings can outperform Euclidean embeddings in representation and generalization ability. Therefore, hyperbolic space provides another desirable property, high expressiveness in low dimensions. Low dimensionality reduces the model complexity that increases explainability, and reduces memory complexity.

The drawback of Nickel and Kiela's embedding approach [NK17] is its incapability to model asymmetric relations. Ganea et al. [GBH18a] proposed another hyperbolic embedding approach (also using the Poincaré ball model) for embedding directed graphs to overcome this problem. The authors utilise cones in hyperbolic space to embed hierarchical data. However, the Poincaré ball model poses challenges to develop training optimisers and numerical instabilities [NK18]. As a result, Nickel and Kiela proposed another embedding model in hyperbolic space to embed hierarchical data, but instead of the Poincaré ball model, they used the Lorentz model of hyperbolic geometry [NK18]. The advantages of using the Lorentz model instead are facilitating the development of a more efficient optimiser and avoiding numerical instabilities. Our proposed hyperbolic model chooses the Poincaré ball model since Chami et al. [CWJ$^+$20] proposed another elegant solution to solve the problems as mentioned earlier in the Poincaré ball model. We discuss these solutions in more detail in the next paragraph.

However, the approaches mentioned above focus primarily on graph embeddings and are not directly developed for KGs. Therefore, we focus this thesis on embedding models developed explicitly for KGs: the MuRP model in [BAH19b] and the AttH (and its variants: RotH and RefH) in [CWJ$^+$20]. The MuRP model is a translational model in hyperbolic space that minimises the distance between the translated tail entity and a stretched head entity. This model achieves promising results in hyperbolic space. However, in comparison to AttH, the training optimisation relies on Riemannian Stochastic Gradient Descents (RSGDs) (the following paragraph introduces RSGDs and its disadvantages shortly). Furthermore, MuRP is based on hyperbolic translation only and a fixed hyperbolic curvature, reducing the model's flexibility. Since this optimisation method and the resulting structure of the approach is not compatible with our proposed approach to inject recursive logical rules. Therefore, our hyperbolic approach is based on key ideas of AttH. In the next paragraph, we discuss the AttH [CWJ$^+$20] model in more detail.

AttH, RotH and RefH. In [CWJ+20], Chami et al. proposed three hyperbolic embedding models: AttH, RotH and RefH. The idea is to use different geometric operations in hyperbolic space to capture logical patterns like symmetry or anti-symmetry. RotH is based on hyperbolic rotation since it was successfully used in RotatE [SDNT19]. RefH is based on hyperbolic reflection. The authors discuss that rotation can model *inversion*, *composition*, *symmetric* and *anti-symmetric* logical patterns. However, only some rotations can model symmetric relations whereas reflection can naturally capture *symmetric* patterns [CWJ+20]. AttH combines rotation and reflection using a hyperbolic attention mechanism to capture mixed-behaviour (neither symmetric nor anti-symmetric) relations. Furthermore, the authors apply a relation specific translation after hyperbolic attention since “translations capture tree-like structures by moving between levels of hierarchy” [CWJ+20]. The resulting scoring function is defined as:

$$s(\mathbf{h}, \mathbf{r}, \mathbf{t}) = -d^{c_r}(Q(h, r), \mathbf{t}^H)^2 + b_h + b_t,$$

where $Q(h, r)$ is a function that applies the hyperbolic attention followed by a hyperbolic translation. Furthermore, b_h and b_t are entity biases that represent margins of the scoring function, $\mathbf{t}^H$ denotes hyperbolic tail entity embedding and c_r is a relation specific curvature. Chami et al. define the curvature as a learnable parameter per relation. Therefore, each relation has its own curvature to provide more flexibility in capturing logical patterns and avoid precision errors [CWJ+20]. Section 2.5 discusses the concept of curvature.

In hyperbolic space, standard SGD is not possible and usually RSGD [Bon13] is applied to solve optimisation problems. However, in [CWJ+20] the authors discuss practical challenges using RSGD. RSGD has the disadvantage of numerical instabilities due to the computation of fractions in the distance functions (depending on the underlying hyperbolic model). Therefore, the authors use tangent space optimisation. This key idea is essential to develop our proposed hyperbolic model. In AttH all parameters are defined at the tangent space at the origin $\mathbf{0}$, this represents the parameter space. As presented in Section 2.5, the tangent space is Euclidean. Therefore, this allows Chami et al. [CWJ+20] to use standard Euclidean optimisation methods for the parameters. In order to compute the hyperbolic scoring function, the authors use the exponential map $\exp_{\mathbf{0}}^{c_r}(\mathbf{e}^E)$ and $\exp_{\mathbf{0}}^{c_r}(\mathbf{r}^E)$ to recover the hyperbolic parameters of Euclidean entity embeddings on tangent space $\mathbf{e}^E$ and Euclidean relation embeddings $\mathbf{r}^E$.

3.1.4 Spherical Space Embeddings

Proposed embedding approaches in spherical space gained much attention for embedding cyclical data. Most of these embedding approaches are not proposed explicitly for KGs. However, they have been successfully applied for text embeddings [MHW+19]. Furthermore, embeddings in spherical space have yielded promising results in the computer vision area [WHPD14]. Another interesting application is spherical embeddings for face recognition [LWY+17]. However, there are more published approaches of non-Euclidean KGE models in hyperbolic space.

To the best of our knowledge, spherical space embeddings have not been used together with rule injection. However, since fundamental research is required to inject rules into spherical KGEs using our proposed method, we do not focus on spherical space in this thesis and leave it open for future work.

3.2 Rule Injection in Knowledge Graph Embeddings

This section introduces existing state-of-the-art KGE models that support the injection of logical rules. Logical rule injection integrates logical rules into the learning process and should ideally guide the training process in a way that the learnt model follows the logical rules. We start by explaining two different approaches of existing models. There are multiple ways to inject logical rules, but in general, we can classify state-of-the-art approaches into two groups [ACLS20]:

1. Formulating a loss function that rewards predictions that satisfy predefined logical rules and penalizes predictions that violate them. A rule-based loss function often has the disadvantage that the rules have to be propositionalized, leading to performance problems. Instead of formulating a rule-based loss function, one can also add regularizers to the existing loss function to capture logical patterns. These methods are not able to provably enforce those rules. We will refer to them as *soft constraint* methods.

2. The second approach explicitly constrains the embedding space by enforcing constraints. This could be done by parameter tying of specific embeddings or by enforcing inequality constraints by applying specific functions on the vector embeddings. These approaches only support a limited number of rule types. We will refer to them as *hard constraints* methods.

As discussed in [ACLS20], it is also important to mention that there is a difference between *capturing* and *injecting* logical rules. While capturing expresses the capabilities of the embedding approach to model logical patterns, injecting indicates that predefined logical background knowledge can be enforced or at least incorporated into the model.

3.2.1 Embedding Models that Support Logical Rule Injection

This subsection gives an overview of current state-of-the-art models that support logical rule injection. Furthermore, we give an overview of the rules that are injectable into these approaches. Table 3.2 shows a summary of recent approaches.

The paper [RSR15] published one of the earliest approaches. In [RSR15], the authors model both ground atoms/facts and rules as logic formulae. Furthermore, they define a rule-based loss function that rewards predictions that satisfy the logical formulae and penalize them if they violate them. By modelling facts and more complex rules both as logical formulae, the authors can optimise them jointly. A similar approach to

Table 3.2: Embedding models that support logical rule injection

	Logical Rule Pattern					
Model	**Sym**	**A-Sym**	**Inv**	**Comp**	**Imp**	**Rec**
Joint [RSR15]	∼[†]	∼[†]	∼[†]	∼[†]	✓	✗
FSL [DRR16]	✗	✗	✗	✗	✓	✗
KALE [GWW+16], RUGE [GWW+18]	∼[†]	∼[†]	∼[†]	✓	✓	✗
DistMult^R, TransE^R, ComplEx^R [MCM+17]	✗	✗	✓	✗	✗	✗
BoxE [ACLS20]	✓	✓	✓	✗	✓	✗
SimplE$^+$ [FRP18]	✓	✓	✓	✗	✓	✗

inject logical rules is used in the KALE [GWW+16] and RUGE [GWW+18] model but specifically for KGs. The authors also employ joint optimisation by modelling facts and rules both as logical formulae. Furthermore, they also define a rule-based loss function and utilize t-norm fuzzy logic [Háj98] to compute the truth values of the formulae. In KALE [GWW+16], KG triples are modeled using the TransE [BUGD+13] model. All three approaches have the disadvantage of the logical rules being grounded (e.g. a grounded atom is a predicate applied to constants); this might cause performance issues for large data sets.

Furthermore, the approaches fail to enforce logical rules provably and assume independence between relations. Therefore, it is not guaranteed that the injected implications hold for facts not seen during training [DRR16]. As a result, these models can be classified as soft constraint models.

In order to avoid the disadvantages mentioned above, Demeester et al. [DRR16] proposed the FSL model. Their work concentrates on implication rules of the form: $r_1(x, y) \to r_2(x, y)$. In order to inject those implication rules, they introduce two constraints on the embeddings:

- **Inequality constraint:** $\mathbf{r_1} \leq \mathbf{r_2}$. This constraint is applied on each relation provided in the logical rule set Σ and $\leq$ indicates the component-wise comparison.
- **Non-negativity constraint:** $\mathbf{e} \geq 0 \;\; \forall \mathbf{e} \in \mathcal{E}$

The authors combine head and tail entity into one entity-tuple t and define the scoring function as dot product between relation and entity: $s(\mathbf{r}, \mathbf{t}) = \langle \mathbf{r}, \mathbf{t} \rangle$. The idea to map the knowledge of the implication rule into the embedding is to ensure that whenever $r_1(x, y)$ is true and has a high score, then $r_2(x, y)$ needs to be also true and has an equal or higher score, but not vice-versa. This idea captures the semantics of the implication that whenever the left-hand side is true, the right-hand side must also

[†] The authors use a rule based loss function. In this table we only list the types of rules the authors used for evaluation and focused their work on. However, the authors give explanation how their approach can be extended to support more complex rules. But their approaches fail to provably enforce the injected rules and assume independence of two atoms.

hold. The scoring function in combination with the two constraints ensure exactly this behaviour: $\langle \mathbf{r_1}, \mathbf{t} \rangle \leq \langle \mathbf{r_2}, \mathbf{t} \rangle$ for every relation in the set of rules. By choosing an appropriate loss function in combination with the constraints, the authors inject these implication rules and furthermore avoid grounding the injected rules, and the independence assumptions [DRR16]. The key idea of these two constraints to inject implication rules is used by SimplE$^+$ and is discussed in more detail in the next subsection. Our proposed approach also uses this idea.

A different approach is presented by Minervini et al. in [MCM$^+$17]. The authors focus on equivalence and inversion rules of the form: $r_1(x, y) \leftrightarrow r_2(x, y)$ and $r_1(x, y) \leftrightarrow r_2(y, x)$. Their approach investigates the compatibility of these two rules with three KGE models from the literature: TransE [BUGD$^+$13], DistMult [YYH$^+$15] and ComplEx [TWR$^+$16a]. They prove that two constraints are sufficient to incorporate the knowledge of the equivalence and inversion rules. For TransE, the following constraint must hold $\mathbf{r_1} = \mathbf{r_2}$ for the equivalence rule $r_1(x, y) \leftrightarrow r_2(x, y)$. Furthermore, the following constraint must hold $\mathbf{r_1} = -\mathbf{r_2}$ to inject the inversion rule $r_1(x, y) \leftrightarrow r_2(y, x)$. The authors chose to incorporate these rules by adding a regularizer to the loss function but discuss the other possibility of enforcing hard constraints instead [MCM$^+$17]. Therefore, we can see that defining constraints on the embedding space is able to capture the behaviour of specific logical patterns. Since the equivalence rule is trivial to inject (i.e. enforce equality constraint $\mathbf{r_1} = \mathbf{r_2}$) for most models we excluded it from Table 3.1 and Table 3.2.

Abboud et al. [ACLS20] address the problem of logical rule injection and capturing logical patterns by modelling relations as regions in the embedding space. The paper analyses and proves that certain types of rules can be injected and captured by their region-based BoxE model. The general idea is to embed entities as points in the vector space and relations as hyper-rectangles (boxes). They inject rules using a combination of *parameter tying* of boxes and growing specific boxes [ACLS20]. Table 3.2 shows what rule types their model supports. In addition to the common supported rule types in Table 3.2, their model also supports the injection of the following two rule types: $r_1(x, y), r_2(x, y) \rightarrow r_3(x, y)$ and $r_1(x, y), r_2(x, y) \rightarrow \bot$.

The work [MR18] proposes a method to inject logical rules by using adversarial training. However, our approach concentrates on enforcing hard constraints on the embedding space to inject logical rules since we want to ensure that the injected rules always hold. More specifically, one of the approaches is based on the SimplE [KP18] model and its extension SimplE$^+$ [FRP18]. The reason why we chose SimplE and SimplE$^+$ as the basis for one of our approaches is because our injection approach is compatible with these models and require fewer adjustments than other approaches. Furthermore, the evaluation results of these two models are promising. Therefore, we will explain these models in more detail in the following subsection.

3.2.2 SimplE$^+$

This subsection presents the SimplE [KP18] KGE model, its extension SimplE$^+$ [FRP18] and their method to inject logical rules. To begin with, SimplE assigns each relation two embeddings and similarly each entity two embeddings. For each relation $r \in \mathcal{R}$ there is one 'forward' embedding $\mathbf{r}^+$ and an embedding for its inverse $\mathbf{r}^-$. Furthermore, each entity $e \in \mathcal{E}$ gets assigned an embedding as head $\mathbf{e}^+$ and as tail $\mathbf{e}^-$. The authors define the scoring function of SimplE, as follows [FRP18]:

$$s(\mathbf{h}, \mathbf{r}, \mathbf{t}) = \sigma\left(\frac{1}{2}\left(\langle \mathbf{h}^+, \mathbf{r}^+, \mathbf{t}^+ \rangle + \langle \mathbf{t}^-, \mathbf{r}^-, \mathbf{h}^- \rangle\right)\right), \tag{3.1}$$

where $\langle \mathbf{x}, \mathbf{y}, \mathbf{z} \rangle$ for vectors $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ of length k is defined as the sum of the element-wise product:

$$\langle \mathbf{x}, \mathbf{y}, \mathbf{z} \rangle = \sum_{l=1}^{k} \mathbf{x}_l \mathbf{y}_l \mathbf{z}_l$$

SimplE [KP18] supports the injection of three logical rules types: (i) *symmetry:* $r_1(x, y) \rightarrow r_1(y, x)$ (ii) *anti-symmetry:* $r_1(x, y) \rightarrow \neg r_1(y, x)$ and (iii) *inversion:* $r_1(x, y) \leftrightarrow r_2(y, x)$. To inject those rules, SimplE uses a *parameter tying* approach. This approach enforces specific parameters of embedding vectors to have specific values. The following list gives an overview how these rules are injected into the embeddings using parameter tying. Additionally, the authors prove that this parameter tying enforces the injected logical rules in embedding space:

- *Symmetry rules:* Tying parameters of $\mathbf{r_1}^-$ to $\mathbf{r_1}^+$
- *Anti-symmetry rules:* Tying parameters of $\mathbf{r_1}^-$ to $-\mathbf{r_1}^+$
- *Inversion rules:* Tying parameters of $\mathbf{r_1}^-$ to $\mathbf{r_2}^+$ and the parameters of $\mathbf{r_2}^-$ to $\mathbf{r_1}^+$

In [FRP18] the authors propose the SimplE$^+$ model, that is an extended approach. SimplE$^+$ additionally supports the injection of implication rules of the form: $r_1(x, y) \rightarrow r_2(x, y)$. Similar to [DRR16], the authors propose two constraints [FRP18]:

- **Inequality constraint:** $\mathbf{r_1} \leq \mathbf{r_2}$. This constraint is applied on each relation that is contained in a given implication and $\leq$ indicates the component-wise comparison.
- **Non-negativity constraint:** $\mathbf{e}^+, \mathbf{e}^- \geq 0 \;\; \forall \mathbf{e} \in \mathcal{E}$

To enforce the non-negativity constraint on entity embeddings, the authors tested several element-wise non-linearity functions. Rectified linear unit (ReLU) yielded the best results. The ReLU function is defined as:

$$f(x) = \max(0, x)$$

In order to apply this constraint, SimplE$^+$ replaces the scoring function $s = (\mathbf{h}, \mathbf{r}, \mathbf{t})$ with $s = (f(\mathbf{h}), \mathbf{r}, f(\mathbf{t}))$ before computing the score. To enforce the inequality constraint, SimplE$^+$ learns for each relation r_1 that is in an implication $r_1(x, y) \rightarrow r_2(x, y)$, a non-negative vector δ_{r1}. The non-negativity of the vector is again ensured by applying the ReLU function on it. The vector δ_{r_1} defines how r_1 differs from r_2 and the following equality holds: $\mathbf{r_1} = \mathbf{r_2} - \delta_{r_1}$. Therefore, SimplE$^+$ does not learn an embedding for r_1 directly but computes the embedding using the value of the embedding of relation r_2 and δ_{r_1} using the above mentioned equality [FRP18]. The idea of capturing the implication rule in the embedding is the same as in [DRR16]. Whenever the left-hand side of the implication (h, r_1, t) is true, the triple should get a high score and (h, r_2, t) should get an equal or higher score. Since the entity embeddings are non-negative, the scoring function is the sum of the element-wise product and the inequality constraints on the relations in an implication are enforced, SimplE$^+$ ensures the following property: $s(\mathbf{h}, \mathbf{r_2}, \mathbf{t}) \geq s(\mathbf{h}, \mathbf{r_1}, \mathbf{t})$ for all implications $r_1(h, t) \rightarrow r_2(h, t)$.

The technique to enforce the inequality constraint in SimplE$^+$ is a crucial aspect we had to consider during the implementation of our proposed approaches. More details about the implementation are discussed in Chapter 4.

3.3 Summary

In Section 3.1 we presented recent Semantic Matching, Translational/Rotational and hyperbolic KGE models. Furthermore, we presented that some models change the representation space (e.g. complex vector space) to increase the expressiveness of the models. We concluded the section by discussing the hyperbolic AttH, RotH and RefH models [CWJ$^+$20] in more detail. Next, we discussed recent approaches to inject logical rules into embeddings. These approaches can generally be divided into 'soft' constraint methods, and 'hard' constraint approaches. Furthermore, we presented an overview of the models along with the injectable rule types. Lastly, we discussed the SimplE$^+$ [FRP18] model in more detail. The next chapter presents our two proposed approaches.

CHAPTER 4

Proposed Method

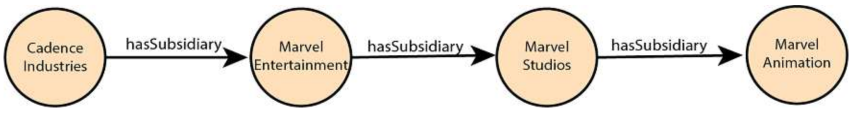
In this chapter, we present our general idea to inject recursive logical rules. It starts by explaining our approach to model the knowledge from the recursive rules using connected implications. Afterwards, we propose two models that support the injection of this approach (multiple implications). One of the proposed methods is based on the state-of-the-art KGE model SimplE$^+$ [FRP18] and the second approach is a hyperbolic embedding model.

4.1 Injecting Recursive Rules

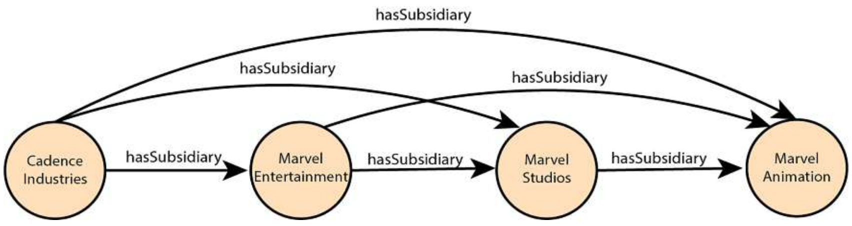
This section discusses the general idea to inject recursive rules of the form $r(x, y), r(y, x) \rightarrow r(x, z)$ into KGEs. In order to develop an approach that is able to inject recursive logical rules, we define the following steps:

1. Identify the knowledge we get from the recursive rule and decide which part of this knowledge we are able to inject.
2. Develop a method to incorporate this knowledge into vector space.
3. Develop/Find KGE models that support the injection using this method.

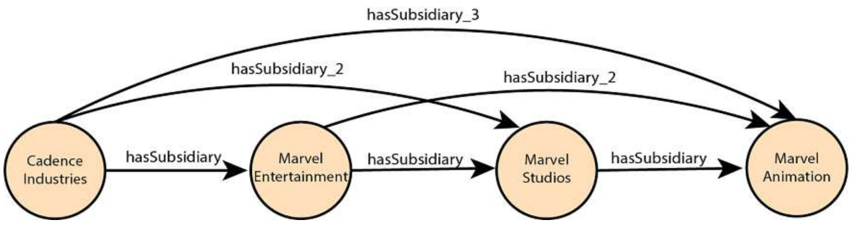
Therefore, we differentiate between the method (the general idea) how to inject recursive logical rules and the actual KGE model that supports this method. We start by presenting the general idea of how to inject recursive logical rules and afterwards present the procedure to incorporate this idea into two KGE models.



(a) Subgraph from Running Example 2.1



(b) After applying the recursive rule:
$hasSubsidiary(X, Y), hasSubsidiary(Y, Z) \rightarrow hasSubsidiary(X, Z)$.

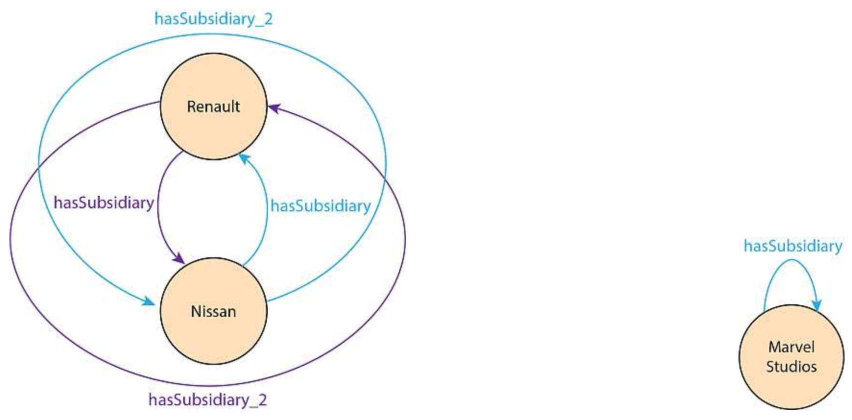


(c) After tracking the recursion depth.

Figure 4.1: Three steps to infer the recursion depths.

Step 1. For the first step, we have to identify the knowledge we get from the recursive rule and decide which part of this knowledge we are able to inject. Recursion calls itself and generates facts, and some of them are inferred from previously generated facts from this recursive rule. As a result, we get inferred facts that are somehow *connected*. Our approach should jointly capture these connected facts resulting from the logical rule. Moreover, the recursive logical rule we are investigating actually computes the transitive closure. These two aspects (connected facts and the transitive closure) represent the knowledge we get from the recursive logical rule.

Our proposed approach uses the idea of unfolding the recursive function calls by tracking the *recursion depth*. We chose the recursion depth as the knowledge we aim to inject into the embeddings. Furthermore, our approach connects these single recursion depths. Therefore, these ideas should model the connected inferred facts. We chose this idea because it should jointly capture the connection between facts and additionally order the entities in vector space according to their inferred recursion depths. The following paragraphs present this idea in more detail. Moreover, the following paragraph discusses the actual procedure of tracking the recursion depth.



(a) Recursion depth of a cycle.

(b) Recursion depth of a self-loop.

Figure 4.2: Recursion depth of cycle and self-loop from subgraphs of the Running Example 2.1.

Inferring Recursion Depth. In Figure 4.1a, a subgraph of the Running Example 2.1 is shown. In Example 5 we presented the usage of logical reasoning by applying the recursive rule $hasSubsidiary(x, y), hasSubsidiary(y, z) \to hasSubsidiary(x, z)$ on this subgraph. The application of this rule results in additionally inferred *hasSubsidiary* relations presented in Figure 4.1b. As already discussed, this represents the transitive closure of the graph. For our proposed approach, we **additionally** track the recursion depth while applying the recursive rule. In our running example, this expresses how many levels (subsidiaries) are between the parent company and its subsidiaries. The tracking process is shown in Figure 4.1c, e.g. the company *MarvelAnimation* is a subsidiary of level 3 of the company *CadenceIndustries*.

After tracking the recursion depth we additionally introduce a new relation for each recursion depth and add it to the set of relations $\mathcal{R}$. The largest recursion depth d is determined. In the case of Figure 4.1c the largest depth is 3. Formally, after applying the recursive rule $r(x, y), r(y, x) \to r(x, z)$, tracking the recursion depth and identifying the largest recursion depth, we add the following relations to the set $\mathcal{R}$:

$$\mathcal{R} \cup \{r_i \mid 1 < i \leq d\},$$

where r_i represents relation r of depth i. For Figure 4.1c this process results in the following new set of relations:

$$\mathcal{R} = \{hasSubsidiary, hasSubsidiary\_2, hasSubsidiary\_3\}$$

As already discussed, our KG contains cycles and self-loops. The following paragraph discusses this specific case in regards to tracking the recursion depth.

Recursion Depth in Cycles. Cycles and self-loops represent a special case for tracking the recursion depth. Our approach tracks the depth until we reach a node that was already

Algorithm 4.1: Transforming recursive rule to chain of implication

```
Input: Set of facts $\mathcal{F}$ (including the tracked recursion depth after applying the
       recursive rule), recursive rule of the form $r(x,y), r(y,z) \rightarrow r(x,z)$
Output: New set of logical rules $\Sigma$
1 $maxDepth \leftarrow$ find largest recursion depth in $\mathcal{F}$;
2 for $i \leftarrow maxDepth$ to 2 do
3     $k \leftarrow (r_i(x,y) \rightarrow r_{i-1}(x,y))$ ; // Create new implication rule
4     $\Sigma \leftarrow \Sigma \cup k$ ;
5 end
6 return $\Sigma$
```

visited. After encountering an already visited node, the tracking stops. For self-loops, we do not infer additional depths. Figure 4.2 shows two examples from subgraphs of our running example, where Figure 4.2a represents the cycle case and Figure 4.2b shows that we do not infer additional depths for self-loops.

Step 2. After previously stating that our approach should embed the recursion depth and connect these depths, the next step is to develop a method on how to embed these connected depths into vector space. Implication rules have already been successfully injected into KGE models (SimplE$^+$ [FRP18] or FSL [DRR16]). Therefore, we know that it is possible to inject implication rules. As a result, we define a set of implication rules based on the newly inferred recursion depth relations and connect these rules. This results in a method that embeds the recursion depths and also models the connection between the single depths. The following paragraph discusses this idea in more detail.

Chain of implications. Our approach to inject recursion depths is to build a *chain of implications*. We want to model the behaviour that larger depths imply smaller depths. For example, suppose we learn from the KG in Figure 4.1 that the company *MarvelAnimation* is a subsidiary of depth/level 3 of the company *CadenceIndustries*. In that case, we know that it is also a subsidiary of level 2 and furthermore a subsidiary in general (of level 1). This actually should represent the knowledge from the recursive rule, since the recursive rule infers (without tracking the recursion depth) the fact *hasSubsidiary*(*CadenceIndustries*, *MarvelAnimation*). By introducing the depths and the chain of implication, we aim to structure the embeddings in the representation space so that they are ordered with respect to the company structures of the KG. Furthermore, the dependencies between the relations in the set of implications should capture the recursive behaviour of connected inferred facts because the rule infers edges over multiple nodes and is not limited to only adjacent nodes.

Algorithm 4.1 presents the procedure of creating these implication rules. The algorithm takes a set of facts $\mathcal{F}$ as input that already includes the tracked recursion depths as discussed in the previous paragraphs. Additionally, it takes a recursive rule as input. Line

1 determines the relations with the largest recursion depth. Afterwards, the algorithm iterates from the largest depth to 2 on Line 2. For each iteration, the following implication rule is generated: $r_i(x, y) \rightarrow r_{i-1}(x, y)$ (Line 3), e.g. for $i = 2$ this would result in the rule $r_2(x, y) \rightarrow r_1(x, y)$. This models that larger recursion depths imply smaller recursion depths. On Line 4, the newly generated rule is added to the set of logical rules and finally returned on Line 6.

As a result, Algorithm 4.1 adds the following implication rules to our set of logical rules Σ:

$$\boxed{r_i(x, y) \rightarrow r_{i-1}(x, y), \quad d \geq i \geq 2}$$

At the end of the paragraph, an example that explains this procedure is given. Note, relations of depth one i.e. r_1 refer to the 'original' relation r: $r_1 = r$. In the following, we will use both notations interchangeably. In order to better illustrate the connection between these single implication rules, we can define them as a chain of implications. This chain of implications has the following form:

$$\boxed{r_d(x, y) \rightarrow r_{d-1}(x, y) \rightarrow \ldots \rightarrow r_2(x, y) \rightarrow r_1(x, y)}$$

The next example illustrates the procedure of Algorithm 4.1 on the KG of Figure 4.1:

Example 6. *Assuming the KG from Figure 4.1 is given, Algorithm 4.1 would create the following new implication rules:*

$$\boxed{\begin{array}{l} hasSubsidiary\_3(x, y) \rightarrow hasSubsidiary\_2(x, y) \\ hasSusbidiary\_2(x, y) \rightarrow hasSubsidiary(x, y) \end{array}}$$

This can be seen as the following chain of implications:

$$\boxed{hasSubsidiary\_3(x, y) \rightarrow hasSubsidiary\_2(x, y) \rightarrow hasSubsidiary(x, y)}$$

Step 3. Step 1 and 2 represent the proposed idea to inject recursive logical rules (introducing new relations for each recursion depth and injecting a chain of implication rules). The next step is to identify/develop KGE models that support the injection of multiple implication rules. For this purpose, we use the existing SimplE$^+$ model and develop a model in hyperbolic space. The following sections explain the procedure of injecting the chain of implications into these two KGE models and introduce the proposed hyperbolic KGE model. Furthermore, we give some details about the implementation of the approaches. These two models enforce constraints in the embedding space, and therefore, we classify them as hard constraint models (discussed in Section 3.2).

4.2 Euclidean Space Model - SimplE$^+$ Adaption

The first approach is based on the SimplE$^+$ model [FRP18] since the model can inject implication rules. The key idea of our proposed method to inject recursive rules is to inject a set of implication rules. Therefore, SimplE$^+$ is a suitable candidate for our method. Hence, we use the same techniques and ideas of the original SimplE$^+$ model: Entities and relations have two embeddings each as discussed in [FRP18] and Section 3.2.2. Moreover, the same scoring function as in SimplE$^+$ (Equation 3.1) and the same loss as in the original paper (introduced in Section 2.4 in Equation 2.2) is used.

In order to inject recursive logical rules using our proposed method with SimplE$^+$, we follow the following procedure. First, we need a recursive rule of the form $r(x, y), r(y, x) \rightarrow r(x, z)$ for the given KG that we want to inject. In the case of a company-subsidiary graph we use the already discussed rule: $hasSubsidiary(x, y), hasSubsidiary(y, x) \rightarrow hasSubsidiary(x, z)$. Our goal is to transform this rule into a set of implication rules (representing a chain of implications) and inject this set of rules. For this purpose we are using Algorithm 4.1 on our KG. The next chapter discusses the process of obtaining the training data and when the inferring of the recursion depths is executed in more detail. However, for now, Algorithm 4.1 provides us with a set of implication rules.

In Section 3.2.2 we discussed how the SimplE$^+$ model injects implication rules. Algorithm 4.1 generated a set of implication rules; this enables us to utilise the same techniques from SimplE$^+$ to inject this type of rule. Therefore, we enforce the same two constraints as in SimplE$^+$ [FRP18]: non-negative entity embeddings and inequality constraints on each relation in the implication rule set. As a result, the recursive logical rule is injected according to our proposed method. The following section discusses Algorithm 4.2 that enforces similar constraints. Therefore, Algorithm 4.2 provides a more detailed insight into how these constraints are actually enforced. However, there are certain aspects to consider during the implementation of the inequality constraints. We present them in the following paragraph.

Computing the values of the embedding vectors concerning the inequality constraint gets more complicated once implications are connected. We define *connected implications* as implications where the relation of the right-hand side of one implication is present in the left-hand side of another implication. These connections are present in a chain of implications. We demonstrate this issue by giving an example:

Example 7. *Assume the following two implication rules are given:*

$$\boxed{\begin{array}{c} r_1(x, y) \rightarrow r_2(x, y) \\ r_2(x, y) \rightarrow r_3(x, y) \end{array}}$$

These two implications are connected since r_2 is on the right-hand side of the first rule and on the left-hand side of the second rule. As described in Subsection 3.2.2, SimplE$^+$

does not learn the embeddings for r_1 and r_2 directly but their value is computed as:

$$\mathbf{r_1} = \mathbf{r_2} - \delta_{r_1}$$

$$\mathbf{r_2} = \mathbf{r_3} - \delta_{r_2}$$

From the two above equalities we can see that we can not directly compute the embedding value of $\mathbf{r_1}$ since we need the embedding value of $\mathbf{r_2}$ and to get that value, we need the embedding value of $\mathbf{r_3}$. We can substitute the second equation into the first one and get:

$$\mathbf{r_1} = \mathbf{r_3} - \delta_{r_2} - \delta_{r_1}$$

From the above example we can see that the embedding vector of $\mathbf{r_1}$ depends on the embedding vector of $\mathbf{r_3}$ and δ_{r_2}. For this example, SimplE$^+$ learns only $\mathbf{r_3}$ as 'real' embedding and for r_1 and r_2 the non-negative vectors δ_{r_1} and δ_{r_2} are learned. Therefore, the above mentioned dependencies have to be considered when computing the relation embedding values.

We used the GitHub repository[1] of the SimplE [KP18] model as codebase for the implementation. Since this code contains the implementation of SimplE and not the extended version SimplE$^+$, we had to extend the codebase. We added the computation functionality of the implications as mentioned above to inject these rules. Furthermore, we enforced the non-negativity constraint by applying the ReLU function on the entity embeddings and applied the sigmoid function to the scoring function (as it is described in [FRP18]).

4.3 Hyperbolic Space Model

This section proposes a hyperbolic KGE model that supports the injection of implication rules. Therefore, this is our second approach to inject recursive logical rules. As already discussed, the motivation to use hyperbolic space is because in [NK17] the authors achieved promising results while embedding the transitive closure in hyperbolic space. The recursive rule type we investigate computes the transitive closure. First, this section presents the general idea of our model, and afterwards, the proposed algorithm of our hyperbolic KGE model is discussed in detail.

4.3.1 General Idea

Our general idea is to specify a KGE model in hyperbolic space and map the ideas of non-negative entity embedding constraints and inequality constraints on relations to hyperbolic space in order to support the injection of a set of implication rules. The central part of a KGE model is its scoring function. As we have already discussed, the enforcement of the constraints is usually connected to the scoring function. Therefore, we

[1]https://github.com/baharefatemi/SimplE

Algorithm 4.2: Scoring function of our proposed hyperbolic model

```
    Input: Head embbedding $\mathbf{h}^E$, Relation embedding $\mathbf{r}^E$, Tail embedding $\mathbf{t}^E$,
           Hyperbolic curvature $c$, Set of implication rules $\Sigma$
    Output: Triple score $s$
    // Map relation to hyperbolic space
 1  $\mathbf{h}^H \leftarrow \exp_{\mathbf{0}}^c(\mathbf{h}^E, c)$;
 2  if $r$ is present as relation on the right-hand side of an implication $(r \to s) \in \Sigma$
 3      $\mathbf{r}^E \leftarrow$ GetBaseValue($\Sigma$);
 4  end
    // Map relation to hyperbolic space
 5  $\mathbf{r}^H \leftarrow \exp_{\mathbf{0}}^c(\mathbf{r}^E, c)$;
    // Apply hyperbolic addition:  head + relation
 6  $\mathbf{hr}^H \leftarrow$ MöbiusAddition($\mathbf{h}^H, \mathbf{r}^H, c$);
    // Map result back to Euclidean tangent space and apply
       ReLU function
 7  $\mathbf{hr}^E \leftarrow$ ReLU($\log_{\mathbf{0}}^c(\mathbf{h}^H, \mathbf{r}^H, c)$);
 8  if $r$ is present as relation on the right-hand side of an implication $(r \to s) \in \Sigma$
 9      $\mathbf{hr}^E$ += GetSumOfBias($r$);
10  end
    // Compute inner product
11  $s \leftarrow \langle \mathbf{hr}^E$, ReLU($\mathbf{t}^E$) $\rangle$;
12  return $s$
```

need to define a suitable scoring function first. Most of the Semantic Matching Models use the Euclidean inner product as similarity/scoring function [BAH19b]. From previous work, we can see that it is possible to enforce the two previously mentioned constraints using the inner product/element-wise multiplications as scoring functions. For example, in Chapter 3 we discussed that the paper [DRR16] defined two constraints: non-negative entity embedding constraint and inequality constraint on relations on the embedding space. In combination with the inner product as a scoring function, the model was able to inject implication rules. A similar approach was proposed in SimplE$^+$ [FRP18] using the sum of the element-wise multiplication as a scoring function in combination with the same constraints. As the inner product is a suitable scoring function for our embedding model, we chose it as the basis for developing our scoring function. Using the Euclidean inner product in hyperbolic space requires non-trivial transformations. These transformations and other non-trivial solutions to encountered issues are discussed in the following Subsection 4.3.2.

4.3.2 Algorithm

Algorithm 4.2 builds the basis for our hyperbolic embedding model and shows the procedure of computing the scoring function and how the constraints are enforced. We start by explaining the input of Algorithm 4.2.

In order to compute the scoring function, our algorithm takes as input the head, relation and tail embeddings. Additionally, the hyperbolic curvature c (used for computations in hyperbolic space) and the chain of implications is provided as input. Since we want to propose a KGE model in hyperbolic space, our entity and relation embeddings should be vectors in hyperbolic space. The following paragraph discusses the transformation from Euclidean embeddings to hyperbolic space.

Parameters and Optimisation. Algorithm 4.2 takes as input embeddings in the tangent space (Euclidean). The reason our algorithm does not define the embeddings in hyperbolic space directly is that, as we have discussed in Subsection 3.1.3, Chami et al. [CWJ+20] identified issues for optimisation in hyperbolic space. Therefore, the authors proposed to define the parameters in the tangent space at the origin. We follow this idea and also define all our embedding parameters in the tangent space at the origin. In order to perform hyperbolic operations on entity and relation embeddings, the exp map retrieves hyperbolic parameters. Afterwards, hyperbolic operations can be performed on them. This procedure can be seen on Lines 1 and 5; the exp map is applied to retrieve the hyperbolic parameters of the head and relation embedding. We define our Euclidean entity and relation embeddings in the tangent space as $\forall e \in \mathcal{E} : (\mathbf{e}^E)$ and $\forall r \in \mathcal{R} : (\mathbf{r}^E)$, and retrieve the hyperbolic parameters on the Poincaré ball using:

$$\mathbf{e}^H = \exp_{\mathbf{0}}^c(\mathbf{e}^E),$$

$$\mathbf{r}^H = \exp_{\mathbf{0}}^c(\mathbf{r}^E).$$

To map the parameters back, we use the log map presented in Equation 2.5. This mapping is performed on Line 7, where the parameters are mapped back to tangent space before computing the scoring function. Furthermore, we also use a Euclidean optimisation method (i.e. SGD) in the tangent space at the origin as proposed in [CWJ+20]. The cross-entropy loss is used as loss function. In order to use the tangent space optimisation, hyperbolic parameters need to be transformed back to the tangent space (Line 7) before optimisation.

Similarity function. Line 11 in Algorithm 4.2 shows the computation of the inner product on embeddings in the tangent space. Therefore, Line 11 presents our chosen similarity function. As discussed at the beginning of this section, the inner product is a suitable similarity function for our proposed approach. However, in hyperbolic space, there does not exist a clear correspondence to Euclidean inner product [TBG19]. To solve this issue and still be able to use the inner product as a similarity function, we use the following method. Tangent space is Euclidean space, and therefore, we can perform Euclidean operations there (i.e. computation of the Euclidean inner product). Our approach to use the inner product in hyperbolic space uses the idea presented in [GBH18b, CYRL19, CWJ+20], which leverages the log map presented in Equation 2.5 to move the hyperbolic embeddings to the tangent space $\mathcal{T}_{\mathbf{0}}\mathbb{B}^{d,c}$ to perform Euclidean transformations there. Figure 2.8 shows the mapping between tangent space and the manifold. As a result, this allows our approach to use the Euclidean inner product as a scoring function and still utilise hyperbolic space. The hyperbolic parameters are mapped to the Euclidean tangent space before computing the inner product between to Euclidean parameters on Line 7.

Hyperbolic Translation. In Algorithm 4.2 the hyperbolic translation (Möbius addition) is computed on Line 6. Before applying the Möbius addition, the algorithm must map the head and relation embedding to hyperbolic space on Lines 1 and 5. Utilising the hyperbolic translation is necessary since only mapping from hyperbolic space to tangent space and computing the Euclidean inner product there does not leverage the benefits of hyperbolic space. A hyperbolic geometric operation is needed to exploit the characteristics of hyperbolic space. Therefore, we decided that our approach additionally incorporates hyperbolic translation that we presented in Equation 2.6. The following paragraph discusses the combination of hyperbolic translation and the inner product that results in our final scoring function.

Final Scoring Function. This paragraph summarises the parameters, similarity function and hyperbolic translation into our proposed scoring function for Algorithm 4.2. This scoring function is essential to present the idea of injecting implication rules into our proposed model. It is a mixture of a Translational Model (to utilise hyperbolic space characteristics) and Semantic Matching Models (in order to be able to enforce constraints and inject implications). Our triple scoring function first maps the head entity's Euclidean parameters (defined in the tangent space) and relation to hyperbolic space (Lines 1 and 5). Then, the hyperbolic translation (Möbius addition) of the head and the relation is applied (Line 6). Afterwards, the translation result is mapped back to the tangent space (Line 7). This mapping allows computing the Euclidean inner product between the translation result and the tail entity (Line 11). As a result, we define the following scoring function:

$$s(\mathbf{h}, \mathbf{r}, \mathbf{t}) = \langle \log_{\mathbf{0}}^{c}(\exp_{\mathbf{0}}^{c}(\mathbf{h}^{E}) \oplus_{c} \exp_{\mathbf{0}}^{c}(\mathbf{r}^{E})) + \mathbf{b}_{r}^{E}, \mathbf{t}^{E} \rangle, \tag{4.1}$$

where $\mathbf{b}_r^E$ denotes a relation specific bias term, which we will discuss in more detail (including its computation) in the following paragraphs, and $\langle , \rangle$ denotes the Euclidean inner product.

This scoring function represents the basis for Algorithm 4.2. The next step is to incorporate rules into our proposed method.

Injecting rules. In the beginning of the section we discussed that the inner product as scoring function in combination with the two mentioned constraints enable the injection of implication rules as in [DRR16, FRP18]. Therefore, we want to incorporate these two constraints into our proposed hyperbolic model: *Non-Negativity Constraint* and

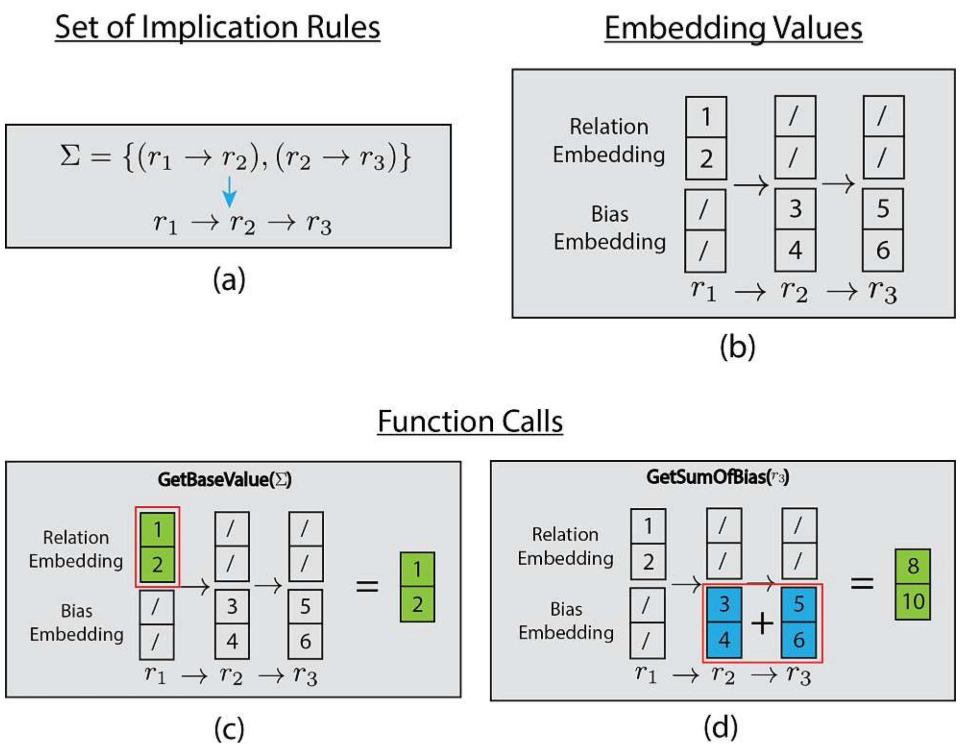


Figure 4.3: Enforcing inequality constraint of proposed hyperbolic model.

Inequality Constraint. The goal is to force $s(\mathbf{h}, \mathbf{r_2}, \mathbf{t}) \geq s(\mathbf{h}, \mathbf{r_1}, \mathbf{t})$ for all $h, t \in \mathcal{E}$, given the implication rule: $r_1(h,t) \rightarrow r_2(h,t)$. As already discussed, this injects the logical properties of the implication rule. Furthermore, this allows us to inject a set/chain of implications and in turn inject recursive logical rules according to our proposed method. The next paragraphs present both constraints.

Non-Negativity Constraint. Algorithm 4.2 enforces the non-negativity constraint on Lines 7 and 11. However, the first constraint is not bound to the entity embeddings as in [DRR16, FRP18]. Instead, we enforce the element-wise non-negativity constraint on the translational result of head entity and relation on Line 7. Furthermore, we apply this constraint on the tail entity embedding. Line 11 enforces this constraint by applying the ReLU function before computing the scoring function:

$$s(\mathbf{h}, \mathbf{r}, \mathbf{t}) = \langle f(\log_{\mathbf{0}}^{c}(\exp_{\mathbf{0}}^{c}(\mathbf{h}^E) \oplus_c \exp_{\mathbf{0}}^{c}(\mathbf{r}^E))) + \mathbf{b}_r^E, f(\mathbf{t}^E)\rangle, \tag{4.2}$$

where $f(x)$ denotes the ReLU function and $\mathbf{b}_r^E$ a relation specific bias term. Note, the bias terms are also defined as non-negative vectors. In the next paragraph we discuss the importance of this constraint to inject the chain of implications.

Inequality Constraint. The GetBaseValue() and GetSumBias() function in Algorithm 4.2 on Lines 3 and 9 are essential to enforce the inequality constraints. Figure 4.3

presents the mechanics of those functions. Before explaining the functions in detail, we need to discuss the relation embeddings and the relation bias.

To begin with, for each relation r we define a standard relation embedding $\mathbf{r}$ and a non-negative relation bias $\mathbf{b}_r^E$. Assume a set of implication rules Σ is given. An example of a given set Σ and the inferred chain of implications is shown in Figure 4.3 (a). Whenever a relation is present on the right-hand side of an implication rule in the set Σ, we learn a relation bias for this relation instead of the relation embedding. Whenever a relation is only present on the left-hand side of an implication rule in the set Σ (or not contained in any rule), the actual relation embedding for this relation is learned instead of the relation bias. Figure 4.3 (b) demonstrates this for a 2 dimensional embedding. For relation r_1 the relation embedding is learned (initial embedding value $\begin{bmatrix}1\\2\end{bmatrix}$), for relation r_2 and r_3 the relation biases are learned. Lines 2 and 8 show this check whether the relation is present on the right-hand side of an implication or not. If it is not, the algorithm proceeds with the standard relation embedding (e.g. r_1 in the case of Figure 4.3). The bias term should model how relations in the chain of implications differ from each other.

To discuss the use of these bias terms and the actual relation embeddings we need to introduce the two functions from Algorithm 4.2 in more detail: GetBaseValue() (on Line 3) and GetSumOfBias() (on Line 9).

GetBaseValue() Function. For simplicity, we will use the notation of the chain of implications $r_1 \rightarrow r_2 \rightarrow r_3$ of Figure 4.3 instead of the set Σ to explain this function. This function basically retrieves the relation embedding of the first relation of the chain of implications (in Figure 4.3 (c) this represents r_1). We always learn a relation embedding for the first relation since it is only present on the left-hand side of any implication rule. If a relation is present on the right-hand side (check on Line 2) the algorithm assigns the "base" value (GetBaseValue() function) to the relation.

GetSumOfBias() Function. The GetSumOfBias() function (shown in Figure 4.3 (d)) takes as input a specific relation. It returns the following sum: $\forall i > 1 : \sum_{k=1}^{i} \mathbf{b}_k^E$, assuming the following chain of implications is given:

$$\boxed{r_1(x,y) \rightarrow r_2(x,y) \rightarrow ... \rightarrow r_{i-1}(x,y) \rightarrow r_i(x,y)}$$

Figure 4.3 (d) shows this computation for relation r_3. Since the bias values are defined as non-negative values we enforce the desired elementwise inequality: $\forall i > 1$: GetSumOfBias(r_i) $\leq$ GetSumOfBias(r_{i+1}). Again, we have to check whether the relation is present on the right-hand side of an implication on Line 8. If it is present, we retrieve the value of the function GetSumOfBias(r) and add it to the translation result on Line 9 otherwise the algorithm continues with the standard relation embedding.

The following explanation demonstrates how the inequality constraint is essential to inject the implication rule:

For example, if we would call GetSumOfBias(r_2) from Figure 4.3 we would receive the vector $\begin{bmatrix}3\\4\end{bmatrix}$ and for GetSumOfBias(r_3) the vector $\begin{bmatrix}8\\10\end{bmatrix}$: $\begin{bmatrix}3\\4\end{bmatrix} < \begin{bmatrix}8\\10\end{bmatrix}$. In the case of r_3 we would add a larger vector on Line 9, so we would receive a bigger value for $\mathbf{hr}^E$. In combination with the non-negativity constraint (Line 7 and Line 11) this results in a higher inner product value on Line 11. This enforces the following property of the scoring function $s(\mathbf{h}, \mathbf{r_3}, \mathbf{t}) \geq s(\mathbf{h}, \mathbf{r_2}, \mathbf{t})$ for all $h, t \in \mathcal{E}$, given the implication rule: $r_2(h, t) \rightarrow r_3(h, t)$. Whenever the left-hand side of the implication is true and receives a high score, the right hand-side gets always an equal or bigger score and in turn this enforces the implication as we have discussed for the SimplE$^+$ [FRP18] and FSL [DRR16] model.

We have covered all the necessary parts of our proposed algorithm. The following paragraphs discuss the final embedding model and some implementation details.

Final Embedding Model. To inject recursive logical rules in our proposed hyperbolic KGE model, we follow the same steps as in Section 4.2. First, we need a recursive rule (of the form $r(x, y), r(y, x) \rightarrow r(x, z)$) we want to inject for the given KG. Afterwards, we transform this rule into a set of implication rules (that represent a chain of implications) using Algorithm 4.1. To inject these rules, we apply the previously discussed steps to inject implication rules.

Implementation. To implement our proposed hyperbolic approach, we used the codebase of [CWJ$^+$20] on GitHub[2]. We extended the codebase with functionalities of our proposed approach including Algorithm 4.1 and Algorithm 4.2. Our code is available on GitHub[3]. The code is written in Python and we mainly used the PyTorch [PGM$^+$19] package and Pandas [pdt20] to implement the embedding models.

4.4 Summary

This chapter presented our two proposed models to inject recursive logical rules. In Section 4.1 we introduced the general idea of injecting recursive rules. Our idea tracks the recursion depth and generates a set of implication rules for the tracked depths. Furthermore, the goal of our two proposed models is to inject this set of implication rules. In Section 4.2 we presented our first model that is based on the SimplE$^+$ [FRP18] model. This approach uses the same idea to inject implication rules as SimplE$^+$. In Section 4.3 we presented our hyperbolic model. This model moves the general ideas of SimplE$^+$ and FSL [DRR16] to inject implication rules from Euclidean space to hyperbolic space. As a result, this model is able to inject a chain of implications that represent recursive rules in our approach.

[2]https://github.com/HazyResearch/KGEmb
[3]https://github.com/kglab-tuwien/recursive-rule-injection

CHAPTER 5

Evaluation

This chapter describes our experimental setup along with the evaluation results. It starts by giving a detailed explanation of the dataset creation process and which datasets were used for the experiments. Then, to explain the creation of the synthetic datasets, we present algorithms used to generate the data. Furthermore, we discuss the metrics and tasks on which we evaluated our method. Finally, the evaluation results and a discussion is presented.

5.1 Creation of the Datasets

The following section presents the method we use to create the datasets. We want to investigate how the injection of recursive rules affects the performance of the proposed embedding models. All our datasets represent company-subsidiary KGs. In the beginning, the datasets consist of entities representing companies and one relation: *hasSubsidiary*. Afterwards, we apply logical reasoning on the whole dataset using the following rule: $hasSubsidiary(x, z), hasSubsidiary(z, y) \rightarrow hasSubsidiary(z, y)$ as we have presented in Example 5. We will denote the set of *newly* inferred facts as $\mathcal{F}^{new}$. Additionally, we track the recursion depth of the recursive rule as we have discussed in Section 4.1, add the newly introduced relations and create a new set of facts with the new relations. We denote this set as $\mathcal{F}^{new_d}$. It is important to mention that we add the newly derived facts from directly applying the recursive rule **and** the facts derived from tracking the recursion depth to the KG:

$$\mathcal{F} \cup \mathcal{F}^{new} \cup \mathcal{F}^{new_d}$$

We will use *hasSubsidiary_add* as relation for the newly inferred facts in set $\mathcal{F}^{new}$ instead of *hasSubsidiary* to differentiate between the sets $\mathcal{F}^{new}$ and $\mathcal{F}$ only in the textual representation of this chapter for illustration purposes. Note that the actual experiments are executed without differentiation. Therefore, a fact $hasSubsidiary(X, Y)$ from $\mathcal{F}^{new}$ is denoted as $hasSubsidiary\_add(X, Y)$.

After adding these facts, Algorithm 4.1 is applied to create the set of implication rules we aim to inject. To better illustrate the creation of the datasets, Listing A.1 in Appendix A shows all steps, inferred facts and implication rules on a subgraph (including all entities that represent companies) of the Running Example 2.1.

To evaluate our results we split the facts into training, test and validation sets: $\mathcal{F}_{train}, \mathcal{F}_{test}$, $\mathcal{F}_{validation}$, where $\mathcal{F}_{train} \cup \mathcal{F}_{test} \cup \mathcal{F}_{validation} = \mathcal{F}$,using an 80/10/10 split-ratio. As we see in Figure 5.1, the distribution of relations in the set of facts is heavily skewed. Therefore, we apply a *stratified* sampling method on the relations of the facts when creating the splits to ensure that each split contains proportionally the same number of different relations. For the dataset splitting, we used the scikit-learn library [PVG+11] that provides a splitting function with stratifying option. We set the stratify parameter to the relation, that ensures that the number of triples for each relation type is proportionally the same in the training, test and validation sets. The validation set is used to select the best model from all iterations.

To prevent the leakage of data, we follow the experimental setup from Nickel et al. [NK17] and use the **link prediction** setting to evaluate our proposed models. In Section 5.3 the metrics and the setting are discussed in more detail. Nickel et al. evaluate their model on the *transitive closure* of the WordNet dataset. For this reason, the authors first compute the transitive closure of the dataset. This procedure is similar to our processing step since we basically also compute the transitive closure of the KG first. After computing the transitive closure, the authors split the data into train, validation and test sets by randomly holding out observed links. We follow this procedure to first process the dataset as described and, as the next step, partition our dataset randomly into pairwise disjoint training, validation and test sets.

Furthermore, this thesis investigates how recursive rule injection influences the performance of the embeddings. Therefore, we also follow the experimental setup to evaluate the effectiveness of incorporating background knowledge into KGEs (used in the SimplE [KP18] and SimplE$^+$ [FRP18] papers). If background knowledge is given, we expect the KG not to include this additional knowledge since it can be inferred from it. Furthermore, this is the general idea of injecting additional knowledge into embeddings so that methods that do not have this background knowledge can never learn these facts. After processing and splitting the data into train, validation and test sets as described above, we remove all redundant facts (that can be inferred from the given recursive logical rule: *hasSubsidiary_add* relations) from the training set (as in [KP18, FRP18]). This procedure enables us to evaluate the hypothesis that providing recursive background knowledge has an effect on the performance compared to not providing any background knowledge.

5.2 Datasets

This section describes the datasets used for training and evaluating our two proposed approaches. First, the retrieval of the DBPedia dataset is discussed. Afterwards, the

Table 5.1: Statistics for each dataset before and after processing

Dataset	Before Processing	#Entities	#Relations	#Facts
DBPedia	✗	10513	1	8506
DBPedia	✓	10513	7	11706
Synthetic Max. Depth: 3	✗	7163	1	6448
Synthetic Max. Depth: 3	✓	7163	4	22017
Synthetic Max. Depth: 5	✗	8788	1	8088
Synthetic Max. Depth: 5	✓	8788	6	38080
Synthetic Max. Depth: 7	✗	9783	1	9083
Synthetic Max. Depth: 7	✓	9783	9	52575

synthetic data generator is presented in detail, including the algorithms used to implement the data generator. In Table 5.1 the statistics (number of entities, relations and facts) before and after applying processing steps are summarized. The synthetic data generator is discussed in more detail in the subsection about the synthetic data, however maximum depth in Table 5.1 indicates the maximum depth of the randomly generated trees. It is essential to mention that usually, a maximum depth of x should imply x number of relations after processing since the recursion depth is limited to the maximum depth of the tree. However, since the data contains cycles and self-loops, more relations than x are introduced.

5.2.1 DBPedia Dataset

This section discusses the characteristics of the DBPedia dataset. The dataset was queried from a SPARQL endpoint of DBPedia[1] using the following query:

```
SELECT (str(?name1) AS ?company1)
       ("hasSubsidiary" AS ?relation)
       (str(?name2) AS ?company2)
WHERE {
    ?comp <http://dbpedia.org/ontology/subsidiary> ?compSub.
    }
```

Listing 5.1: SPARQL query for company subsidiary dataset from DBPedia

[1] https://dbpedia.org/sparql/, Access: 27 July, 2021

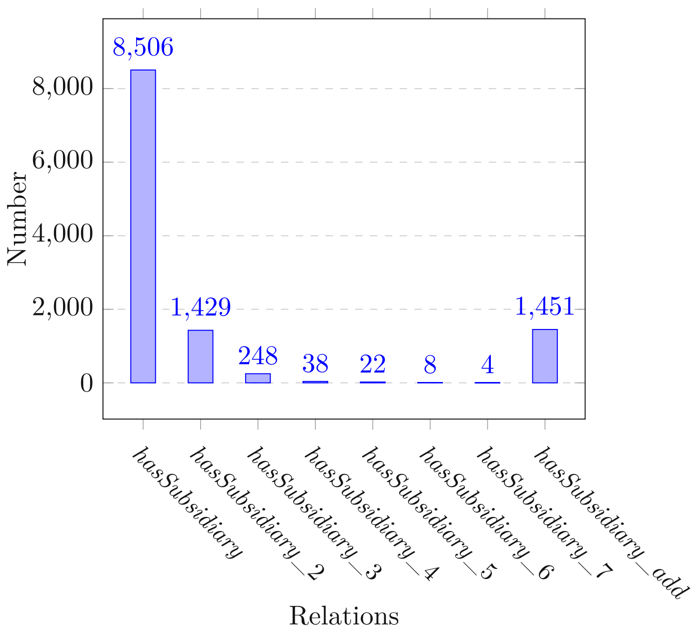


Figure 5.1: DBPedia dataset statistics: frequency of relations in the facts

Table 5.2: DBPedia dataset number of parents per company

Number of Parents	Percentage
1	94%
2	5%
3	$> 1\%$
4	$> 1\%$
5	$> 1\%$
6	$> 1\%$
7	$> 1\%$
8	$> 1\%$
9	$> 1\%$

The query retrieves all companies that are in a subsidiary relation with another company from DBPedia. The Unified Resource Identifiers (URIs) of the company entities are returned. As already mentioned above, Table 5.1 summarizes the statistics of the DBPedia dataset before and after processing (discussed in Section 5.1). A more detailed overview of the number of facts having a specific relation is shown in Bar Plot 5.1. It shows the frequency of facts that contain a specific relation.

Another interesting characteristic is shown in Table 5.2, where we can see that most companies (94%) only have one parent node. Therefore, we can state that the dataset has a tree-like structure and therefore is hierarchical.

5.2.2 Synthetic Dataset

Besides the DBPedia dataset, we implemented a random company-subsidiary generator. This program generates random company-subsidiary graphs. In the following, we discuss the implementation details, assumptions and design decisions. As we have seen in Figure 5.1, the distribution of the relations in the set of facts is heavily skewed. These skewed distributions should serve as a reference point, and therefore, we also use skewed distributions for our random generator. However, we also want to generate bigger datasets than the DBPedia dataset to evaluate our approach on different characteristics. Therefore, we change these skewed distributions accordingly.

The general procedure of the data generator starts by randomly generating hierarchical company-subsidiary trees. As we have discussed in Section 2.1, companies can have multiple parent nodes. Therefore, the next step after creating the trees is to merge these trees on randomly selected nodes by introducing new edges. The last step is to insert edges in order to introduce cycles and self-loops randomly. This procedure consists of the following steps:

- Generate n number of trees.
- Merge m number of trees, in order to combine them and avoid only having single trees.
- Add c number of cycles or self-loops.

This procedure allows us to have more control over the generation process (e.g., regulating the maximum depth of trees that affects the recursion depth) compared to randomly generating nodes and edges. The following paragraphs discuss these three steps in more detail.

Tree Generation. As already discussed, the generator starts by randomly generating trees. Algorithm 5.1 presents the steps to generate the random trees. The algorithm takes as input the maximum depth of the trees and the number of trees n that should be generated. First, the algorithm generates a node that represents the root node on Line 3. The next step is to pick a random number of children for the root node on Line 4. The maximum number of children for the root node is nine, and the skewed probability distribution to select the number of children is shown in Figure 5.2a. The reason why we decided to have a maximum number of nine children and utilize the previously mentioned distribution, is because the percentage of nodes having more than nine nodes drops below 1% on the DBPedia dataset. The distribution of the DBPedia dataset is shown in Figure 5.2c (this figure only shows the number of nodes up until ten children per node). However, to not precisely mimic the DBPedia dataset and to test our approaches on larger datasets, we introduce a different distribution in the following levels of the tree, which is discussed in the following.

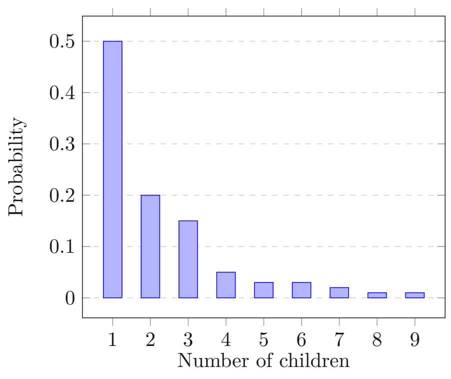
Algorithm 5.1: Generating multiple random trees

```
   Input: Maximum depth $max_{depth}$, number of trees to generate $n$
   Output: A list of randomly generated trees
 1 Initialize trees ← empty list;
 2 for $i \leftarrow 1$ to $n$ do
 3     root ← Generate node;
       // Different distributions for root and root's children
 4     $r \leftarrow$ Randomly pick number from skewed distribution $\in [1, 9]$;
 5     Create $r$ number of nodes and add it to root's children list;
 6     Initialize $q \leftarrow$ Queue with root's children in it;
 7     while q not empty do
 8         cur_node ← Pop element from $q$;
 9         if cur_node.getLevel() $\geq max_{depth}$ then
10             break ; // Reached maximum depth
11         end
12         $rc \leftarrow$ Randomly pick number from skewed distribution $\in [0, 9]$;
13         Create $rc$ number of nodes add it to cur_node's children list and $q$;
14     end
15     trees ← trees ∪ root;
16 end
17 return trees
```

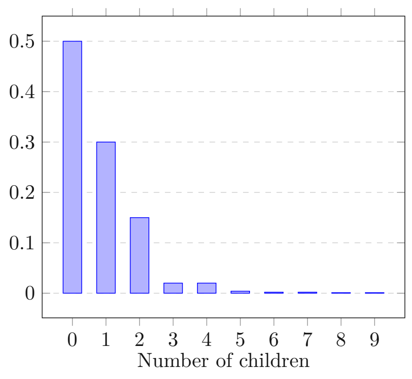
After creating the selected number of children and adding them to the children's list of the parent on Lines 4 and 5, the algorithm initializes a queue. Then, it adds the newly created nodes to it (Line 6). This queue is used to sequentially add a random number of children nodes to the queue. The algorithm starts by popping the first node of the queue (Line 8) and checking if the maximum depth is already reached (Line 9); if so, the algorithm stops. Since the nodes are inserted at the end of the queue level by level, we know that whenever the first node of the queue has a depth greater or equal to the maximum depth, we need to stop the generation process. After popping the first node, the algorithm generates a random number of children for this node (Line 12) and adds it to the node's list of children **and** to the end of the queue (Line 13). The distribution to generate a random number of children is shown in Figure 5.2b.

The distribution of numbers of children per company from the DBPedia dataset is shown in the Bar Plot 5.2d. Note that there are also companies that do not have any subsidiaries[2]. As already mentioned, we do not want to exactly mimic the characteristics of the DBPedia dataset and generate larger datasets to evaluate our models. Therefore, we modified the Skewed Distribution 5.2b of companies per node in comparison to the

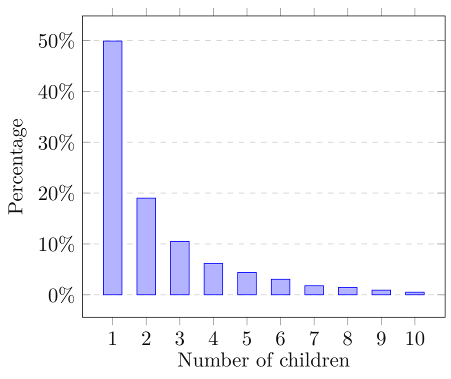
[2]It is important to differentiate between the distribution of the root node (Figure 5.2a) and the distribution of the following levels (Figure 5.2b) since the data generator can not just generate a root node without any subsidiaries. Only generating a root node without any connections would not produce a triple for our dataset. Therefore, the distribution for the root node (Figure 5.2a) excludes 0.



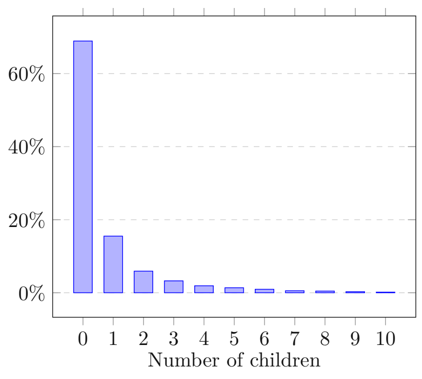
(a) Distribution: Number of children per root



(b) Distribution: Number of children per root nodes except root



(c) DBPedia-Dataset: Number of children per company (excluding companies that do not have any subsidiaries)



(d) DBPedia-Dataset: Number of children per company

DBPedia Distribution 5.2d. We decreased the probability of having no subsidiaries by 18% and increased the probability of having 1 or 2 subsidiaries by 15% and 10%, respectively. This modification should increase the size of the dataset, and it indeed does. Table 5.1 shows that our three synthetic datasets, generated by the data generator, have 1.8, 3.3 and 4.5 times more triples than the DBPedia dataset (after processing).

The algorithm repeats this process of popping a node from the beginning of the list until the queue is empty or we reach the maximum depth. Then, lastly, the root node of this tree is added to the list of trees (Line 15), and the whole procedure is executed n times.

Merge Trees. This paragraph discusses the procedure to combine/merge single trees. Algorithm 5.2 presents the procedure. The algorithm takes as input the maximum depth, a list of generated trees and the number of trees to merge. The maximum depth is needed to avoid merging two trees on nodes that would result in a combined tree exceeding the maximum depth. This parameter allows us to have more control over the random generator and test whether larger recursion depths influence the performance of the

Algorithm 5.2: Merging random trees

```
Input: Maximum depth $max_{depth}$, List of generated trees, number of trees to
       merge $m$
Output: Random subsidiary KG including combined trees
1  $n \leftarrow$ Number of trees;
   /* seq is used to randomly choose which trees to merge.
      The two neighbouring indices are always merged.
      Indices $\in [0, n-1]$.                                        */
2  seq $\leftarrow$ Randomly generate sequence of indices of length $m$;
   // Merge trees
3  for $i \leftarrow 0$ to $m$ by $2$ do
       // Select random node from a tree using indices seq
4      parent_node $\leftarrow$ Randomly pick node from tree: trees[seq[$i$]];
5      Loop
           // Select random node from the next tree in the seq
6          child_node $\leftarrow$ Randomly pick node from tree: trees[seq[$i+1$]];
           // Current height of the merged tree
7          $cr \leftarrow$ GetRemainingHeight(child_node) + GetLevel(parent_node) + 1;
8          if $cr \leq max_{depth}$ then
9              Add child_node to parent_node's children list ;
10             break ; // Success: Did not exceed maximum depth
11         end
12     EndLoop
13 end
```

embedding. Figure 5.3 demonstrates the merging process of Tree A and Tree B by introducing a new relation from node A to node B. The algorithm starts by generating a random sequence of indices of the input list of trees (Line 2). This sequence is used to determine which trees to merge. Our algorithm always merges two neighbouring trees from the sequence. The following example should give a better insight:

Example 8. *Assuming 4 trees are generated, and we want to merge 4 trees. Therefore, we would have a list of 4 trees: [tree0, tree1, tree2, tree3]. The generator would produce a sequence of the form [0,3,2,1], indicating which trees to merge. Firstly, the first tree is merged with the third tree, and the second tree is merged with the first tree (based on the generated indices of the sequence).*

To start this merging process, the algorithm iterates over the sequence of indices on Line 3. Assuming we still have the sequence [0,3,2,1] from the example above, the algorithm picks the tree with the index 0 and selects a random node from this tree (Line 4). This node represents the parent node. Afterwards, a random node from the next tree with index 3 (the next index from the sequence) is chosen. This node represents the child

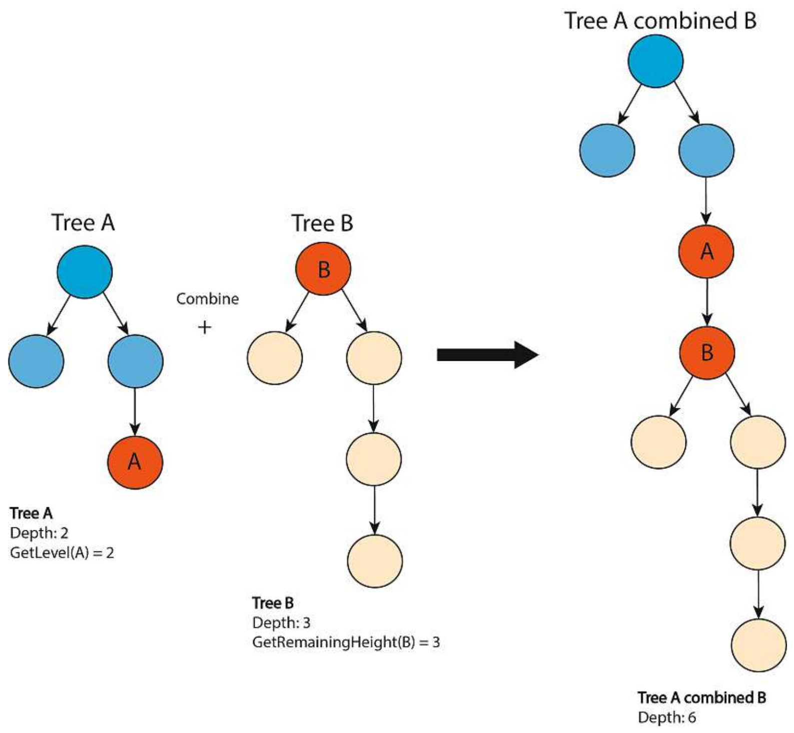


Figure 5.3: Combine Tree A with Tree B. The GetLevel() function returns the level of a specific node, in this case of node A. The GetRemainingHeight() function returns the maximum remaining height of a specific node. In this case of the node B. Tree A gets combined with Tree B by introducing a relation between node A and node B.

node. To not exceed the maximum depth limit, we compute the height of the resulting tree, assuming we would introduce a new relation from the parent node to the child node on Line 7. The function GetRemainingHeight() computes the remaining height of all children of a specific node and selects the maximum out of them. Furthermore, the function GetLevel() returns the level on which a specific node is located in the tree. In Figure 5.3 these two functions are illustrated. Moreover, in this figure, we can see that if we had a maximum depth of 5, we would exceed this limit if we introduce a relation from node A to node B. This check is done on Line 8. Since there is no guarantee that we can combine the two nodes without exceeding the maximum depth, we loop over the random child selection process until we do not exceed it (Line 5).

Afterwards, if the check on Line 8 is successful, we add a relation (*hasSubsidiary*) from the parent node to the child node and successfully merge the two trees. The algorithm would continue by combining the tree with index 2 and index 1 in our small example.

Adding Cycles and Self-Loops. The last step is to add cycles and self-loops. Algorithm 5.3 presents the procedure. The algorithm takes as input the parameters: maximum depth, a list of already merged trees and the number of cycles/self-loops to insert. Note

Algorithm 5.3: Adding cycles and self-loops to trees

```
  Input: Maximum depth $max_{depth}$, List of generated combined trees, number of
         cycles or self-loops $c$
  Output: Random subsidiary KG including loops
  // Insert cycles and self loops
1 for $i \leftarrow 0$ to $c$ do
      // Cycles are of length > 1
2     Randomly decide whether self-loop or cycles;
3     if self-loop then
4         rand_node ← Select random node from random tree from trees;
5         Add edge from rand_node to rand_node;
6     else if cycle then
7         rand_path ← Select random path from random tree of random length ∈
            $[2, max_{depth}]$;
8         Add edge from last node to first node of the path ; // This
            introduces a cycle
9 end
```

Table 5.3: Parameters to create three synthetic datasets with varying maximum depth parameter.

Maximum depth	Number of trees	Number of trees to merge	Number of cycles/self-loops
3	1000	400	100
5	1000	400	100
7	1000	400	100

that the generated trees are actually no “real” trees anymore (since there are children with multiple parents and multiple roots), but we still refer to them as trees for simplicity. The maximum depth is needed to know the maximum length of the cycle and select paths of random length. First, the algorithm decides to either introduce a self-loop or a cycle (a cycle has a length > 1) using a probability of 50% (Line 2). If self-loop is selected, a *hasSubsidiary* relation between a random node to itself is added (on Lines 3 to 5). To introduce a cycle, we first select a random tree and from this tree a random path of random length $1 < n \leq max_{depth}$. Afterwards, we add a *hasSubsidiary* relation between the end node and the path's start node. These steps are done on Lines 6 to 8. This procedure adds cycles and self-loops to the KG.

Parameters and Datasets. In order to evaluate whether the recursion depth has an influence on the embedding's performance, we create three datasets with different maximum depth parameters $\in [3, 5, 7]$ using the algorithms described above. Table 5.3 summarizes the parameters we used to generate the datasets.

The maximum depth parameter also increases the recursion depth of our proposed methods. Therefore, this parameter supports the control of the data generation process itself and provides regulations for our recursion depth approach. As already mentioned, Table 5.1 summarized the statistics of the three generated datasets (before and after processing.)

5.3 Metrics

This section discusses the metrics and the approach to evaluate our proposed embedding models. We evaluate our approaches on the link prediction task. Additionally, for each KG we use the standard data augmentation process that was proposed in [LUO18] and also used in [CWJ+20]. This process adds for each training triple (h, r, t) its inverse (t, r^{-1}, h) to the training set. Furthermore, at test time the queries of the form $(?, r, t)$ are answered as $(t, r^{-1}, ?)$. In order to measure the performance of our proposed models we compute for each test triple (h, r, t) the score (h, r, t') for all $t' \in \mathcal{E}$, rank the entities and compute the ranking $rank_t$ of the triple having t. Additionally, we do the same for (t, r^{-1}, h') and compute the ranking $rank_h$ of the triple having h. We use the two standard evaluation ranking-based metrics from KGE literature: (i) MRR and (ii) H@K, where $K \in \{1, 3, 10\}$. MRR measures the mean of the inverse ranks assigned to correct entities and is defined as follows:

$$\text{MRR} = \frac{1}{2|\mathcal{F}_{train}|} \sum_{(h,r,t)\in\mathcal{F}_{train}} \frac{1}{rank_h} + \frac{1}{rank_t}$$

Furthermore, H@K computes the proportion of correct triples whose ranking is among top K predicted triples [KP18].

In [BUGD+13], the authors identified a problem with computing the metrics mentioned above. We demonstrate this issue by providing an example:

Example 9. *Suppose we are given the following training and test sets (for simplicity, the validation set is left out). This example is a subgraph from our Running Example 2.1:*

$$\begin{aligned}\mathcal{F}_{train} = \{&(MarvelStudios, hasSubsidiary, MarvelCinematicUniverse),\\ &(MarvelStudios, hasSubsidiary, MarvelAnimation),\\ &(MarvelEntertainment, hasSubsidiary, MarvelStudios)\end{aligned}$$

$$\mathcal{F}_{test} = \{(MarvelStudios, hasSubsidiary, MarvelStudios)\}$$

The test triple gets converted to the query $(MarvelStudios, hasSubsidiary, ?)$ and we want to predict $(MarvelStudios, hasSubsidiary, MarvelStudios)$. Assume we have two models. The first model ranks $(MarvelStudios, hasSubsidiary, MarvelAnimation)$ first, whereas the second model ranks $(MarvelStudios, hasSubsidiary, MarvelEntertainment)$ first. We can *see that the first model predicts a correct triple (it is included in $\mathcal{F}_{train}$), in contrast to the second model that predicts an incorrect triple. However, H@1 would assign the same score for both models for the test triple, although the first model should receive a higher score.*

The above example shows that these metrics penalize certain correct predictions. In order to avoid this issue, the authors of [BUGD+13] proposed a modification, *filtered MRR and filtered H@K*. Instead of finding the rank for the test triple (h, r, t) among the triples (h, r, t') for all $t' \in \mathcal{E}$, the modification only considers triples (h, r, t'), where $t' \notin \mathcal{F}_{train} \cup \mathcal{F}_{test} \cup \mathcal{F}_{validation}$, for ranking. For the triples (t, r^{-1}, h') the same protocol is applied. These triples are filtered out during test time and therefore the name *filtered MRR* and *filtered H@K*. As a result, for this thesis we use this filtered setting proposed in [BUGD+13].

5.4 Evaluation Results

In this section, we present our experiments. We evaluated our two proposed methods, SimplE$^+$ adaption and the hyperbolic model (we refer to this model as *HRec*), on the four presented datasets: DBPedia and synthetic dataset with maximum depth $n \in \{3, 5, 7\}$. For each model, we evaluated the results two times: (i) with the injection of recursive rules (as described in the previous chapter) (ii) without injection of logical rules. Using this setup, we can evaluate whether the method of injecting recursive rules does increase the performance. Furthermore, we evaluated the state-of-the-art hyperbolic model AttH [CWJ+20] on the four data sets to have reference values and a baseline. Additionally, SimplE$^+$ without injection also represents a state-of-the-art approach.

Hyper-Parameters. In Table A.1 we summarize the hyper-parameters used during the experiments. We set the maximum number of iterations to 500, and the best model of the 500 iterations gets selected by evaluating it on the validation set. Afterwards, it is evaluated on the test set. In [CWJ+20] the authors showed that hyperbolic models yield better results in low dimensions compared to Euclidean models in higher dimensions. As a result, we set the embedding size of HRec and AttH to 32 and the Euclidean models to 100. We tested several hyper-parameter combinations and reported the settings which yielded the best results in Table A.1. An extensive grid search to select the best hyperparameters was not possible due to computational limitations. However, since our goal is to evaluate whether the injection of recursive rules improves the performance, it is not essential to find the best hyperparameters. For the AttH model, we tried every hyper-parameter setting reported in [CWJ+20] and chose the one with the best results.

The following paragraphs discuss the evaluation results in more detail. We use the following notation in the reported tables: Values in bold mark the best results comparing each model with and without injection of recursive rules. Underlined values highlight the best scores from all models on each metric.

Table 5.4: DBPedia dataset evaluation. Results in **bold** mark the better scores comparing each model with or without injection of recursive rules. Best overall results are underlined.

Model	Rule Injection	DBPedia MRR	DBPedia H@1	DBPedia H@3	DBPedia H@10
AttH	✗	.248	.219	.264	.297
SimplE$^+$	✓	**.179**	**.125**	**.214**	**.282**
SimplE$^+$	✗	.172	.117	.212	.263
HRec	✓	**.192**	**.117**	**.241**	**.329**
HRec	✗	.142	.062	.186	.325

Table 5.5: Synthetic dataset evaluation. Results in **bold** mark the better scores comparing each model with or without injection of recursive rules. Best results are underlined.

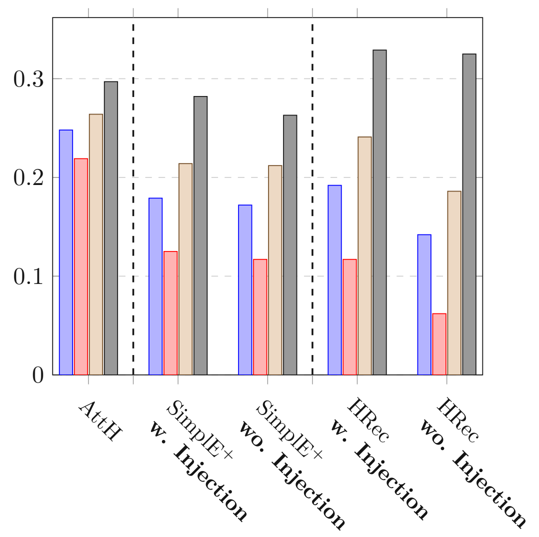
		Synthetic Dataset: Max. Depth: 3				Synthetic Dataset Max. Depth: 5				Synthetic Dataset Max. Depth: 7			
Model	Rule Injection	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
AttH	✗	.507	.467	.540	.571	.540	.514	.558	.579	.530	.482	.556	.625
SimplE$^+$	✓	**.330**	**.183**	**.423**	**.626**	**.308**	**.147**	**.380**	**.668**	**.292**	**.128**	**.354**	**.672**
SimplE$^+$	✗	.304	.172	.397	.547	.283	.125	.374	.619	.270	.112	.341	.622
HRec	✓	**.359**	**.159**	**.471**	.795	**.324**	**.122**	**.411**	**.797**	**.299**	**.112**	**.352**	**.774**
HRec	✗	.260	.066	.283	**.812**	.233	.058	.230	.777	.204	.047	.189	.696

Results and Insights - DBPedia Dataset. Table 5.4 compares the evaluation results of the experiments on the DBPedia dataset. Additionally, Figure 5.4a shows a visual comparison of the results using a bar plot.

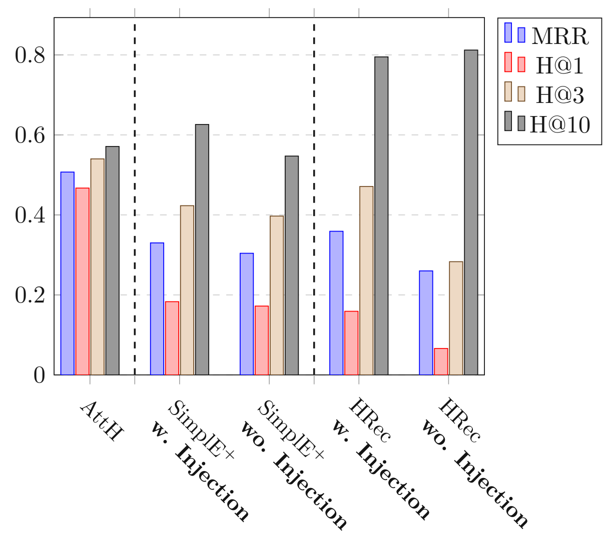
From the results, we can see that the injection of recursive rules outperforms each metric for both SimplE$^+$ and HRec compared to no injection. For the hyperbolic model, the performance gain between injection and no injection is greater than for SimplE$^+$. While the improvement ranges from 0.2% to 1.9% across the metrics for SimplE$^+$, the performance gain for HRec ranges from 0.4% to 5.5%. Furthermore, HRec yields better results compared to SimplE$^+$ (except on the H@1 metric). AttH outperforms SimplE$^+$ and HRec (both with injection) on all metrics except the H@10 metric. For the H@10 metric HRec yields a 3.2% performance gain compared to the state-of-the-art model AttH.

Furthermore, it is essential to mention that the hyperbolic models only have a dimensionality of 32. However, they still outperform the Euclidean model SimplE$^+$ with dimensionality 100. Usually, a higher dimensionality implies more expressive capabilities and, as a result, better results. These advantages demonstrate the effectiveness of using the hyperbolic space.

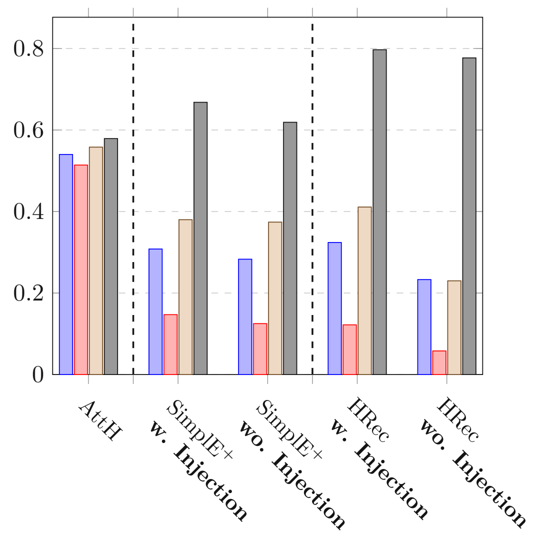
Results and Insights - Synthetic Dataset. The synthetic datasets allows us to not only evaluate the effectiveness of injecting recursive rules but also investigate whether



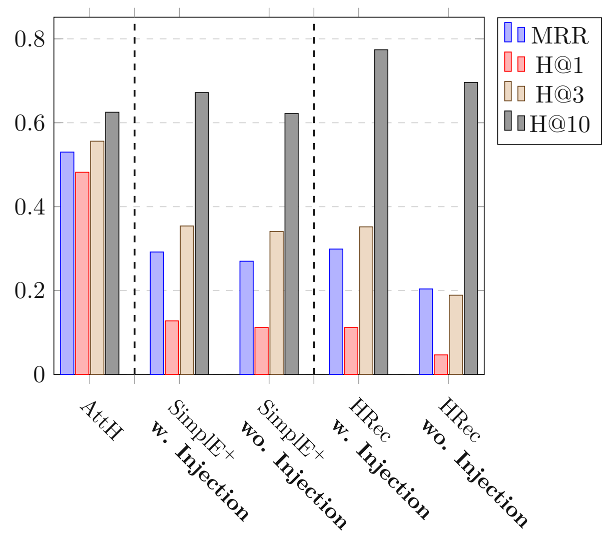
(a) DBPedia evaluation results.



(b) Synthetic dataset (max. depth: 3) evaluation results.



(c) Synthetic dataset (max. depth: 5) evaluation results.



(d) Synthetic dataset (max. depth: 7) evaluation results.

the maximum recursion depth has an impact on the results. As already discussed, we generated synthetic datasets using a maximum depth $d \in \{3, 5, 7\}$. Table 5.5 summarizes the evaluation results on these three datasets. Furthermore, Figure 5.4b shows a bar plot for $d = 3$, Figure 5.4c for $d = 5$ and Figure 5.4d for $d = 7$.

For the synthetic dataset, we get similar insights compared to the DBPedia dataset:

- Injecting recursive rules does yield a performance gain compared to not injecting background knowledge (except for the metric H@10 evaluated on the dataset having maximum depth 3 using the HRec model).
- HRec has a higher performance gain when injecting recursive rules compared to no injection in comparison to SimplE$^+$. HRec performance gain ranges from 2% to 16.3% (excluding the one instance where we got a performance loss), and SimplE$^+$ ranges from 0.6% to 4.9%.
- HRec receives higher scores compared to SimplE$^+$, except on the H@1 metric. However, SimplE$^+$ outperforms HRec on the H@1 metric on every dataset.
- AttH outperforms our proposed models except on the H@10 metric on all datasets. We can see that, SimplE$^+$ and HRec (SimplE$^+$ with injection and HRec with and without injection) outperform AttH on the H@10 metric. For the H@10 metric, we get a performance gain of up to 24%. However, our models perform very poorly on the H@1 metrics.
- It is interesting to mention that HRec has the highest performance gain with the injection of recursive rules compared to no injection on the H@3 metric on all datasets, while for SimplE$^+$ it is the H@10 metric.
- There is no clear pattern to see whether the recursion depth impacts the performance. In general, the evaluation of all three datasets yielded similar results.

To sum up, this confirms our hypothesis that the proposed methods of injecting recursive logical rules yield a performance gain. Furthermore, injecting the recursion depth proves to be a suitable method to inject recursive rules. We could not find any patterns that indicate that the maximum recursion depth influences our proposed models. Our models could not outperform the state-of-the-art hyperbolic AttH model except on the H@10 metric, where our models clearly outperformed AttH.

5.5 Summary

To conclude this chapter, we discussed the different datasets we used for evaluation alongside their statistics and the processing steps. Furthermore, we outlined the experiments we conducted and the metrics used. This chapter also put a focus on the creation of synthetic datasets to evaluate our approach. Lastly, we presented our evaluation results and discussed them. Our proposed method does yield a substantial performance gain on Hit@10 on some datasets of up to 24%. Therefore, we can conclude that injecting recursive rules can greatly improve KGEs.

CHAPTER 6
Conclusion

This thesis aims to investigate methods to combine logical knowledge with ML models. More concretely, we researched incorporating recursive logical rules into KGE. The recursive rules should guide the training of the embedding values to be consistent with the given background knowledge. Some approaches support the injection of logical rules into KGEs. However, no model actively supports recursive background knowledge to the best of our knowledge.

In order to inject recursive rules, we proposed an algorithm and adapted an existing state-of-art KGE model to support the proposed algorithm. Furthermore, besides Euclidean space, we investigated hyperbolic space as representation space. Previously published papers achieved promising results for hyperbolic KGEs. Therefore, a hyperbolic model that supports the injection of recursive logical rules was also proposed. The proposed models have been implemented and evaluated on data from DBPedia and synthetic data.

6.1 Discussion of Research Questions

This section gives an overview of the answers to the proposed research questions in Chapter 1.

Research Question 1 *What is an appropriate method to inject recursive logical rules into KGEs?*

In Chapter 4 we proposed a general method to inject recursive logical rules. Our proposed method transforms recursive rules into a set of implications based on the recursion depths. Additionally, we add a new relation to the KG for each recursion depth. We utilise this method to extend the existing KGE model SimplE$^+$ [FRP18] to support recursive rule injection. Furthermore, we proposed a KGE model in hyperbolic space that supports the injection of implications and, in turn, our proposed method to incorporate recursive logical rules.

Research Question 2.1 *Does the proposed method to inject recursive rules yield better overall performance compared to the same method without an injection?*

In Chapter 5 we presented our experiments and the achieved results. The injection of recursive logical rules does yield a performance gain for every dataset for both proposed models (SimplE$^+$ adaption and HRec). However, HRec performs better in general compared to SimplE$^+$ (except on the H@1 metric). Furthermore, the improvement for injecting recursive rules compared to no injection is higher for HRec in comparison to SimplE$^+$.

Research Question 2.2 *Does the recursion depth affect the performance of the proposed injection method?*

We could not find any patterns that indicate that the recursion depth influences the performance of the proposed method. All different recursion depths result in similar evaluation metrics.

6.2 Future Work

There are several aspects to continue the research conducted during this thesis. On the one hand, there are improvements for the proposed models, and on the other hand, a more detailed investigation and comparison of the models could be conducted. The following listing contains further research:

- One way to improve the models could be to investigate methods to increase the integration of logical reasoning in the training process or even during prediction. For example, to build a hybrid system that can potentially detect that the KGE prediction is not correct and react to this incorrectness by switching to logical reasoning for this specific instance instead.
- A method that does not rely on the information of the recursion depths. We need the recursion depths for our proposed approach, and this knowledge might not be available for specific datasets. Therefore, a method that infers the depths during training would be more favourable.
- Since recursion is, by definition, cyclic, spherical geometry could be more efficient to inject recursive rules. Therefore, further research in this area could develop a spherical geometry model similar to our proposed hyperbolic model and compare both methods.

- Extending the model to support more complex types of recursive rules. Another future research direction would be to extend the model to support multiple recursive logical rules and evaluate them on different datasets. In general, identifying other datasets to evaluate our proposed models could give insights into the models.
- Lastly, an in-depth theoretical analysis of our proposed hyperbolic model is needed. Some further questions would be if the hyperbolic aspect of our proposed method yields the performance gains or if the proposed scoring function is the reason. Another question would be to investigate why the model performs so well on H@10 but on H@1 so poorly.

APPENDIX A

Appendix

Table A.1: Hyperparameter of the models. NA negative samples indicates that no negative samples were used (full cross-entropy loss is used)

Dataset	Embedding Dimension	Model	Rule Injection	Optimizer	Learning Rate	Batch Size	Negative Samples
DBPedia	32	AttH	✗	Adam	.0005	500	50
	100	SimplE+	✓	Adagrad	.1	256	1
	100	SimplE+	✗	Adagrad	.1	256	1
	32	HRec	✓	Adam	.0005	256	NA
	32	Hrec	✗	Adam	.0005	256	NA
Synthetic Max. Depth: 3	32	AttH	✗	Adam	.0005	500	50
	100	SimplE+	✓	Adagrad	.1	256	1
	100	SimplE+	✗	Adagrad	.1	256	1
	32	HRec	✓	Adam	.0005	256	NA
	32	HRec	✗	Adam	.0005	256	NA
Synthetic Max. Depth: 5	32	AttH	✗	Adam	.0005	500	50
	100	SimplE+	✓	Adagrad	.1	256	1
	100	SimplE+	✗	Adagrad	.1	256	1
	32	HRec	✓	Adam	.0005	256	NA
	32	Hrec	✗	Adam	.0005	256	NA
Synthetic Max. Depth: 7	32	AttH	✗	Adam	.0005	500	50
	100	SimplE+	✓	Adagrad	.1	256	1
	100	SimplE+	✗	Adagrad	.1	256	1
	32	HRec	✓	Adam	.0005	256	NA
	32	Hrec	✗	Adam	.0005	256	NA

Listing A.1 gives an overview of the single steps of the dataset creation process for our Running Example 2.1.

```
Complete Set of Facts

Before Recursion
hasSubsidiary(CadenceIndustries, MarvelEntertainment).
hasSubsidiary(MarvelEntertainment, MarvelStudios).
hasSubsidiary(MarvelStudios, MarvelStudios).
hasSubsidiary(MarvelStudios, MarvelCinematicUniverse).
hasSubsidiary(MarvelStudios, MarvelAnimation).
hasSubsidiary(WaltDisneyStudios(division), MarvelStudios).
hasSubsidiary(Renault, Nissan).
hasSubsidiary(Nissan, Renault).

Tracking Recursion Depth: $\mathcal{F}^{new_d}$
hasSubsidiary_2(CadenceIndustries, MarvelStudios).
hasSubsidiary_3(CadenceIndustries, MarvelStudios).
hasSubsidiary_3(CadenceIndustries, MarvelCinemativUniverse).
hasSubsidiary_3(CadenceIndustries, MarvelAnimation).
hasSubsidiary_2(MarvelEntertainment, MarvelStudios).
hasSubsidiary_2(MarvelEntertainment, MarvelCinematicUniverse).
hasSubsidiary_2(MarvelEntertainment, MarvelAnimation).
hasSubsidiary_2(WaltDisneyStudios(division), MarvelStudios).
hasSubsidiary_2(WaltDisneyStudios(division),
                MarvelCinematicUniverse).
hasSubsidiary_2(WaltDisneyStudios(division), MarvelAnimation).
hasSubsidiary_2(Renault, Renault).
hasSubsidiary_2(Nissan, Nissan).

Inferred facts from recursive rule: $\mathcal{F}^{new}$
hasSubsidiary_add(CadenceIndustries, MarvelStudios).
hasSubsidiary_add(CadenceIndustries, MarvelCinemativUniverse).
hasSubsidiary_add(CadenceIndustries, MarvelAnimation).
hasSubsidiary_add(MarvelEntertainment, MarvelCinematicUniverse).
hasSubsidiary_add(MarvelEntertainment, MarvelAnimation).
hasSubsidiary_add(WaltDisneyStudios(division),
                  MarvelCinematicUniverse).
hasSubsidiary_add(WaltDisneyStudios(division), MarvelAnimation).
hasSubsidiary_add(Renault, Renault).
hasSubsidiary_add(Nissan, Nissan).

Implication Rules:
hasSubsidiary_3(X,Y) → hasSubsidiary_2(X,Y).
hasSubsidiary_2(X,Y) → hasSubsidiary(X,Y).
```

Listing A.1: Overview of running example's original facts, newly inferred facts, facts with recursion depth and the resulting implication rules

List of Figures

List of Tables

Listings

List of Algorithms

Acronyms

AI	Artificial Intelligence. xi, 1, 7, 10
CWA	Closed World Assumption. 17
EGD	equality-generating dependency. 13, 14
FOL	first-order logic. 13
H@K	Hits at K. 4, 63, 64
KB	Knowledge Base. 10
KG	Knowledge Graph. ix, xi, 1–3, 5–15, 17, 18, 26, 31, 32, 34, 41–44, 51, 53, 54, 60, 62, 63, 69
KGE	Knowledge Graph Embedding. ix, xi, 1–7, 9–12, 14–17, 26, 29, 32, 33, 35–37, 39, 42, 43, 45, 47, 51, 54, 63, 67, 69, 70
ML	Machine Learning. 1, 5, 7, 23, 26, 69
MRR	Mean Reciprocal Rank. 4, 63, 64
NN	Neural Network. 2, 16
OWA	Open World Assumption. 17
ReLU	rectified linear unit. 36, 37, 45, 49
RSGD	Riemannian Stochastic Gradient Descent. 31, 32
SGD	Stochastic Gradient Descent. 16, 18, 19, 32, 47
TGD	tuple-generating dependency. 13, 14
URI	Unified Resource Identifier. 56

Bibliography

[ACLS20] Ralph Abboud, İsmail İlkan Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. Boxe: A box embedding model for knowledge base completion. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1995.

[AM13] Gabor Angeli and Christopher Manning. Philosophers are mortal: Inferring the truth of unseen facts. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 133–142, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[BAH19a] Ivana Balazevic, Carl Allen, and Timothy M. Hospedales. Tucker: Tensor factorization for knowledge graph completion. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 5184–5193. Association for Computational Linguistics, 2019.

[BAH19b] Ivana Balažević, Carl Allen, and Timothy Hospedales. Multi-relational poincaré graph embeddings, 2019.

[BB01] F. Barca and M. Becht. *The control of corporate Europe*. Oxford University Press, 2001.

[BBC$^+$20] Luigi Bellomarini, Marco Benedetti, Stefano Ceri, Andrea Gentili, Rosario Laurendi, Davide Magnanimi, Markus Nissl, and Emanuel Sallinger. Reasoning on company takeovers during the COVID-19 crisis with knowledge graphs. In Sotiris Moschoyiannis, Paul Fodor, Jan Vanthienen, Daniela Inclezan, Nikolay Nikolov, Francisco Martín-Recuerda, and Ioan Toma, editors, *Proceedings of the 14th International Rule Challenge, 4th Doctoral Consortium, and 6th Industry Track @ RuleML+RR 2020 co-located with 16th Reasoning Web Summer School (RW 2020) 12th DecisionCAMP 2020 as part of Declarative AI 2020, Oslo, Norway (virtual due to Covid-19 pandemic), 29 June - 1 July, 2020*, volume 2644 of *CEUR Workshop Proceedings*, pages 145–156. CEUR-WS.org, 2020.

[BBG+20] Luigi Bellomarini, Marco Benedetti, Andrea Gentili, Rosario Laurendi, Davide Magnanimi, Antonio Muci, and Emanuel Sallinger. Covid-19 and company knowledge graphs: Assessing golden powers and economic impact of selective lockdown via ai reasoning, 2020.

[BFGS19] Luigi Bellomarini, Daniele Fakhoury, Georg Gottlob, and Emanuel Sallinger. Knowledge graphs and enterprise ai: The promise of an enabling technology. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 26–37, 2019.

[BGS18] Luigi Bellomarini, Georg Gottlob, and Emanuel Sallinger. The vadalog system: Datalog-based reasoning for knowledge graphs, 2018.

[Bon13] Silvere Bonnabel. Stochastic gradient descent on riemannian manifolds. *IEEE Transactions on Automatic Control*, 58(9):2217–2229, Sep 2013.

[BRC+20] Federico Bianchi, Gaetano Rossiello, Luca Costabello, Matteo Palmonari, and Pasquale Minervini. Knowledge graph embeddings and explainable AI. In Ilaria Tiddi, Freddy Lécué, and Pascal Hitzler, editors, *Knowledge Graphs for eXplainable Artificial Intelligence: Foundations, Applications and Challenges*, volume 47 of *Studies on the Semantic Web*, pages 49–72. IOS Press, 2020.

[BSV20a] Luigi Bellomarini, Emanuel Sallinger, and Sahar Vahdati. *Chapter 6 Reasoning in Knowledge Graphs: An Embeddings Spotlight*, pages 87–101. Springer International Publishing, Cham, 2020.

[BSV20b] Luigi Bellomarini, Emanuel Sallinger, and Sahar Vahdati. Knowledge graphs: The layered perspective. In Valentina Janev, Damien Graux, Hajira Jabeen, and Emanuel Sallinger, editors, *Knowledge Graphs and Big Data Processing*, volume 12072 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2020.

[BUGD+13] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[CBG20] Calin Cruceru, Gary Bécigneul, and Octavian-Eugen Ganea. Computationally tractable riemannian manifolds for graph embeddings, 2020.

[CCD17] Benjamin Paul Chamberlain, James Clough, and Marc Peter Deisenroth. Neural embeddings of graphs in hyperbolic space, 2017.

[CFG$^+$19] Jack Clearman, Ruslan R. Fayzrakhmanov, Georg Gottlob, Yavor Nenov, Stéphane Reissfelder, Emanuel Sallinger, and Evgeny Sherkhonov. Feature engineering and explainability with vadalog: A recommender systems application. In Mario Alviano and Andreas Pieris, editors, *Datalog 2.0 2019 - 3rd International Workshop on the Resurgence of Datalog in Academia and Industry co-located with the 15th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2019) at the Philadelphia Logic Week 2019, Philadelphia, PA (USA), June 4-5, 2019*, volume 2368 of *CEUR Workshop Proceedings*, pages 39–43. CEUR-WS.org, 2019.

[CWJ$^+$20] Ines Chami, Adva Wolf, Da-Cheng Juan, Frederic Sala, Sujith Ravi, and Christopher Ré. Low-dimensional hyperbolic knowledge graph embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6901–6914, Online, July 2020. Association for Computational Linguistics.

[CYRL19] Ines Chami, Rex Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks, 2019.

[DFO20a] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning.* Cambridge University Press, Cambridge New York, NY Port Melbourne New Delhi Singapore, 2020.

[DFO20b] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning.* Cambridge University Press, 2020.

[DRR16] Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Lifted rule injection for relation embeddings. In Jian Su, Xavier Carreras, and Kevin Duh, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1389–1399. The Association for Computational Linguistics, 2016.

[EH56] Euclides and Thomas L Heath. *The thirteen books of Euclid's elements : 1. Introduction and books I, II.* Dover Publ., New York, NY, 2. ed. rev. with additions. edition, 1956.

[FRP18] Bahare Fatemi, Siamak Ravanbakhsh, and David Poole. Improved knowledge graph embedding using background taxonomic information, 2018.

[GBH18a] Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic entailment cones for learning hierarchical embeddings. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1632–1641. PMLR, 2018.

[GBH18b] Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic neural networks, 2018.

[GHLZ13] Todd J. Green, Shan Shan Huang, Boon Thau Loo, and Wenchao Zhou. 2013.

[GSGR19] Albert Gu, Frederic Sala, Beliz Gunel, and Christopher Ré. Learning mixed-curvature representations in product spaces. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[GWW$^+$16] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Jointly embedding knowledge graphs and logical rules. In Jian Su, Xavier Carreras, and Kevin Duh, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 192–202. The Association for Computational Linguistics, 2016.

[GWW$^+$18] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Knowledge graph embedding with iterative guidance from soft rules. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 4816–4823. AAAI Press, 2018.

[Háj98] Petr Hájek. *Metamathematics of Fuzzy Logic*, volume 4 of *Trends in Logic*. Kluwer, 1998.

[HNA$^+$97] Stephen Hyde, Barry W. Ninham, Sten Andersson, Kåre Larsson, Tomas Landh, Zoltan Blum, and Sven Lidin. Chapter 1 - the mathematics of curvature. In Stephen Hyde, Barry W. Ninham, Sten Andersson, Kåre Larsson, Tomas Landh, Zoltan Blum, and Sven Lidin, editors, *The Language of Shape*, pages 1–42. Elsevier Science B.V., Amsterdam, 1997.

[JLHZ16] Guoliang Ji, Kang Liu, Shizhu He, and Jun Zhao. Knowledge graph completion with adaptive sparse transfer matrix. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference* *on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 985–991. AAAI Press, 2016.

[JPC$^+$21] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*, page 1–21, 2021.

[KP18] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 4289–4300, 2018.

[Lee13] J.M. Lee. *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics. Springer New York, 2013.

[LLS$^+$15] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, pages 2181–2187. AAAI Press, 2015.

[LUO18] Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2869–2878. PMLR, 2018.

[LWY$^+$17] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6738–6746. IEEE Computer Society, 2017.

[MCM$^+$17] Pasquale Minervini, Luca Costabello, Emir Muñoz, Vít Novácek, and Pierre-Yves Vandenbussche. Regularizing knowledge graph embeddings via equivalence and inversion axioms. In Michelangelo Ceci, Jaakko Hollmén, Ljupco Todorovski, Celine Vens, and Saso Dzeroski, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18-22, 2017, Proceedings, Part I*, volume 10534 of *Lecture Notes in Computer Science*, pages 668–683. Springer, 2017.

[MHW$^+$19] Yu Meng, Jiaxin Huang, Guangyuan Wang, Chao Zhang, Honglei Zhuang, Lance M. Kaplan, and Jiawei Han. Spherical text embedding. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8206–8215, 2019.

[MR18] Pasquale Minervini and Sebastian Riedel. Adversarially regularising neural NLI models to integrate logical background knowledge. In Anna Korhonen and Ivan Titov, editors, *Proceedings of the 22nd Conference on Computational Natural Language Learning, CoNLL 2018, Brussels, Belgium, October 31 - November 1, 2018*, pages 65–74. Association for Computational Linguistics, 2018.

[NK17] Maximilian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations, 2017.

[NK18] Maximilian Nickel and Douwe Kiela. Learning continuous hierarchies in the lorentz model of hyperbolic geometry. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3776–3785. PMLR, 2018.

[NMTG16] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, Jan 2016.

[NTK11] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 809–816, Madison, WI, USA, 2011. Omnipress.

[Pau17] Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, 8(3):489–508, 2017.

[pdt20] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[PGM$^+$19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[PVG+11] 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*, 12:2825–2830, 2011.

[RG11] Jürgen Richter-Gebert. *Perspectives on Projective Geometry: A Guided Tour Through Real and Complex Geometry.* Springer Publishing Company, Incorporated, 1st edition, 2011.

[RM51] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, 1951.

[RSR15] Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In Rada Mihalcea, Joyce Yue Chai, and Anoop Sarkar, editors, *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1119–1129. The Association for Computational Linguistics, 2015.

[Sal13] Emanuel Sallinger. Reasoning about Schema Mappings. In Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt, editors, *Data Exchange, Integration, and Streams*, volume 5 of *Dagstuhl Follow-Ups*, pages 97–127. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2013.

[SCMN13] Richard Socher, Danqi Chen, Christopher D. Manning, and Andrew Y. Ng. Reasoning with neural tensor networks for knowledge base completion. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'13, page 926–934, Red Hook, NY, USA, 2013. Curran Associates Inc.

[SDNT19] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[SGB20] Ondrej Skopek, Octavian-Eugen Ganea, and Gary Bécigneul. Mixed-curvature variational autoencoders. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[Sin12] Amit Singhal. Introducing the knowledge graph: things, not strings. `https://blog.google/products/search/introducing-knowledge-graph-things-not/`, 2012. Accessed: 2021-06-15.

[TBG19] Alexandru Tifrea, Gary Bécigneul, and Octavian-Eugen Ganea. Poincare glove: Hyperbolic word embeddings. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[Tuc64] L. R. Tucker. The extension of factor analysis to three-dimensional matrices. In H. Gulliksen and N. Frederiksen, editors, *Contributions to mathematical psychology.*, pages 110–127. Holt, Rinehart and Winston, New York, 1964.

[TWR$^+$16a] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2071–2080. JMLR.org, 2016.

[TWR$^+$16b] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction, 2016.

[Ung01] A.A. Ungar. Hyperbolic trigonometry and its application in the poincaré ball model of hyperbolic geometry. *Computers & Mathematics with Applications*, 41(1):135–147, 2001.

[Vij06] Anand M. Vijh. Does a parent-subsidiary structure enhance financing flexibility? *The Journal of Finance*, 61(3):1337–1360, 2006.

[Wei02] Eric W. Weisstein. Great circle. `https://mathworld.wolfram.com/GreatCircle.html`, 2002. Accessed: 2021-05-18.

[Wei14] Steven H. Weintraub. Chapter 4 - smooth manifolds. In Steven H. Weintraub, editor, *Differential Forms (Second Edition)*, pages 141–206. Academic Press, Boston, second edition edition, 2014.

[Wei21] Eric W. Weisstein. Spherical geometry. `https://mathworld.wolfram.com/SphericalGeometry.html`, 2021. Accessed: 2021-08-05.

[WHPD14] Richard C. Wilson, Edwin R. Hancock, Elzbieta Pekalska, and Robert P. W. Duin. Spherical and hyperbolic embeddings of data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(11):2255–2269, 2014.

[WMWG17] 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*, 29(12):2724–2743, 2017.

[WZFC14] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference* *on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, pages 1112–1119. AAAI Press, 2014.

[YYH+15] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[ZTJ+21] Shuai Zhang, Yi Tay, Wenqi Jiang, Da-Cheng Juan, and Ce Zhang. Switch spaces: Learning product spaces with sparse gating. *CoRR*, abs/2102.08688, 2021.