Fall 2021

HRotatE: Hybrid Relational Rotation Embedding for Knowledge Graph

Akshay Mukundbhai Shah
University of Windsor

HRotatE: Hybrid Relational Rotation Embedding for Knowledge Graph

By

Akshay Mukundbhai Shah

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2021

HRotatE: Hybrid Relational Rotation Embedding for Knowledge Graph

by

Akshay Mukundbhai Shah

APPROVED BY:

M. Mirhassani

Department of Electrical and Computer Engineering

K. Selvarajah

School of Computer Science

Z. Kobti, Advisor

School of Computer Science

12 August 2021

Declaration of Co-Authorship/Previous Publication

1. **Co-Authorship**

   I hereby declare that this thesis incorporates material that is the result of research conducted under the supervision of Dr. Ziad Kobti. In all cases, the key ideas, primary contributions, experimental designs, data analysis, and interpretation were performed by the author, and the contribution of co-authors was primarily through the proofreading of the published manuscripts. Mr. Bonaventure Molokwu contributed to the discussion of concept and publication.

   I am aware of the University of Windsor Senate Policy on Authorship, and I certify that I have properly acknowledged the contribution of other researchers to my thesis and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

   I certify that, with the above qualification, this thesis, and the research to which it refers, is a product of my own work.

2. **Previous Publication**

   This thesis includes the paper that has been accepted in a peer reviewed conference, as follows:

Thesis Chapter	Publication Title/Full Citation	Publication Status
Chapter 3	A. Shah, B. Molokwu, and Z. Kobti. HRotatE: Hybrid Relational Rotation Embeddingfor Knowledge Graph. In 2021 International Joint Conference on Neural Networks, IJCNN 2021. Institute of Electrical and Electronics Engineers (IEEE).	Published

I certify that I have obtained written permission from the copyright owner(s) to include the above-published material(s) in my thesis. I certify that the above material describes work completed during my registration as a graduate student at the University of Windsor.

3. **General**

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained written permission from the copyright owner(s) to include such material(s) in my thesis.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Knowledge Graph (KG) represents the real world's information in the form of triplets (head, relation, and tail). However, most KGs are generated manually or semi-automatically, which resulted in an enormous number of missing information in a KG. The goal of a Knowledge-Graph Completion task is to predict missing links in a given Knowledge Graph. Various approaches exist to predict a missing link in a KG. However, the most prominent approaches are based on tensor factorization and Knowledge-Graph embeddings, such as RotatE and SimplE. The RotatE model depicts each relation as a rotation from the source entity (Head) to the target entity (Tail) via a complex vector space. In RotatE, the head and tail entities are derived from one embedding-generation class, resulting in a relatively low prediction score. SimplE is primarily based on a Canonical Polyadic (CP) decomposition. SimplE enhances the CP approach by adding the inverse relation where head embedding and tail embedding are taken from the different embedding-generation classes, but they are still dependent on each other. However, SimplE is not able to predict composition patterns very well. This paper presents a new, hybridized variant (HRotatE) of the existent RotatE approach. Essentially, HRotatE is hybridized from RotatE and SimplE. We have used the principle of inverse embedding (from the SimplE model) in a bid to improve the prediction scores of HRotatE. Hence, our results have proven to be better than the native RotatE. Also, HRotatE outperforms several state-of-the-art models on different datasets. Conclusively, our proposed approach (HRotatE) is relatively efficient such that it utilizes half the number of training steps required by RotatE, and it generates approximately the same result as RotatE.

Dedication

I would like to dedicate my thesis to my parents Mukundbhai Shah and Nilima Shah, as well as my brother Nishant Shah for their continuous support, love, and care.

Acknowledgements

The time I spent working on these ideas has been one of the most rewarding periods of my life. I will forever be grateful to my advisors Dr. Ziad Kobti who are not only brilliant researchers but amazing human beings. Under his guidance, I had enjoyed a lot working on my research work. He provided me the opportunity to select my research domain and his continuous encouragement and guidance helped me to improve my research work. I learned a lot from his profound understanding of research problems, and I am very grateful for his encouragement. It was always an immense pleasure to work and discuss with him. Without his support, this won't have been possible.

I would also like to thank my thesis committee, Dr. Kalyani Selvarajah, and Dr. Mitra Mirhassani, for their insightful comments and inspiration for the completion of this thesis. I would also like to thank my lab mate Mr. Bonaventure Molokwu for helping me with his insights and his skills in writing papers, and his suggestions. It's a great honor to become a part of the University of Windsor. I would like to thank SHARCNET for providing computational resources. Besides that, I would like to thank the other UWindsor professors who guided me throughout my MSc program in various courses. I would like to express my appreciation to Mrs. Gloria Mensah, Mrs. Melissa Robinet, and Mrs. Christine Weisener who always supported me when I needed assistance in various academic issues.

Mostly, I want to express my gratitude to my parents for supporting me at every stage of my life. Whenever I talk to them, I gain new inspiration and enthusiasm, and any problems don't seem difficult to solve. Furthermore, I would like to thank my brother for fostering the love of learning and discovery and continuously encouraging me to achieve my goals.

Finally, I would like to thank my lab mates, friends, and my roommates for their con-

tinuous support, encouragement, caring. Special thanks to Mr. Manil Patel for intriguing ideas and elaborate discussions. It can be extremely challenging to live away from home, but with the support of my friends, it became a lot easier for me. Words cannot express how grateful I am to my parents, brother, and friends for always supporting me in every situation.

Akshay Mukundbhai Shah

Table of Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence.

CNN Convolutional Neural Network.

CP Canonical Polyadic.

CPU Central Processing Unit.

CSV Comma-separated Values.

DL Deep Learning.

GPU Graphics Processing Unit.

KG Knowledge Graph.

KGC Knowledge Graph Completion.

LP Link Prediction.

MAE Mean Absolute Error.

ML Machine Learning.

MRR Mean Reciprocal Rank.

NLP Natural Language Processing.

RMSE Root Mean Square Error.

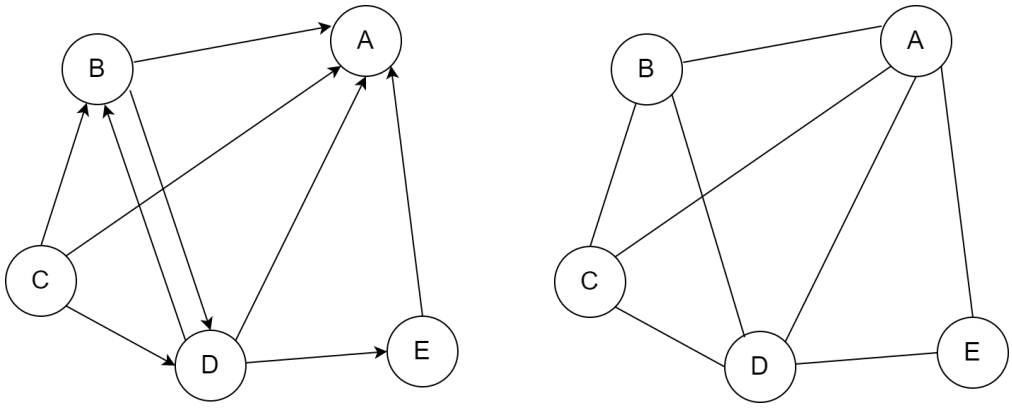
Chapter 1

Introduction

Knowledge Graph (KG) represents the real world's information in the form of triplets (head, relation, and tail). However, most KG are highly incomplete. Knowledge Graph Completion (KGC) aims to automatically infer missing facts by exploiting information already present in a KG. In this thesis, We explore the problem of learning representations of entities and relations in KG for predicting missing links. A promising approach for KGC is embedding-based. This thesis presents a new, hybridized variant (HRotatE) of the existent RotatE and SimplE approach.

1.1 Background

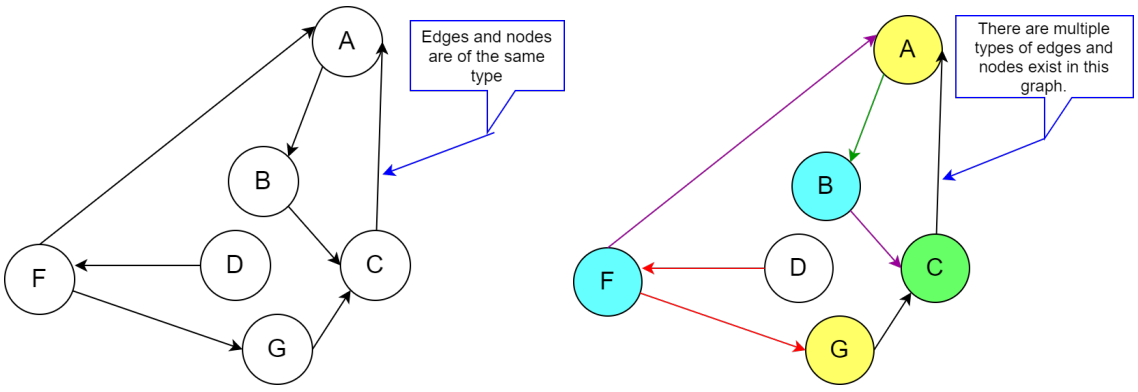
A **graph** [1] is a structure used to represent the information. A graph consists of two sets: nodes (vertices) and edges (line or arcs). Each edge connects to a pair of nodes and can be described as a relation between those nodes. This relation can either be undirected or directed [2]. For example, if a graph is created to illustrate the friendship between two different people, then the edges will be undirected because it represents that two people are friends; on the contrary, if the graph represents how users follow each other, then the graph will be directed. Figure 1.1(a) represent the directed graph. In this case, $edge(B, A) \neq edge(A, B)$. While Figure 1.1(b) represents the undirected graph. In this case, $edge(B, A) = edge(A, B)$.



(a) Directed Graph (b) Undirected Graph

FIGURE 1.1: Example of the directed and undirected graph

Graphs can be either homogeneous or heterogeneous [3]. In a **homogeneous graph**, all the relations represent the same type, and all the nodes also represent the instant of the same type. For example, in a social network of people and their connection, nodes generally represent the people, and edges generally represent the connection between two nodes. In a **heterogeneous graph**, the nodes and edges can be of several types. For example, in a bibliography network, nodes represent the different things like paper, author, institution, etcetera, and edges also represent the different relations like cited by, written by, etcetera. Figure 1.2(a) illustrated the directed homogeneous graph while Figure 1.2(b) illustrates the heterogeneous graph where distinct color represents the distinct type of nodes or edges.



(a) Homogeneous Graph (b) Heterogeneous Graph

FIGURE 1.2: Example of the homogeneous and heterogeneous graph

Multigraph is the graph form that contains multiple edges between the same pair of nodes. These multiple edges indicate the different types of relations. They may contain

the self-loops also. Most of the multigraphs are heterogeneous graphs. **KG** is the directed heterogeneous multi graph where each relation is associate with pair of entities [4].

1.1.1 Knowledge Graph

Sir Tim Berners-Lee introduced the Semantic Web [5], with the aim of providing a common framework for sharing, analyzing, and reusing data across applications, enterprises, and community boundaries. Initially, the Semantic Web aimed to facilitate integration and combination of data collected from different sources by defining standard formats for exchanging data on the web. In recent years, it has become more helpful in describing how data can relate to real-world entities. With the development of the Semantic Web, knowledge graphs have often been associated with linked open data projects, focusing on the relationships between concepts and entities [6, 7].

Knowledge Graph (KG) term was introduced by Google in 2012 [8]. After that, KG term began to be used in many domains frequently. The KG does not have a formal definition. In a broader perspective, KG can be described as very large semantic networks integrating different and heterogeneous information sources to represent a deep understanding of domains of discourse.

In general, KG is a graph variant that holds data in the form of triplets comprising a head, a relation, and a tail (where heads and tails are referred to as subject\source and object\target entities). To refrain from defining a formal definition, we can outline the desired characteristics of a KG.

- Primarily, KG describes real-world information in the form of triplets: (head, relation, and tail).
- The possible classes as well as the relations existing between entities of a KG are defined in a schema.
- A KG permits potential interrelations between arbitrary entities.
- A KG can be employed in a wide variety of topical domains.

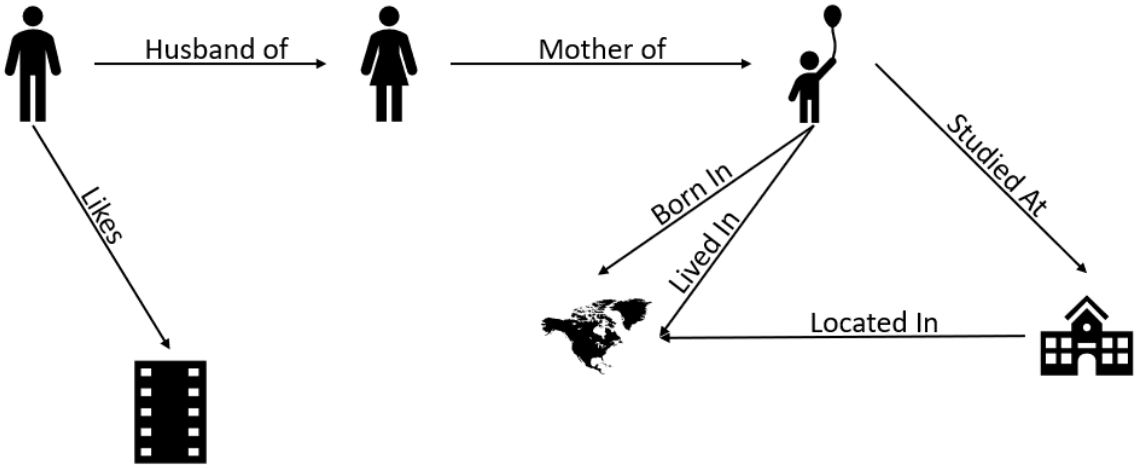


FIGURE 1.3: Example of knowledge graph

Figure 1.3 represents the example of Knowledge Graph, where entities (head and tail) are represented by the icons and relationships are represented by the edges. The fact 'Person likes a movie' can be represented as a triple (Person, likes, Movie).

1.1.2 Knowledge Graph Completion

Since most of the KGs are generated manually or semi-automatically, a large number of implicit entities and relationships may not have been found. Thus incompleteness becomes a problem in almost all KGs [9]. For example, 71% of people do not have a birthplace in the Freebase dataset [10]. Link Prediction (LP) [11, 12] is the problem of predicting the existence of a link between two entities in a network [13]. LP is used in various areas, including social network analysis, recommendation systems, protein-protein network, and many others [14, 15]. LP algorithms can predict these missing links. This problem is widely known as the Knowledge Graph Completion (KGC) problem.

The objective of KGC is to eliminate the problems associated with incompleteness and sparsity of KGs by finding missing instances and connections in order to improve KGs efficiency. KGC completes the graph structure by predicting missing links (entities or relation), and finding new facts and etcetera. This KGC algorithms widely applied in many applications such as question answering, Natural Language Processing (NLP) task [9]. KGC problem can be described as these three kinds of tasks.

- predicts the tail entity from the head entity and their relationships, such as (Delhi, capitalOf, ?);
- predicts the head entity from the tail entity and their relationships, such as (?, capitalOf, India);
- predicts the relationship from the head and tail entity from the triplet, such as (Delhi, ?, India).

Thus, from any two elements given in a triple and the third element, can be predicted. The below Figure 1.4 illustrate the KGC problem. Where based on a ground knowledge shown in Figure 1.3, KGC algorithm tries to predict the missing links.

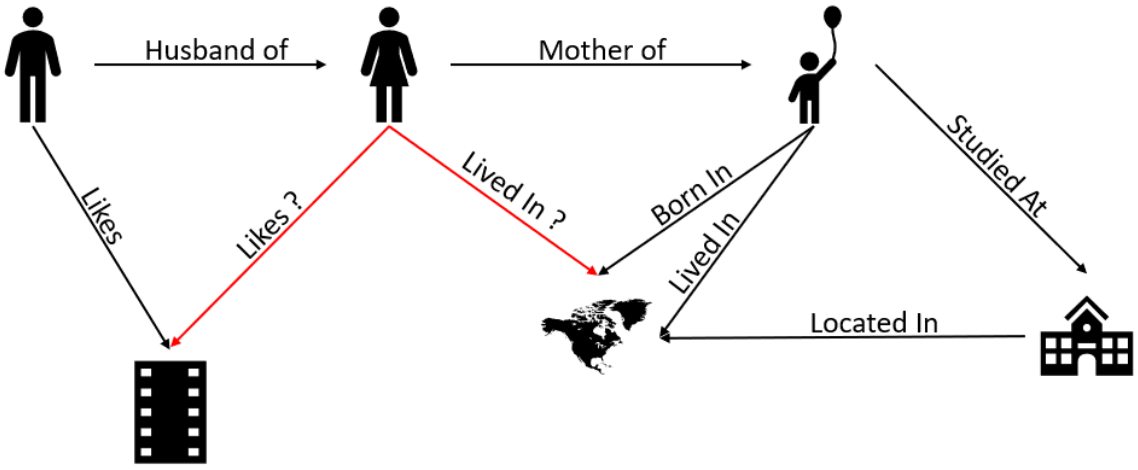


FIGURE 1.4: Example of knowledge graph completion

1.1.3 Knowledge Graph Embedding

Node Embedding maps each node of the graph into low-dimensional embedding space. These Embeddings provide information about the network's node structure and its similarity to other nodes. Generally, similar nodes are embedded nearer to each other. Since there are multiple types of relations exist in KG, the problem of KGC is more complex than the general social network LP problem. Thus, In the KG, every relation is also embedded in embedding tensor.

Figure 1.5 [16] represent the embedding matrix of entities and relations. Here, every entity is embedded in the d dimensions vector. Let M is the embedding matrix of entities.

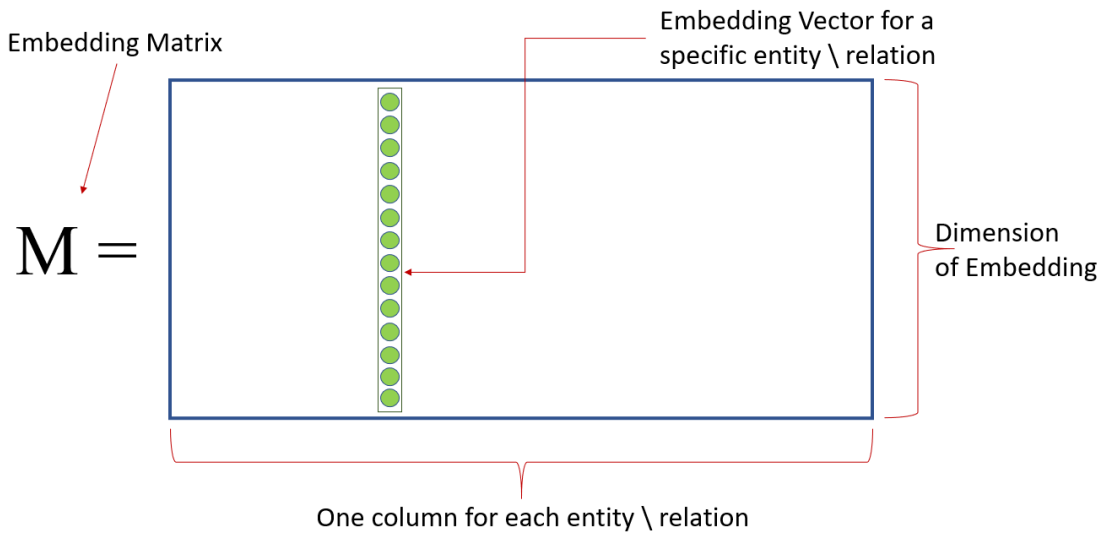


FIGURE 1.5: Illustration of knowledge graph embedding

Then every column in the matrix M indicates the embedding of the entity. The total number of rows is equal to the dimension of the embedding d. The embedding matrix for relation is also calculated in the same way as the entity embedding matrix. In general, the relation embedding is used to map the head entity to the tail entity. Our proposed approach (HRotatE) creates two embedding matrices for entities, one represents the head entity, and another represents the tail entity. For every relation, HRotatE also generates the two embedding matrices: relation embedding matrix and relation inverse embedding matrix. The relation embedding matrix is used to calculate the main score for HRotatE model while the relation inverse matrix is used to calculate the inverse score.

1.2 Problem Definition

KG comprises the sets of entities, $\mathcal{E}$, and relations, $\mathcal{R}$. Every record in KG can be represented via a triple of (h, r, t) such that $h \in \mathcal{E}$ is the head, $r \in \mathcal{R}$ is the relation, and $t \in \mathcal{E}$ is the tail of each triple. ζ represents all the true triplet sets, such as $(Delhi, capitalOf, India)$. ζ' represents the false triplet sets, such as $(Delhi, capitalOf, Japan)$.

Given KG $G = (\mathcal{E}, \mathcal{R}, \mathcal{E})$, where $\mathcal{E}$ represent the entity set and $\mathcal{R}$ represent the relation

set, KGC completes graph G by finding the missing set of triples $T' = \{(h, r, t)|h \in \mathcal{E}, r \in \mathcal{R}, t \in \mathcal{E}, (h, r, t) \notin T\}$ in the incomplete KG [17].

1.3 Thesis Motivation

The KG is basically a semantic network, which is a formal description of things in the real world and their relationships [18]. In general, the KG contains the vast number of entities and their complex and diverse relationship with other entities [9]. Thus, any large-scale KGs includes millions of entities and their relations. That is why any real-world KG are too big. Since most KGs are generated manually or semi-automatically, a large number of implicit entities and relationships are missing. Thus incompleteness becomes a problem in almost all KGs [9]. For example, 71% of people do not have a birthplace in the Freebase dataset [10], 94% have no known parents, and 99% have no known ethnicity [10]. This is considerably basic information because every person has a parent and a place of birth. This missing information affects the efficiency of the KG.

Thus, it is an important goal of general knowledge graphs to integrate more entity-relationship information as comprehensively as possible. This problem is widely known as KGC. This completed KG used in various applications including but not limited to Question Answering, Recommended System, Information Retrieval, and various other domain-specific applications. KGC can be done by various approaches for example KGC based on rule reasoning, KGC based on probabilistic graph model, KGC based on graph calculation, and KGC based on graph embedding and representation learning. Based on recent research, embedding and representation approaches perform better than the others [19].

The embedding-based approach involves learning the embedding for each triplet (head, relation, and tail), after which these learned embeddings are used to resolve prediction-based problems within the KG. The most popular embedding-based approaches are, viz: translation-based approach [20–22], bilinear-based approach [19, 23–25], and neural network-based approach [26–28]. The translation-based approaches use the relative distance existing between constituent embedding vectors of the KG. These approaches are less complex than the bilinear and deep learning (neural-network-based) approaches. Additionally, the bilinear-based approaches employ tensor factorization with regard to the generation of em-

bedding vectors in the KG. The neural-network-based approaches are essentially nonlinear models that vary according to different deep learning architectures.

Each of the approaches possesses its respective strengths and weaknesses. For instance, translation-based approaches are based on sets of logical rules which can not apply to all kinds of properties of KGs. In this regard, earlier translation-based models are not fully expressive. Nowadays, neural-network-based approaches perform well on different datasets, but they are like black boxes. They are not transparent and poorly understood in comparison to other approaches [29].

One of the best translation-based approaches is RotatE [22]. RotatE represents the relation as a rotation between the head entity and the tail entity in the complex vector space. RotatE calculates the score using $\|h \circ r - t\|$. In this way, RotatE can predict symmetric, anti-symmetric, inverse, and composition patterns. In RotatE, the head and tail entities are derived from the same embedding-generation class. However, we cannot treat head and tail entirely differently because both are still entities. CP [30] address both entities entirely independent of each other, and as such, does not perform very well. SimplE[19] addresses this problem by adding the inverse relation where head embedding and tail embedding are taken from the different embedding-generation classes but are still dependent on each other. This research proposes a novel hybridized framework of the existing approach RotatE and SimplE, that uses different embedding generation classes or different embedding spaces for each entity (head and tail). Our proposed approach improves the score of the existing KGC approach RotatE.

1.4 Thesis Statement

The objective of this research is to create a novel hybridized approach that can perform better in the KGC task. We aim to improve the existing KGC algorithm such that it performs better on various datasets and it also runs efficiently.

Our proposed approach (HRotatE) is based on RotatE [22] and SimplE [19]. As our model (HRotatE) is directly derived from the RotatE. It utilizes all the characteristics of RotatE. HRotatE uses two different embedding generation classes to generate head and tail

embedding vectors. That gives the advantage to HRotatE in comparison to the RotatE. However, the head and tail are still part of entities. Thus, by inspired by SimplE, we introduce the concept of inverse relation embedding in the RotatE. This Inverse embedding function helps our model to learn efficiently, and it also improves the performance of our approach.

In this thesis, we solve the problem of KGC which helps to predict the missing information in the KG. The main objective of this thesis is to use different embedding vectors for the head and tail embedding so that the algorithm can understand the KG embedding efficiently. By doing this, we expect to improve the prediction score as well as reduce the training time.

1.5 Thesis Contribution

This thesis addresses the KGC problem and proposes a novel hybridized variant "HRotatE - Hybrid Relational Rotation Embedding for Knowledge Graph" of the existing approach RotatE and SimplE. The proposed approach predicts the missing information in KG by learning the KG embedding. KG is given as the input in the model; the model generates the negative samples, processes the data, and makes predictions. Our key contributions can be found in the following list.

1. We have proposed a new hybrid model for resolving the Knowledge-Graph Completion problems.
2. We have relatively compared our approach (HRotatE) against five (5) popular benchmark datasets.
3. We have shown that our model (HRotatE) can achieve approximately the same accuracy as RotatE with just half the number of training steps required in RotatE.
4. Our model is linear, and it also outperforms several state-of-the-art models upon benchmarking using popular KG datasets.

1.6 Background and Notation

In this thesis, we express matrices using uppercase notation and vectors via lowercase notation. We can access a particular value from the vector by its indices. For example: v_k represents the k^{th} element of vector, v. Also, $\circ$ represents the element-wise multiplication between two vectors; and $\cdot$ denotes the dot product between two vectors. L1 Norm is represented by the following sign $\|.\|$.

Models admit a set of parameters exactly and exclusively satisfying an **inference pattern** if they are derived directly from that pattern [31]. This interference pattern helps the model to understand the KG. For example: once a model learns a composite pattern, it can reliably predict facts in the composite closure of r. In the following part, we define several inference patterns.

Definition 1 *A relation r is **reflexive** on a set of entities $\mathcal{E}$ if $(e, r, e) \in \zeta$ and for all entities $e \in \mathcal{E}$.*

Definition 2 *A relation r is **symmetric** if $\forall x, y \in \mathcal{E}$*

$$r(x, y) \in \zeta \iff r(y, x) \in \zeta$$

Such a clause exhibits a symmetric pattern (e.g., Marriage, Family, Roommate).

Definition 3 *A relation r is **anti-symmetric** if $\forall x, y \in \mathcal{E}$*

$$r(x, y) \in \zeta \iff \neg r(y, x) \in \zeta$$

Such a clause exhibits an anti-symmetric pattern (e.g., Mother of, Father of).

Definition 4 *A relation r is **transitive** if $\forall$ $x,y,z \in \mathcal{E}$*

$$r(x, y) \in \zeta \wedge r(y, z) \in \zeta \implies r(x, z) \in \zeta$$

Such a clause exhibits a transitive pattern (e.g., Brother of, Sister of).

Definition 5 *A relation* r_1 *is* ***inverse*** *to relation* r_2 *if* $\forall$ $x,y \in \mathcal{E}$

$$r_1(x,y) \in \zeta \iff r_2(y,x) \in \zeta$$

Such a clause exhibits a inversion pattern (e.g., Husband-Wife, Mother-Child, Father-Child).

Definition 6 *A relation* r_1 *is* ***composed*** *of relation* r_2 *and* r_3 *if* $\forall$ $x,y,z \in \mathcal{E}$

$$r_2(x,y) \in \zeta \wedge r_3(y,z) \in \zeta \implies r_1(x,z) \in \zeta$$

Such a clause exhibits a composition pattern (e.g., My mother's husband is my father).

Definition 7 *A relation* r_1 *and* r_2 *are in* ***hierarchy*** *if* $\forall$ $x,y, \in \mathcal{E}$

$$r_1(x,y) \in \zeta \implies r_2(x,y) \in \zeta$$

Such a clause exhibits a hierarchical pattern (e.g., If a person is paying taxes in Canada, he\she is definitely working in Canada.).

Definition 8 *An* ***intersection*** *between relation* r_1 *and* r_2 *define as* $\forall$ $x,y, \in \mathcal{E}$

$$r_1(x,y) \in \zeta \wedge r_2(x,y) \in \zeta \implies r_3(x,y) \in \zeta$$

Such a clause exhibits an intersection pattern (e.g., Person born in and lives in India then person is the citizen of India.).

A tensor factorization approach is called **fully expressive** if an entity and relation embedding will separate true triples accurately from false triples if there exists an entity and relation embedding for any ground-truth over all entities and relations [29].

In this thesis, a **Training Step** is defined as one gradient update. For example, if we have a batch size of 1024, then in each training step, 1024 triples shall be processed per gradient update (with respect to training the model). Also, the **Maximum Training Steps** determines the total/maximum number of steps required to train the model to optimal fit.

1.7 Thesis Organization

The rest of the thesis/research work is organized in the following manner.

In chapter 2, we discuss related work and literature review in the field of KGC based on a graph embedding. The Literature Review is further divided into three subcategories: translation-based approach, bilinear-based approach, and neural network-based approach.

In chapter 3, we introduce our proposed approach (HRotatE) to solve the KGC problem. Basically, HRotatE is the hybridized variant of the existing approach RotatE and SimplE. This chapter discusses step by step process of our approach and how it predicts the missing information in KG.

In chapter 4, we explain our experimental setup and environment, which includes tools and library used to implement our model (HRotatE), System Configuration, Dataset details, Hyper-parameters for training, detail of the statistical significant test, and detail of evaluation metrics that used to evaluate our model.

In chapter 5, we compared our results with various state-of-the-art models. We have conducted our experiments on five (5) different benchmark datasets. This chapter also includes the detailed results of the statistical significance test (independent two-sample t-test) on all the five-benchmark datasets. Additionally, we also compare our approach to native RotatE in which our model trained with just half of the number of the training steps required by the native RotatE model.

In Chapter 6, We conclude our research, explain the insights we gained during our research work, and describe some of the opportunities for future work.

Chapter 2

Related Work and Literature Review

Knowledge Graph (KG) represents the real-world information in the form of a triple consisting of a head, relation, and tail. Most of the KGs are highly incomplete. Thus, the fundamental goal of the Knowledge Graph Completion (KGC) algorithm is to complete the missing information in the knowledge graphs. This chapter presents several state-of-the-art models implemented to solve the problem of KGC and which are based on knowledge graph embedding. The fundamental goal of the KG embedding-based approaches is to learn the embedding. So, that, it can predict the missing information from it. KG is the active research field due to the fact that it used in various real-life applications including social networks [32–34], recommendation systems [35, 36], question answering systems [37–39], Natural Language Processing (NLP) [40, 41], etcetera. Depending on the characteristics of each approach, we divided the literature review in three parts: translation-based approach [20–22], bilinear-based approach [19, 23–25], and neural-network-based approach [26–28].

2.1 Translational Approaches

Translation-based models compute embedding based on entities and relations such that the distance between two entities is used to calculate the score function. The relation

embedding is used to map the entity pairs. In general, translation-based models are better at representing some properties of KG. However, many of them are not fully expressive.

TransE [20]: TransE model embeds each entity, e, in a vector, $v_e \in \mathbb{R}^d$; and each relation, r, in a vector, $v_r \in \mathbb{R}^d$; where d is the size of the embedding dimension [20]. Given a set of triple (h, r, t), TransE computes the score function via the formalism: $h + r - t$ such that $h, t \in \mathcal{E}$ denote the constituent entities, and $r \in \mathcal{R}$ denotes the binding or constituent relation. Thus, the properties of TransE are based on $h + r - t$. Basically, in a given ideal case, the summation of the head entity to the relation is close to the tail entity for any true triple: $(h, r, t) \in \zeta$. If otherwise, the summation is usually far away from the tail entity for any false triple: $(h, r, t) \in \zeta'$. Some merits of the TransE model are, viz: simplicity, efficiency, and it yields desirable results on several datasets. However, TransE cannot be able to learn the symmetric relation, and it cannot capture many-to-many, one-to-many, and many-to-one expressions too.

TransH [42]: TransH (Translating on Hyperplanes) improves upon the TransE model by resolving the issues of reflexive/one-to-many/many-to-one/many-to-many relations [42]. TransH projects the constituent relations and entities into a hyperplane. TransH uses two different vectors with regard to its translations such that: vector d_r is relation specific, while vector w_r is for the hyperplane. For any given triple (h, r, t), TransH embeds the constituent head and tail embeddings onto the hyperplane vector w_r. This projection is denoted via: $h_\perp$ and $t_\perp$. Thus, the formalism of TransH scoring is computed as: $\|h_\perp + d_r - t_\perp\|$.

TransR [21]: In TransE [20], it uses one embedding space for its constituent relations. However, in TransR [21], it uses distinct embedding space (relation-specific) for each relation in a bid to improve its prediction result. TransR represents each entity as a vector $v_e \in \mathbb{R}^d$, and model each relation as a vector $v_r \in \mathbb{R}^k$. In addition, TransR uses one additional projection matrix, $M_r \in \mathbb{R}^{d \times d}$, to project each entity into a relation-specific embedding space. For any given triple (h, r, t), TransR computes its score via the following formalism: $\|h_r + r - t_r\|$, where $h_r = hM_r$ and $t_r = tM_r$. Although, TransR resolves several pitfalls of TransE; however, it cannot effectively capture composite relationships.

RotatE [22]: Unlike TransE, RotatE uses a rotation to optimize the score. Inspired from ComplEx, RotatE maps the entities and relations into a complex space. In order to

accomplish this mapping to a complex plane, RotatE uses Euler's identity $e^{i\theta} = \cos\theta + i\sin\theta$. So, the primary difference between RotatE and transE lies in the fact that RotatE defines each relation as a rotation from the head entity to the tail entity. For any triplet (h, r, t), RotatE does element-wise multiplication $t = h \circ r$ where $h, r, t \in \mathbb{C}^d$ [22].

BoxE [43]: In BoxE, relations are represented as a set of boxes or hyper-rectangles; and entities are represented as a point in these boxes or hyper-rectangles. Every entity in BoxE is represented via two vectors e_i and b_i, such that e_i defines the base portion and b_i defines the translation bump. Both $e_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}^d$, where d denotes the dimension of the Euclidean space. Also, every relation r is represented by a n hyper rectangle. For instance: $r^{(1)}, ..., r^{(n)} \in \mathbb{R}^d$ [43]. BoxE is a fully expressive model, and it performs well on several datasets, especially YAGO3 [44]. However, BoxE is unable to predict composition pattern.

2.2 Bilinear Approaches

Generally, effectiveness and performance on Link Prediction (LP) tasks is desirable with bilinear-based models in comparison to the translation-based models. By embedding entity and relationship representations in a vector space, the semantic matching model (Bilinear Model) can mine the possible semantic association between entities and relationships [9]. These models use relations after bilinear transformation to describe the relationship between entities and relations, which helps to capture various interactions between data. Bilinear-based models represent relations in the form of a $d \times d$ matrix. Each relation r is represented via a $M_r \in \mathbb{R}^{d \times d}$; and each entity e is represented as a vector $v_e \in \mathbb{R}^d$. The similarity score function is computed as a product of the relation and its associated pair of entities. Generally, most of the bilinear-based models are fully expressive.

RESCAL [25]: RESCAL uses three-way tensor factorization and the inherent structure of dyadic relational data. In RESCAL, every entity is represented by the vector $v_e \in \mathbb{R}^d$ and relation is represented by the relationship matrix $M_r \in \mathbb{R}^{d \times d}$. RESCAL scoring function contains a full-rank relation matrix $M_r \in \mathbb{R}^{d \times d}$ and the product of the entities (head and tail) $M_r \cdot vec(v_h \otimes v_t)$, where $\otimes$ represents the outer product of two vectors. RESCAL

requires too much computation and relatively too many parameters. As a result, RESCAL faces overfitting [25].

DistMult [23]: DistMult tends to reduce the complexity of RESCAL via reducing the full-rank relation matrix to a diagonal matrix. In DistMult, embedding for each entity e is represented via the vector $v_e \in \mathbb{R}^d$, and each relation r is represented via the vector $v_r \in \mathbb{R}^d$. DistMult is unable to differentiate between the source entity and the target entity. In this regard, it can not effectively predict asymmetric relations. Hence, DistMult is not a fully expressive model.

ComplEx [24]: - ComplEX improve the DistMult by considering complex values for relations and entities instead of their respective real values. This gives ComplEX the advantage to address the asymmetric relations. In ComplEx, each entity, e, embedding is represented by means of real (re) and imaginary (im) parts. For example: $re_e \in \mathbb{C}^d$ and $im_e \in \mathbb{C}^d$. Likewise, each relation, r, embedding in ComplEx is represented by both real and imaginary parts. E.g. $re_r \in \mathbb{C}^d$ and $im_r \in \mathbb{C}^d$. ComplEx is not able to predict composition rules because it does not model a bijection mapping from the source node (head) to the destination node (tail) by relation.

SimplE [19]: SimplE is based on Canonical Polyadic decomposition (CP) [30]. In CP, entities are mapped to the vectors, $h_e, t_e \in \mathbb{R}^d$, where h and t denotes the head entity and tail entity, respectively. In CP, every relation is mapped to a single embedding vector, $v_r \in \mathbb{R}^d$. SimplE [19] computes the inverse of every relation, and generates a corresponding embedding for it, $v_r^{-1} \in \mathbb{R}^d$. Additionally, SimplE makes the embedding of both entities (head and tail) dependent on each other. SimplE computes its main score function via the multiplication of the relation embedding with both head and tail embeddings. Also, SimplE computes an inverse score function, which is the multiplication of the inverse-relation embedding with both the head embedding and tail embedding. Thus, the score function in SimplE is considered to be the average of the main score and inverse score.

TuckER [29]: TuckER is based on TuckER decomposition [45]. TuckER algorithm decomposes the tensor into smaller tensors and matrices. TuckER uses one additional tensor, $W \in \mathbb{R}^{d_e \times d_r \times d_e}$, such that d_e represents the dimension of each entity, e; where d_r represents the dimension of each relation, r. In each training step, TuckER encodes some

knowledge into its core tensor, W, which is shared by all the entities and relations for the purpose of multi-task learning. Thus, the resultant scoring function is computed via the formalism: $W \times h \times r \times t$.

2.3 Neural Network-based Approaches

In Deep Learning (DL) models, deep neural networks are used to perform the LP task. In order to recognize significant patterns, neural networks learn parameters such as weights and biases. Deep neural networks typically organize parameters into layers, generally interspersed with non-linear activation functions. Neural network-based approaches yielded remarkable predictive performance in recent studies. However, they are harder to train and more prone to overfitting.

E-MLP [27]: In E-MLP, each entity e is represent by a vector, $v_e \in R^d$; and each relation r is represented by a matrix $M_r \in \mathbb{R}^{2k \times m}$ and vector $v_r \in \mathbb{R}^m$. The E-MLP model is a two-layer neural-network model where the first layer contains the weights from matrix M_r, and the second layer contains the weights from vector v_r. The input to the neural network is a 2-tuple, $(v_h, v_t) \in \mathbb{R}^{2d}$. ER-MLP [46] proposed an improved variant of E-MLP model which is essentially a four-layer neural network architecture.

ConvE [26]: ConvE is a LP model based on the architecture of Convolutional Neural Network (CNN). CNN has the ability to extract multi-scale local space features and combining them to build efficient representation. ConvE uses 2D-convolution over embeddings and multiple layers of nonlinear features to model knowledge graphs. ConvE uses very few parameters in comparison to DistMult and several other linear models. Also, it employs dropout and batch normalization to mitigate the effect of overfitting.

ConvKB [47]: ConvKB [47] employs CNNs to encode the concatenation of entities and relations without reshaping. In ConvKB, each triple (head entity, relation, tail entity) is represented by the 3- column matrix where each column vector represents an element of the triple (head, relation, tail). After that, this 3- column matrix is passed to a convolution layer where different filters are applied to the matrix to generate different feature maps. In the next step, these feature maps are concatenated into a single feature vector which

represents the input triple [47]. The final score is calculated by the dot product between the feature vector and the weight vector. By concatenating a set of latent feature maps, the ability of latent features is increased. ConvKB maintains the transitional characteristic compared to ConvE, which captures local relationships [47].

HypER [48]: - HypER uses a hyper-network model for knowledge graph completion. Generally, in the hyper-network model, one network generates the weights for the other network, which is useful for weight sharing across both networks. Just like ConvE, HypER is based on a CNN, and it can be used for LP. It used a 1D convolution filter for each relation. HypER uses hard regularization by setting most element weight tensor to zero.

Despite achieving nearly the same result as linear models, neural-network models are highly unintuitive. They are like black boxes. They are not transparent and less interpretable in comparison to the translation-based and bilinear-based models. Hence, in this paper, we have compared our approach against a couple of neural-network-based approaches.

Table 2.1 summarizes several models and the inference patterns they capture [19, 22, 43]. The detailed proofs of this inference pattern are contained in literature [20, 22, 24, 29, 43]. Our proposed model (HRotate) inherits the properties of RotatE [24], so it can predict any inference pattern that RotatE predicts (See Appendix B for more details).

Table 2.2 summarizes the literature review. The table contains information about the embedding space for entities and relations, as well as scoring for each method. In table 2.2, h, r, t represents the head, relation and tail, d is the dimension of the embedding, $\circ$ represents the element-wise multiplication or Hadamard product, e_i defines the base portion and b_i defines the translation bump, $r^{(i)}$ represent the i^{th} number of hyper-rectangle, $\otimes$ refers the outer product, Re denotes the real vector, $\mathcal{W} \in \mathbb{R}^{d_e \times d_r \times d_e}$ is the core tensor of a Tucker decomposition, U_r denotes the linear layer, b_r is the bias, σ defines the sigmoid activation function while $tanh$ and $relu$ defines the hyperbolic tangent activation function and Rectified Linear Unit, $*$ denotes the convolution operation, Ω is the filter, and H denotes the hyper network.

TABLE 2.1: Inference patterns captured by selected models

Model	Symmetry	Anti-symmetry	Inversion	Composition	Hierarchy	Intersection
TransE [20]	✗	✓	✓	✓	✗	✓
DistMult [23]	✓	✗	✗	✗	✓	✗
ComplEx [24]	✓	✓	✓	✗	✓	✗
RotatE [22]	✓	✓	✓	✓	✗	✓
TuckER [29]	✓	✓	✓	✗	✓	✗
BoxE [43]	✓	✓	✓	✗	✓	✓
HRotatE	✓	✓	✓	✓	✗	✓

TABLE 2.2: A comprehensive summary of the Knowledge Graph Completion models

Category	Model	Entity Embedding	Relation Embedding	Scoring Function
Translational	TransE [20]	$h, t \in \mathbb{R}^d$	$r \in \mathbb{R}^d$	$h + r - t$
	TransH [42]	$h, t \in \mathbb{R}^d$	$r, d_r \in \mathbb{R}^d$	$\|h_\perp + d_r - t_\perp\|$
	TransR [21]	$h, t \in \mathbb{R}^d$	$r \in \mathbb{R}^k, M_r \in \mathbb{R}^{k \times d}$	$\|M_r h + r - M_r t\|$
	RotatE [22]	$h, t \in \mathbb{C}^d$	$r \in \mathbb{C}^d$	$\|h \circ r - t\|$
	BoxE [43]	$e_i, b_i \in \mathbb{R}^d$	$r^{(1)}, ..., r^{(n)} \in \mathbb{R}^d$	$\sum_{i=1}^{n} \|dist(e_i{}^{r(e_1,..,e_n)}, r^{(i)})\|_x$
Bilinear	RESCAL [25]	$h, t \in \mathbb{R}^d$	$M_r \in \mathbb{R}^{d \times d}$	$M_r \cdot vec(v_h \otimes v_t)$
	DistMult [23]	$h, t \in \mathbb{R}^d$	$r \in \mathbb{R}^d$	$h^\top M_r t$
	ComplEx [24]	$h, t \in \mathbb{C}^d$	$r \in \mathbb{C}^d$	$Re(\langle r, h, \bar{t} \rangle)$
	SimplE [19]	$h, t \in \mathbb{R}^d$	$r, r^{-1} \in \mathbb{R}^d$	$\frac{1}{2}(\langle h_{e_i}, v_r, t_{e_j} \rangle + \langle h_{e_j}, v_r^{-1}, t_{e_i} \rangle)$
	TuckER [29]	$h, t \in \mathbb{R}^d$	$r \in \mathbb{R}^d$	$\mathcal{W} \times h \times r \times t$
Neural Network	E-MLP [27]	$h, t \in \mathbb{R}^d$	$M_r \in \mathbb{R}^{2k \times m}$, $v_r \in \mathbb{R}^m$	$U_r^\top tanh(h^\top M_r^{[1..k]} t + v_r \begin{bmatrix} h \\ t \end{bmatrix}) + b_r$
	ConvE [26]	$M_h \in \mathbb{R}^{d_w \times d_h}, t \in \mathbb{R}^d$	$r \in \mathbb{R}^{d_w \times d_h}$	$\langle \sigma(vec(\sigma([M_h, M_r] * \Omega))W)t \rangle$
	ConvKB [47]	$h, t \in \mathbb{R}^d$	$r \in \mathbb{R}^d$	$concat(\sigma([h, r, t] * \Omega)).w$
	HypER [48]	$h, t \in \mathbb{R}^d$	$r \in \mathbb{R}^d$	$\langle ReLU(vec(h * vec^{-1}(rH))W)t \rangle$
Ours	HRotatE	$h, t \in \mathbb{C}^d$	$r, r^{-1} \in \mathbb{C}^d$	$\frac{1}{2}(\|(h_{e_i} \circ v_r) - t_{e_j}\| + \|(h_{e_j} \circ v_r^{-1}) - t_{e_i}\|)$

Each of the aforementioned approaches possesses its respective strengths and weaknesses. For instance, translation-based approaches are generally based on sets of logical rules which can not apply to all kinds of properties of KGs and neural network are like black boxes. Neural network-based approaches are not transparent and poorly understood in comparison to other approaches [29]. Hence, this thesis focuses on translation-based and bilinear approaches. In the next section, we are going to define our proposed approach HRotatE, which is based on RotatE and SimplE. We select RotatE over the other models because its scoring function is less complex and the performance of RotatE is quite Impressive.

Chapter 3

Proposed Approach

3.1 Introduction

In this section, we have described our proposed hybrid variant of RotatE [22]. Basically, HRotatE is a hybridization of SimplE [19] and RotatE [22] models. HRotatE uses the inverse relation, and it computes an inverse score based on the inspiration drawn from SimplE [19].

In RotatE, the head entity and the tail entity are mapped by Euler's identity to a complex vector space such that $h, t \in \mathbb{C}^d$. The relation, r, embedding is essentially an element-wise rotation from the head entity to the tail entity. RotatE defines the scoring function as shown in equation 3.1:

$$d_r(h, t) = \|h \circ r - t\| \tag{3.1}$$

In equation 3.1, $\circ$ represents an element-wise multiplication or Hadmard product, d_r is a scoring function that measures the distance between $h \circ r$ and t, and $\|.\|$ is used for the L1 Norm. In this way, the expectation of RotatE is as shown in equation 3.2, where $(h_i, r_i, t_i) \in \mathbb{C}$ and $|r_i| = 1$ [1]

$$t_i = h_i \circ r_i \tag{3.2}$$

[1] $|r_i| = 1$ is based on the property of Euler's identity

SimplE[19] is based on a Canonical Polyadic (CP) decomposition [30]. In SimplE, the entity set, e, is comprised of two (2) distinct embedding vectors, $h_e, t_e \in \mathbb{R}^d$, per triplet. Furthermore, each relation, r, is represented via two vectors: $v_r, v_r^{-1} \in \mathbb{R}^d$. The major problem with Canonical Polyadic (CP) is that both entity (embedding) vectors are independent of each other. Thus, SimplE tends to resolve this problem via the introduction of an inverse-relation embedding function; and this aims at infusing interdependence between the embedding vectors for the head and tail entities. For triple (e_i, r, e_j), SimplE defines the scoring function as Equation 3.3

$$score = \frac{1}{2}(\langle h_{e_i}, v_r, t_{e_j}\rangle + \langle h_{e_j}, v_r^{-1}, t_{e_i}\rangle) \tag{3.3}$$

3.2 Euler's identity

RotatE, and our proposed approach use relation as a Rotation between the head to the tail entity. In order to employ rotation, we used Euler's identity [49].

$$e^{i\pi} + 1 = 0 \tag{3.4}$$

Equation 3.4 represent the Euler's identity, where e is Euler's constant, i is the imaginary number ($i^2 = -1$), π is an irrational number (with unending digits) that is the ratio of the circumference of a circle to its diameter. 3.1415... is the approximate value of π [50]. Euler's constant (e) can be defined as the base of natural logarithms that arise naturally through the study of compound interest and calculus. The approximate value of e is 2.7182... [49].

The general motivation of Euler's identity is comes from Euler's equation ($e^{i\theta} = \cos\theta + i\sin\theta$), which is stated that a unitary complex number can be viewed as a rotation in the complex plane [51]. Figure 3.1 represents Euler's formula. In the equation, when the value of θ is equal to π, the resultant equation is known as Euler's identity (Equation 3.4). Thus, we embedded our entities (head and tail), relation, and relation inverse into the complex vector space. In HRotatE, relation defines the rotation from the source entity to the target entity, while relation inverse defines the rotation from the target entity to the source entity.

In HRotatE, we uniformly distributed r and r^{-1} in [-π, π]. Here, we constrain the

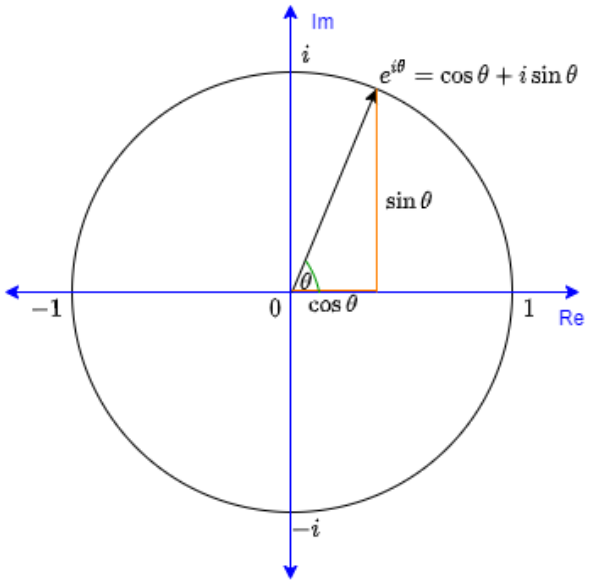


FIGURE 3.1: Euler's equation

modulus of each element of $r \in \mathbb{C}^k$, i.e. , $r_i \in \mathbb{C}$, to be $|r_i| = 1$. Thus, r_i becomes in the form of $e^{i\theta_{r,i}}$. Therefore, r_i corresponds to a rotation by $\theta_{r,i}$ radians about the origin of the complex plane. The same way r_i^{-1} corresponds to a rotation by $\theta_{r^1,i}$ radians about the origin of the complex plane.

3.3 HRotatE

In HRotatE, each entity, e, is represented by the two vectors, $h_e, t_e \in \mathbb{C}^d$, and each relation, r, is also represented by two vectors $v_r, v_r^{-1} \in \mathbb{C}^d$. Here, we define different embedding-generation classes for the head entity and the tail entity. However, both entities (head and tail) are still interdependent because of the inverse relation connecting both entities. Essentially, these places our proposed model, HRotatE, at an edge above RotatE and other similar models. The scoring function of HRotatE on a triple (e_i, r, e_j) is defined as in equation 3.5:

$$d_r(h,t) = \frac{1}{2}(\|(h_{e_i} \circ v_r) - t_{e_j}\| + \|(h_{e_j} \circ v_r^{-1}) - t_{e_i}\|) \tag{3.5}$$

Furthermore, Figure 3.2 depicts the main-score function, and Figure 3.3 depicts the inverse-score function of HRotatE. Thus, we have used the average of both the main score,

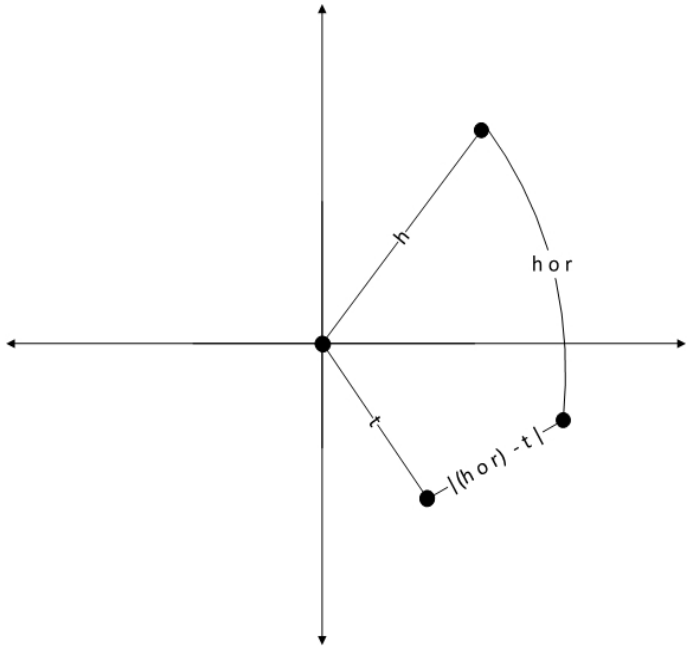


FIGURE 3.2: HRotatE calculates the main score by modeling the relation, r, as a rotation between head, h, to tail, t, in the complex plane.

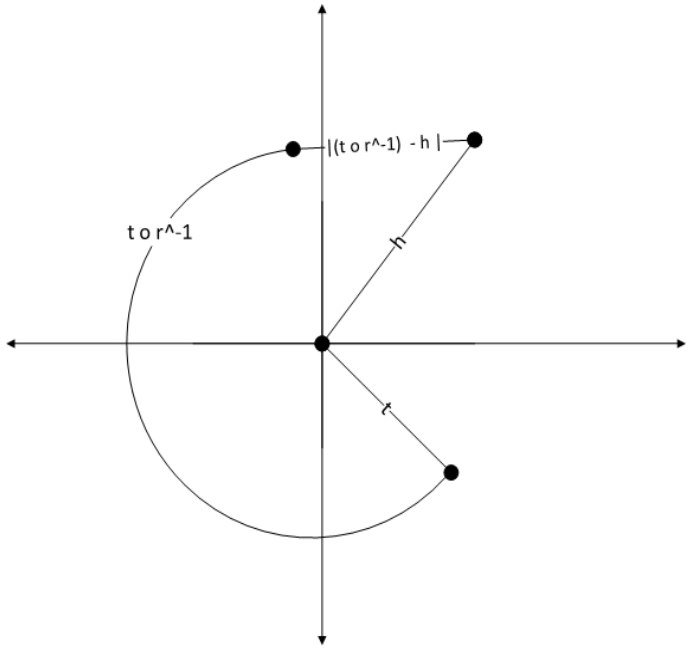


FIGURE 3.3: HRotatE calculates the inverse score by modeling the inverse relation, r^{-1}, as a rotation between tail, t, to head, h, in the complex plane.

(e_i, r, e_j), and inverse score, (e_j, r^{-1}, e_i). The primary advantage of our proposed model (HRotatE) over RotatE is that, we have used different embedding-generation classes for generating embedding vectors for the head entity and the tail entity. Hence, this allows

HRotatE to learn more efficiently. Our proposed model, HRotatE, uses Adam optimizer and mini-batches to train the model. In the following chapter, we demonstrate that our model not only gives better results than the other models, but it also achieves the same results as RotatE, with relatively much lesser training steps (half the number of training steps), on many datasets.

3.4 Negative Sampling

The positive triples are the ground-truth triples given in the dataset. Therefore, to effectively train our HRotatE model, there lies the need to generate negative samples as well [24, 52]. In optimization, the goal is to maximize the plausibility of positive facts while minimizing the plausibility of negative facts; in practice, this amount is applying a triplet loss function [53]. Herein, the HRotatE goal is to reduce the score (distance) for positive triplets and increase the score (distance) for negative triplets. New ways for generating negative triples have been proposed in recent years, including generating negative samples from the uniform way and generating negative samples from the adversarial way. BoxE [43] and RotatE [22] show the advantages of adversarial sampling method over uniform method. In this regard, herein, we used the same self-adversarial negative sampling, as employed in RotatE [22, 52], to generate negative samples for training our model, HRotatE. In, Uniform Sampling, negative samples are randomly generated by corrupting the triples. The performance of our approach on the uniform sampling is discussed in Appendix A.

Milkov et. al [52] proposed a negative sampling loss function capable of effectively optimizing the distance-based model.

$$L = -\log \sigma(\gamma - d_r(h,t)) - \sum_{i=1}^{n} \frac{1}{k} \log \sigma(d_r(h'_i, t'_i) - \gamma) \tag{3.6}$$

Equation 3.6 describes the loss function proposed by Milkov et. al [52], where σ is the sigmoid function, γ is a fixed margin, and (h'_i, r, t'_i) is the i-th negative triplet. However, the above approach performs the negative sampling uniform way. The main drawback of uniform sampling is that, during the training process, as training progresses, some of the samples are becomes obviously false, and this doesn't provide any meaningful information.

To avoid such issue, RotatE propose the self-adversarial negative sampling method. The basic idea behind the self-adversarial negative sampling is, it samples negative triples based on the current embedding model. Thus, RotatE used the following distribution, as shown in equation 3.7, for the self-adversarial negative sampling:

$$p(h'_j, r, t'_j | \{(h_i, r_i, t_i)\}) = \frac{\exp \alpha f_r(h'_j, t'_j)}{\sum_i \exp \alpha f_r(h'_i, t'_i)} \tag{3.7}$$

In equation 3.7, α is used to adjust the sampling strategy. (h'_i, r, t'_i) is the i-th negative term. the above probability (equation 3.7) is treated as the weight of the negative sample. Hence, the final loss function for negative sampling is as computed via equation 3.8, where σ is a sigmoid function, and γ is fixed margin.

$$L = -\log \sigma(\gamma - d_r(h, t)) - \sum_{i=1}^{n} p(h'_i, r, t'_i) \log \sigma(d_r(h'_i, t'_i) - \gamma) \tag{3.8}$$

The main difference between our proposed model (HRotatE) and RotatE is lies in the scoring function. Thus, HRotatE calculates the $d_r(h'_i, t'_i)$ based on the equation 3.5 while, in RotatE, $d_r(h'_i, t'_i)$ is calculated as per the equation 3.1.

3.5 HRotatE Algorithm

Algorithm 1: Learning HRotatE

Input: Training Set $S = \{(h, r, t)\}$

Initialize: Initialize the hyper-parameter α, γ, learning rate, hidden dimension and generating negative samples, generating embedding class for h, r, t, r^{-1}

Loop until the terminal condition is met == Maximum Training Steps:

- $S_{batch} \leftarrow sample(S, b)$ `// Sample a minibatch of size, b`
- $h_e, v_r, t_e, v_r^{-1} \leftarrow h, r, t$ `// Generating Embedding vector`
- **for** $(e_i, r, e_j) \in S_{batch}$ **do**
  - score $= \frac{1}{2}(\|(h_{e_i} \circ v_r) - t_{e_j}\| + \|(h_{e_j} \circ v_r^{-1}) - t_{e_i}\|)$ `// calculate Score using this equation`
- Update embeddings w.r.t.
  - $-\log \sigma(\gamma - d_r(h, t)) - \sum_{i=1}^{n} p(h'_i, r, t'_i) \log \sigma(d_r(h'_i, t'_i) - \gamma)$

The summarized training process for HRotatE is depicted via the Algorithm 1. The input to Algorithm 1 is the set of all true (ground truth) triplets given in the dataset. After that, we initialize the hyper-parameters and generate the negative samples. HRotatE computes its scoring function using equation 3.5. Finally, HRotatE updates the embedding using equation 3.8.

As the size of the Knowledge Graph (KG) keep growing [54], to keep up with this growth, Knowledge Graph Completion (KGC) model must have a linear time and space complexity. Table 2.2 illustrates the information of the space required for each entity and relation embedding. A model with too many parameters usually tends to overfit which resulted in poor performance.Thus, it is important to trade-off between model expressivity and model complexity. The time complexity of the HRotatE algorithm is $O(d)$, where d is the size of the embedding vector. TransR [21] requires projection matrix, $M_r \in \mathbb{R}^{d \times d}$, which increase the time complexity to $O(d^2)$. RESCAL [25] and E-MLP [27] have a quadratic time complexity [19]. DistMult [23], ComplEx [24], SimplE [19], and RotatE [22] have linear time complexities and the number of their parameters grow linearly with embedding dimension d.

3.6 Steps of the HRotatE

In this section, we are going to explain the step-by-step procedure of the HRotate learning algorithm. To make it simpler, we are going to explain the steps with the help of a sample example.

Input: The training dataset is the input of the model, which contains the positive triples. For example, triple T1 = (Sam_Raimi, directed, Spider-Man_(2002_film)), the example is taken from the Yago 3-10 train dataset [44].
Output: The output of the model is the score, which is then fed into a loss function to generate a loss and update the embedding.

Step 1: The dataset only contains the ground truth (positive triples). Thus, to learn the embedding better, we have to generate the negative samples. Negative samples can be generated by corrupting the head, tail, or relation entity. In HRotatE, negative samples are

generated from corrupting either head or tail entity. For the given an example (Sam_Raimi, directed, Spider-Man_(2002_film)), negative samples can be (Sam_Raimi, directed, jurrassic_park), (Christopher_Nolan, directed, Spider-Man_(2002_film)), (Sam_Raimi, directed, Boston_United_F.C.), etcetera. Let's consider the negative triple of the triple T1 is T2 = (Sam_Raimi, directed, jurrassic_park).

Step 2: The algorithm generates the embedding for each entity and relation in the dataset. In HRotatE, we generated two separate embeddings matrices for the head and tail entity, and two separate embeddings matrices for the relation and relation inverse embedding. Here, we took the dimension of the embedding as five (5) to better understand the algorithm. However, the embedding dimension is usually higher in the KG embedding. For the triple T1 and T2, we can define embedding vectors as below.

$$
\begin{aligned}
h_{Sam\_Raimi} &= [-0.1358, -1.1237, -0.5276, -0.8798, -0.4187] \\
t_{Sam\_Raimi} &= [0.0471, 0.8367, 0.2426, -1.1717, 1.1334] \\
h_{Spider-Man\_(2002\_film)} &= [-0.3826, -0.0883, 0.9972, -0.0141, 1.1319] \\
t_{Spider-Man\_(2002\_film)} &= [-0.8701, 0.6348, -0.8115, -0.3820, -0.3912] \\
r_{directed} &= [0.6030, 0.4455, -0.5405, 1.0803, 1.1042] \\
r^{-1}_{directed} &= [-0.3722, -0.2959, 0.3670, 0.2196, 0.3715] \\
h_{jurrassic\_park} &= [-0.8928, 0.6561, 0.1231, 0.7386, -1.1642] \\
t_{jurrassic\_park} &= [-0.7527, 0.5164, -0.2676, -0.7987, -0.2568]
\end{aligned}
$$

Step 3: In this step, the HRotatE algorithm calculates the score as per the equation 3.5. Thus, for the triple T1, the scoring equation looks like this:

$$
\begin{aligned}
Score_{T1} = \frac{1}{2}(&\|(h_{Sam\_Raimi} \circ r_{directed}) - t_{Spider-Man\_(2002\_film)}\| \\
&+ \|(h_{Spider-Man\_(2002\_film)} \circ r^{-1}_{directed}) - t_{Sam\_Raimi}\|)
\end{aligned}
$$

For the triple T2 (Negative Sample), the scoring equation looks like this:

$$
Score_{T2} = \frac{1}{2}(\|(h_{Sam\_Raimi} \circ r_{directed}) - t_{jurrassic\_park}\|
$$

$$+ \|(h_{jurrassic\_park} \circ r^{-1}_{directed}) - t_{Sam\_Raimi}\|)$$

The final score for triple T1 and T2 is calculated as follows.

$$Score_{T1} = [0.6708126, -1.01700835, 0.5527678, -0.15174794, -0.20552854]$$
$$Score_{T2} = [0.47800638, -1.02392417, 0.17767275, 0.59107431, -0.88571442]$$

Step 4: After calculating the score, the loss is calculated by the equation 3.8. Equation 3.8 can be split into two parts: positive loss and negative loss.

$$Positive\ Score = \log \sigma(\gamma - Score_{T1})$$
$$Negative\ Score = \sum_{i=1}^{n} p(T2) \log \sigma(Score_{T2}) - \gamma)$$

$$Positive\ Loss = [-0.4965, -1.2936, -0.4339, -0.5543, -0.9082]$$
$$Negative\ Loss = [0.7823]$$

As per the illustration purpose, we took only one negative sample for the example. If the algorithm generates more than one negative sample, the output of the negative loss will contain multiple values in the form of a vector. After the calculation of the loss, HRotatE transforms the positive and negative loss into 1-dimension by calculating the sum of the values. Furthermore, the final loss is calculated by the average of both losses.

$$Positive\ Loss = -3.6866$$
$$Negative\ Loss = 0.7823$$
$$Loss = -1.45215$$

In the end, HRotatE updates the embedding with respect to the loss value. This process is repeated from Steps 3-5 for all training steps.

In this chapter, we proposed a hybridized variant of RotatE and SimplE model. We took

two different embedding vectors for the head and the tail entity. In addition to that, to improve the performance model, we added the inverse relation embedding into our scoring function. In the next chapter, we are going to discuss the experimental setup and evaluation setting.

Chapter 4

Experimental Setup

This chapter describes our experimental setup and environment, including tools and libraries used to implement our model (HRotatE), System Configuration, Hyper-parameters for training, Dataset details, and detail of evaluation measures that used to evaluate our model.

4.1 Tools and Libraries

We have implemented our model (HRotatE) using the existing RotatE code developed and made available by Edward-Sun [1]. The up-to-date version of our code for the model, HRotatE, proposed herein is available on GitHub [2]. We have implemented our code in Python 3.6 language [55]. The details of the used libraries are listed below.

- PyTorch 1.7.1
- NumPy 1.19.2
- SciPy 0.20.2

[1] https://github.com/DeepGraphLearning/KnowledgeGraphEmbedding
[2] https://github.com/programmingboy/HRotatE

4.2 System Configuration

We tested our model in Compute Canada - SHARCNET Graham and Cedar cluster on Nvidia Tesla T4 and Nvidia Volta v100L Graphics Processing Unit (GPU). Nvidia Tesla T4 provides 16 GB of memory, while Nvidia Volta v100L provides 32 GB of memory. In terms of the Central Processing Unit (CPU) core, Tesla T4 provides 44 CPU cores while v100l provides 32 CPU cores.

4.3 Datasets

The proposed model is a generalized model which works well on several datasets. We have evaluated our model (HRotatE) in five (5) benchmark datasets with several other state-of-the-art models. Data within all five datasets are divided into three groups: train, test, and validation. Table 4.1 summarizes the basic statistics of each benchmark dataset. The datasets contain the data (triplets) in the form of a Comma-separated Values (CSV) file.

- **FB15k:** FB15k [20] dataset is a subset of a large Freebase [56] dataset which contains real world facts. The FB15k dataset contains knowledge base relation triples and textual mentions of Freebase entity pairs.
- **WN18:** WN18 [20] dataset is a subset of the wordnet [57]. WN18 dataset consists of a collection of triplets (synset, relation_type, triplet) extracted from WordNet 3.0 [3]. This data set can be seen as a 3-mode tensor depicting ternary relationships between synsets. Wordnet contains the lexical relationships between words. As part of a large lexical database of the English language, nouns, verbs, adjectives, and adverbs are divided into groups, also known as cognitive synonyms, with each expressing a distinct contextual concept[57].
- **WN18RR:** WN18RR [26] dataset is a subset of WN18. WN18RR is derived from WN18, with data removed to eliminate test-set leakage due to inverse relations. WN18RR features 11 relations only, no pair of which is reciprocal.

[3] http://wordnet.princeton.edu

Table 4.1: Dataset Statistics

Dataset	Entities	Relations	Training	Validation	Test
WN18	40,943	18	141,442	5,000	5,000
FB15k	14,951	1,345	483,142	50,000	59,071
WN18RR	40,943	11	86,835	3,034	3,134
FB15k-237	14,541	237	272,115	17,535	20,466
Yago3-10	123,182	37	1,079,041	5,000	5,000

- **FB15k-237:** FB15k-237 [58] dataset is a subset of FB15k. FB15k was found to suffer from major test leakage through inverse relations, and a large number of test triples can be obtained simply by inverting triples in the training set [58]. In order to create a dataset without this characteristic, the FB15k-237 was introduced – a subset of FB15k where inverse relations were removed [58].

- **Yago3-10:** YAGO 3 combines the information from the Wikipedias in multiple languages with WordNet, GeoNames, and other data sources. YAGO 3 taps into multilingual resources of Wikipedia, getting to know more local entities and facts [44] [4]. Yago3-10 is a subset of Yago3 [44] in which every entity has at least 10 relations.

4.4 Parameter Optimization

Optimizing the hyperparameters is a crucial step towards producing good results. This section defines the hyperparameters that our model uses, and their values use to generate the same result as mentioned in chapter 5.

Negative Sampling - This parameter illustrates the number of negative samples requires to train our model.

Batch Size - In Machine Learning (ML), batch size refers to the number of training examples trained in one iteration.

Learning Rate - Learning rate is a parameter that controls how much model weights are updated based on the estimated error. Basically, it determines the step size at each

[4] https://yago-knowledge.org/downloads/yago-3

TABLE 4.2: Best performing hyper-parameter values for HRotatE

Dataset	Negative Sampling	Batch Size	Learning Rate	Dimension	Alpha	Gamma	Maximum Step
WN18	32	1,024	0.00100	500	0.5	12	80,000
FB15k	128	256	0.00010	1,000	1.5	24	100,000
WN18RR	64	1,024	0.00005	500	0.5	6	80,000
FB15k-237	256	512	0.00010	1,000	1.5	10	150,000
Yago3-10	400	1,024	0.00020	500	1.0	24	100,000

iteration while moving toward a minimum of a loss function. Choosing the correct value for the learning rate is necessary for any ML algorithm.

Embedding Dimension - The embedding dimension defines the size of the embedding vector per entity or relation. As the size of the embedding dimension increases, it requires more computation power.

Alpha - Alpha (α) is the parameter which used in the probability distribution for the self-adversarial negative sampling.

Gamma - Gamma (γ) is the fixed margin used in the Loss function to differentiate positive sample to negative sample

We have chosen the optimal value for each hyper-parameter by performing a grid search over the scores obtained during training with respect to each dataset. In this regard, we performed series of experiments for each dataset to find the best value for each hyper-parameter. For the WN18 and WN18RR datasets, we have chosen 1,024 as the training batch size, and we used a dimension size of 500 for the generation of our embedding vectors. With regard to the WN18 dataset, we obtained optimal results using a 0.001 learning rate; and for the WN18RR dataset, we obtained optimal results using a learning rate of 0.00005. For the FB15k and FB15k-237 datasets, we used a relatively much higher dimension size of 1,000 with regard to the generation of embedding vectors; and we set the learning rate to 0.0001. For the Yago3-10 dataset, we used a learning rate of 0.0002 for training and applied a dimension size of 500 for the generation of its embedding vectors. Details of the hyper-parameters, with regard to each dataset, are contained in Table 4.2.

4.5 Evolution Metrics

In any ML approach, the evolution metric plays a key part in measuring its performance. Thus, choosing the right metric helps to learn and assess the ML model. In most cases, accuracy is used to evaluate the performance of the Artificial Intelligence (AI) algorithms. However, in the recommended system, or in Link Prediction (LP) system, accuracy is not performing well because these tasks are rank-based. Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) is a type of the prediction accuracy metrics. The primary focus of these metrics is to compare the actual value to the predicted value. Therefore, this type of metric can be used to determine how far off the prediction value is from the actual value. However, they are not suitable for the ranking system.

Decision support metrics such as Precision, Recall, or F1 Score are also not beneficial for the ranking task. Decision support metrics focus on measuring how well a recommender helps users make good decisions. However, they cover the entire dataset, not the 'Top-N' recommendation. Thus, we need to use Rank-Aware Evaluation Metrics such as Mean Rank, Mean Reciprocal Rank (MRR), Hit Ratio etcetera. Mean Rank, MRR and Hit Ratio are standard evaluation measures for these datasets. We did not employ Mean Rank herein as an objective function because it is highly influenced by a single bad ranking [19, 43]. In this regard, MRR has proven to be a much effective measure/metric over Mean Rank. In this paper, we evaluate our approach using the MRR and Hit Ratio.

4.5.1 Mean Reciprocal Rank

We have used the MRR filter version, which implies that we have only used test triples in the computation of the MRR score. To measure and compare the performances of different models, We rank test triples (h, r, t) against all other candidate triples not appearing in the training, validation, or test set, where candidates are generated by corrupting subjects ((h', r, t) for all $h' \in \mathbb{E}$) and objects ((h, r, t') for all $t' \in \mathbb{E}$). Basically, the MRR is computed based on the formula shown in equation 4.1.

$$MRR = \frac{1}{2 * |tt|} \sum (\frac{1}{rank_h} + \frac{1}{rank_t}) \tag{4.1}$$

In equation 4.1, tt denotes the test triples such that $(h, r, t) \in tt$. Basically, MRR is the average of the inverse of the obtained rank. The value always ranges between 0 and 1, and the higher it is, the better the model results.

4.5.2 Hit Ratio

In a LP, the hit ratio is the ratio of predictions for which the rank is equal or lesser than a threshold k. It also denotes $Hit@k$. Generally, the value of K is 1, 3, 5, or 10. The higher the value of $Hit@k$, the better the model results. As the value of k increases, the higher the hit ratio becomes because there is a higher chance that the correct answer is included in the prediction list. Therefore, it is important to choose an appropriate value for k [53]. The low values of k allow differentiating models easily compared to the higher value of k. Especially when the value of k is one ($k = 1$), it indicates the proportion of test facts where the target is correctly predicted by the first attempt. MRR and Hit@1 are often closely related, because MRR is also considers the most relevant prediction in its formula [53]. With respect to our benchmark experiments, the objective functions employed herein for comparative analyses are: and Hit score (Hit@1, Hit@3, and Hit@10).

4.6 Statistical Significance Test

The statistical significance test is useful in many domains to test the performance of the model [59]. **Significance Testing** or **Hypothesis Testing** is used to test the validity of a claim (**Null Hypothesis (H0)**) that is made about a population using sample data. In the event that the null hypothesis is not true, the **Alternative Hypothesis (H1)** is considered as true. In other words, first, a claim is made (null hypothesis), and then it is tested by some random sample to check the validity of the claim. If the claim is not valid, then the alternative hypothesis is accepted as a true hypothesis [59].

To check the hypothesis is valid or not, the **Significance Level (δ)** is used, which is a threshold value for the hypothesis test. Usually, the significance level is denoted by the α. However, we already determine the α for self-adversarial negative sampling; Thus, we use δ for the significance level. Choosing the right value of δ is necessary for any significance

test. If the value of δ is too big, it requires less evidence to reject the null hypothesis. If δ is smaller, it will require more evidence to reject the Null Hypothesis.

Generally, across all domains, the standard threshold value is 5% (0.05) accepted. That means, if an unexpected change in probability is less than 5%, then it can be concluded that there is a difference in the behavior of the two approaches. (1 - significance layer) is known as the **Confidence Level**, which indicates the confidence of the model (i.e., 95% confidence that 'model A; performs better than 'model B'). To determine if a statistically significant difference exists, there are various methods available such as P-value, Student's T-Test, Z score, etcetera [60]. We perform the Student's T-Test to determine the statistical significance of the HRotatE over the RotatE model.

4.6.1 Student's T-test

T-test (Student's T-test) is a statistical significance test that is used to compare the means of two groups\models and determine if the difference in means is statistically significant. This test is widely used in data analysis and statistical analysis [59]. T-Test was first invented by 'William Sealy Gosset' when he was working at Guinness Brewery [61]. So basically, Student's t-test is a method of testing hypotheses about the mean of a small sample drawn from a normally distributed population [59]. As the sample size (**degrees of freedom**) increases, the t distribution approaches the bell shape of the standard normal distribution.

Assumptions for Conducting a T-test

In order to perform a t-test, there is a need to keep the following criteria in mind [60]:

1. The data should follow a continuous or ordinal scale.
2. It is important to select the observations in data randomly.
3. The data should follow a bell-shaped curve when we plot it, i.e., it should be normally distributed.
4. Large sample size should be taken. Larger sample size means the distribution of results should approach a normal bell-shaped curve.

5. Variances among the groups should be equal (for independent two-sample t-test).

Types of t-tests

The main part of performing the t-test is in computing the T Statistic, which depends on the type of the t-test. There are mainly three types of t-test.

1. One sample t-test

   In a one sample t-test, the mean of one group is compared to the mean of another group. This mean can be a theoretical value, or the population mean. One sample t-test following formula to compute the t-statistic [61].

$$t = \frac{\bar{x} - \mu}{s/\sqrt{n}}$$

   Where $\bar{x}$ is the sample mean, μ is the population mean, s is the standard deviation, and n is the sample size. The numerator is also known as a signal, and the denominator is also known as noise. A common analogy is that the t-value is the signal-to-noise ratio.

   The numerator subtracts the null hypothesis value from the sample mean/ If the sample mean is 12, the null hypothesis is 6, the difference, or signal, is 6. If there is no difference between the null value and sample mean, the signal in the numerator, as well as the value of the entire ratio, becomes zero. As the difference between the sample means and the null hypothesis means increases, the signal's strength increases. The denominator measures the variability known as the standard error of the mean. This statistic describes how accurately the sample estimates the mean of the population. In general, a more significant number indicates a less accurate estimate due to a higher level of random error [59].

2. Independent two-sample t-test

   A two-sample t-test compares the means of two different samples \models. The number of data in each model should be equal for the comparison. This is where a two-sample t-test is used. Here's the formula to calculate the t-statistic for a two-sample

t-test [59].

$$t = \frac{\bar{x_1} - \bar{x_2}}{\sqrt{\frac{s_1^2}{n_1} + \sqrt{\frac{s_2^2}{n_2}}}}$$

where, $\bar{x_1}, \bar{x_2}$ is the mean of sample 1 and 2, s_1^2, s_2^2 is the sample variances, and n_1, n_2 is the size of the sample. With unequal sample sizes and varying variances of the two samples, determining degrees of freedom (df) is not as straightforward as in a 1-sample t-test. To calculate the degrees of freedom (df), Welch-Satterthwaite's formula is used [59].

$$df = \frac{(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2})^2}{\frac{1}{n_1 - 1}(\frac{s_1^2}{n_1})^2 + \frac{1}{n_2 - 1}(\frac{s_2^2}{n_2})^2}$$

The above equation can be generalize as below.

$$df = n_1 + n_2 - 2$$

Depending on the degree of freedom and δ, the critical value of T can be found in the lookup table [5]. This critical value of T (T-critical) is further used to determine null hypothesis is valid or not.

3. Paired sample t-test

   In the Paired Samples t-test, the means of two measurements taken from the same individual, object, or related units are compared. These "paired" measurements can represent things like A measurement taken at two different times, A measurement taken under two different conditions, A Measurements taken from two halves or sides of a subject or experimental unit. The formula to calculate the t-statistic for a paired t-test is:

$$t = \frac{\mu}{\frac{s}{\sqrt{n}}}$$

   Where μ is the mean of the group, s is the standard deviation of the group, and n is the size of the group.

We have used an independent two-sample t-test approach to compare our model with RotatE. This approach is widely used to compare the different knowledge graph models

[5] https://www.stat.purdue.edu/ lfindsen/stat503/t-Dist.pdf

[62].

This chapter explained the requirement for the implementation, dataset details, detail of the statistical significance test, and hyper-parameter setting to obtain the optimal result. In the next chapter, we will explain our results with a statistical significance test in detail.

Chapter 5

Discussions, Comparisons, and Analysis

In this chapter, we compare our results with various state-of-the-art models that include the translation-based models (such as TransE, RotatE, BoxE, etc.), bilinear models (such as DistMult, SimplE, TuckER, etc.), and the deep-neural-network based models (such as HypER, ConvE, etc.). We conducted our experiments on five (5) different benchmark datasets. The details of the experimental setup and hyperparameters setting are explained in Chapter 4. Herein, we have not compared our results with ComplEx-N3[63] because [63] it is based on 2000 embedding dimensions, which requires relatively too many parameters to implement. We have also performed the statistical significance test (independent two-sample t-test) on all the five benchmark dataset results. Additionally, To show the efficiency of our algorithm, we trained our model, HRotatE, using half of the number of training steps required by the native RotatE during its training. Thus, our comparative results indicate that our model converged to optimum much quicker than the native RotatE.

5.1 Result Analysis

5.1.1 WN18

Table 5.1 shows the detailed result of the WN18 dataset. Most of the data in this dataset are in the form of symmetry, inverse, and anti-symmetry. So, most of the approaches perform well on this dataset. For this dataset, HRotatE achieved its best score at Hit@10. For the other objective functions/measures, our result is comparable to TuckER and HypER.

Table 5.1: Result on WN18

Model	**MRR**	**Hit@1**	**Hit@3**	**Hit@10**
TransE [20]	0.495	0.113	0.888	0.943
TransR [21]	0.605	0.335	0.876	0.940
DistMult [23]	0.822	0.728	0.914	0.936
ComplEx [24]	0.941	0.936	0.945	0.947
HolE [64]	0.938	0.930	0.945	0.949
Analogy [65]	0.942	0.939	0.944	-
TorusE [66]	0.947	0.943	0.950	0.954
SimplE [19]	0.942	0.939	0.944	0.947
ConvE [26]	0.943	0.935	0.946	0.956
RotatE [22]	0.949	0.944	0.952	0.959
TuckER [29]	**0.953**	**0.949**	**0.955**	0.958
HypER [48]	0.951	0.947	0.955	0.958
HRotatE	0.951	0.945	0.954	**0.960**

5.1.2 FB15k

Table 5.2 highlights the results of comparative analyses on the FB15k dataset. Similar to WN18, most of the data in this dataset are in the form of symmetry, inverse, and anti-symmetry. Our proposed model, HRotatE, outperforms all other baselines (benchmark models) on this dataset, such that HRotatE achieves a state-of-the-art score for the MRR and Hit@1 measures/metrics. Obviously, TransE performed worst on this dataset because it is unsuitable for predicting links based on symmetric relations.

Table 5.2: Result on FB15k

Model	**MRR**	**Hit@1**	**Hit@3**	**Hit@10**
TransE [20]	0.465	0.297	0.578	0.749
TransR [21]	0.346	0.218	0.404	0.582
DistMult [23]	0.654	0.546	0.733	0.824
ComplEx [24]	0.692	0.599	0.759	0.840
HolE [64]	0.524	0.400	0.613	0.739
Analogy [65]	0.725	0.646	0.785	-
TorusE [66]	0.733	0.674	0.771	0.832
SimplE [19]	0.727	0.660	0.773	0.838
ConvE [26]	0.657	0.558	0.723	0.831
RotatE [22]	0.797	0.746	0.830	0.884
TuckER [29]	0.795	0.741	**0.833**	**0.892**
HypER [48]	0.790	0.734	0.829	0.885
HRotatE	**0.799**	**0.751**	0.833	0.832

Table 5.3: Result on WN18RR

Model	**MRR**	**Hit@1**	**Hit@3**	**Hit@10**
TransE [20]	0.226	-	-	0.501
DistMult [23]	0.430	0.390	0.440	0.490
ComplEx [24]	0.440	0.410	0.460	0.510
ConvE [26]	0.430	0.400	0.440	0.520
ConvKB [47]	0.249	0.057	0.417	0.524
RotatE [22]	0.476	0.428	0.492	0.571
TuckER [29]	0.470	**0.443**	0.482	0.526
HypER [48]	0.465	0.436	0.477	0.522
BoxE(u) [43]	0.470	-	-	0.523
BoxE(a) [43]	0.451	-	-	0.541
HRotatE	**0.483**	0.438	**0.499**	**0.572**

5.1.3 WN18RR

WN18RR is a subset of the WN18 dataset in which the inverse relations have been deleted. Thus, Table 5.3 illustrates the comparative results of different benchmark models on the WN18RR dataset. Most of the data in the WN18RR dataset follows symmetry and hierarchical patterns. Theoretically, HRotatE cannot capture hierarchical patterns; however, it still performs better than all other baselines (benchmark models) on the measure/metrics of MRR, Hit@3, and Hit@10. The primary reason behind this is that RotatE captures the composition patterns, which can be helpful to capture symmetric patterns. This is because, in RotatE, the composition of two symmetric relations is symmetric [43]. Thus, HRotatE

inherits this property from RotatE.

5.1.4 FB15k-237

FB15k-237 is a subset of the FB15K dataset in which the inverse relations have been deleted. Thus, Table 5.4 shows the comparative results of different benchmark models on the FB15k-237 dataset. This dataset contains several composition patterns. Our proposed model, HRotatE, performs well on FB15k-237; however, TuckER still outperforms our HRotatE model on this dataset. Thus, the primary reason behind TuckER's outstanding performance is that it is used as a shared parameter for multi-task learning.

TABLE 5.4: Result on FB15k-237

Model	**MRR**	**Hit@1**	**Hit@3**	**Hit@10**
TransE [20]	0.294	-	-	0.465
DistMult [23]	0.242	0.155	0.263	0.419
ComplEx [24]	0.247	0.158	0.275	0.428
ConvE [26]	0.325	0.237	0.356	0.501
ConvKB [47]	0.243	0.155	0.371	0.421
RotatE [22]	0.338	0.241	0.375	0.533
TuckER [29]	**0.358**	**0.266**	**0.394**	**0.544**
HypER [48]	0.341	0.252	0.376	0.520
BoxE(u) [43]	0.318	-	-	0.514
BoxE(a) [43]	0.337	-	-	0.538
HRotatE	0.338	0.243	0.373	0.530

5.1.5 Yago3-10

Yago3-10 is a subset of Yago3 [44] in which every entity has at least ten relations. Table 5.5 show the comparative results of different benchmark models on the Yago3-10 dataset. This dataset contains several hierarchical patterns. BoxE can capture hierarchical patterns effectively. As a result, this places BoxE at an edge over other benchmark models (baselines) on this dataset. On the Yago3-10 dataset, the BoxE model, which is based on uniform sampling, performs better than the BoxE model, which is based on adversarial sampling. In addition, BoxE employs data augmentation, which gives it an additional advantage.

TABLE 5.5: Result on Yago3-10

Model	**MRR**	**Hit@1**	**Hit@3**	**Hit@10**
DistMult [23]	0.340	0.240	0.380	0.540
ComplEx [24]	0.360	0.260	0.400	0.550
ConvE [26]	0.440	0.350	0.490	0.620
RotatE [22]	0.495	0.402	0.550	0.670
TuckER [29]	0.529	-	-	0.670
HypER [48]	0.533	**0.455**	**0.580**	0.678
BoxE(u) [43]	**0.567**	-	-	**0.699**
BoxE(a) [43]	0.560	-	-	0.691
HRotatE	0.497	0.399	0.554	0.681

5.2 Statistical Significance Test Results

The RotatE approach and the proposed HRotatE approach were tested fifteen times on all five datasets. We recorded the Mean Reciprocal Rank (MRR), Hit@1, Hit@3, Hit@10 scores for each test. We conducted an independent two-sample t-test on a MRR score of our approach (HRotatE) and RotatE. We plotted the normal distribution of all the test results of the dataset. For an independent two-sample t-test, we define our hypothesis as follows.

H0: No significant performance difference between the RotatE and HRotatE models.

H1: There is statistically significant that HRotatE performs better than RotatE.

For this test, we defines $\delta = 0.05$, which is standard δ value for hypothesis test. As we conducted experiment 15 times for both HRotatE and RotatE, the value of the degree of freedom (df) becomes 28 ($df = n_1 + n_2 - 2 = 15 + 15 - 2 = 28$). For $\delta = 0.05$ and $df = 28$, critical value for one-tail the t-test is $t - critical = 2.048$ [1]. We reject the Null hypothesis (H0) in case of $t - test > t - critical$, which leads to accepting our alternative hypothesis (H1) that there is a statistical significant that HRotatE performs better than the RotatE on the given dataset

[1] http://www.math.odu.edu/stat130/t-tables.pdf

5.2.1 Statistical Significance Test Results on WN18 Dataset

In WN18 dataset, RotatE achieves 0.949 mean MRR and the value of the standard deviation is 0.00032 while HRotatE achieves 0.951 mean MRR and the value of the standard deviation is 0.000381. We have performed the standard independent two-sample t-test on the MRR to test our hypothesis, which resulted in the $t - value = 10.91509$, which is higher than the $t - critical$. Thus, we rejected H0 in favor of H1. Figure 5.1 represent the normal distribution curve of the RotatE and HRotatE model on the WN18 database.

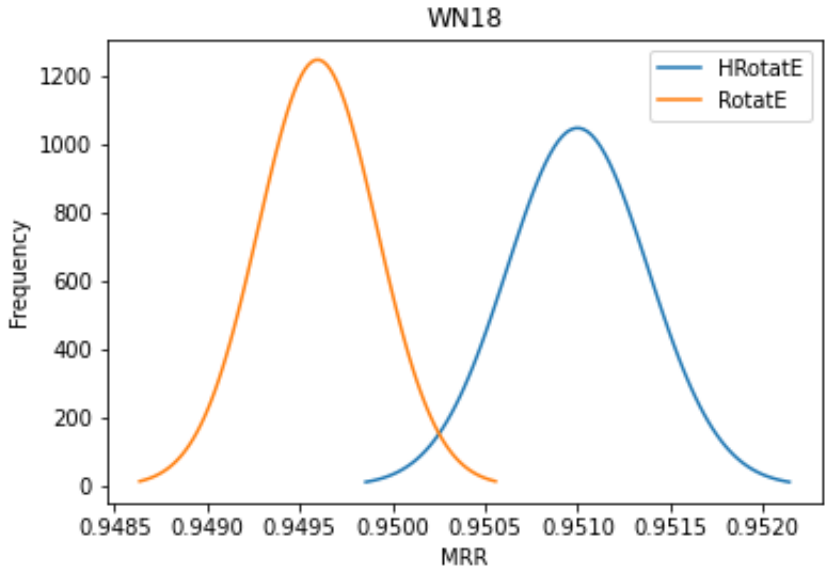


FIGURE 5.1: Normal distribution of the RotatE and HRotatE model on WN18 dataset

5.2.2 Statistical Significance Test Results on FB15k Dataset

In FB15k dataset, RotatE achieves 0.797 mean MRR and the value of the standard deviation is 0.00081 while HRotatE achieves 0.799 mean MRR and the value of the standard deviation is 0.0006. We have performed the standard independent two-sample t-test on the MRR to test our hypothesis, which resulted in the $t - value = 7.682869$, which is higher than the $t-critical$. Thus, we rejected H0 in favor of H1. Figure 5.2 represent the normal distribution curve of the RotatE and HRotatE model on the FB15k database.

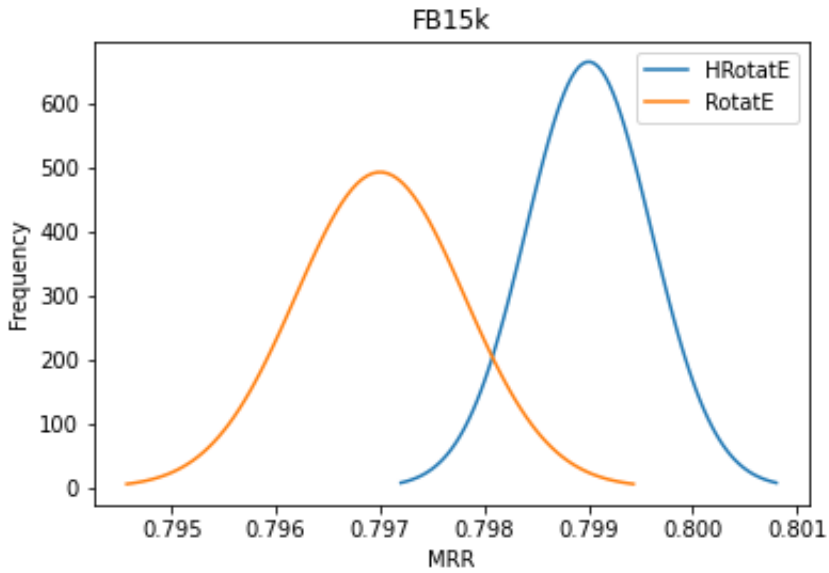


FIGURE 5.2: Normal distribution of the RotatE and HRotatE model on FB15k dataset

5.2.3 Statistical Significance Test Results on WN18RR Dataset

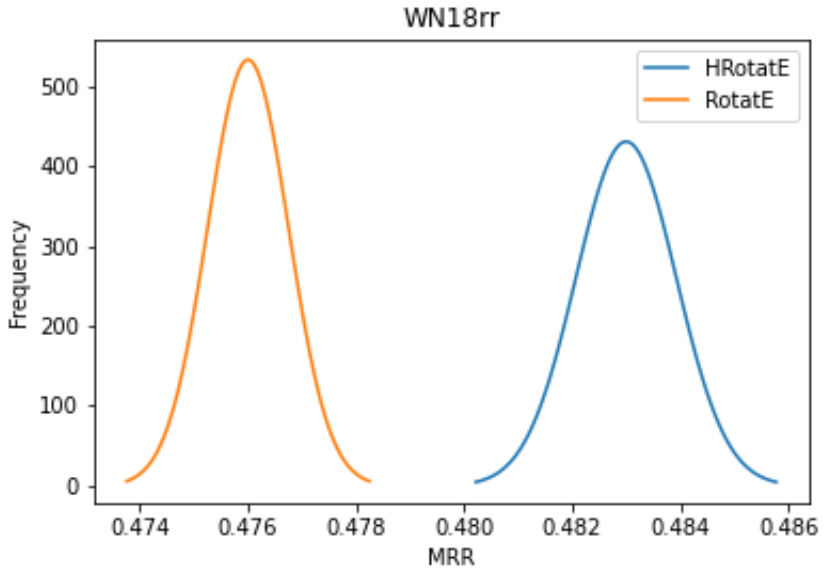


FIGURE 5.3: Normal distribution of the RotatE and HRotatE model on WN18RR dataset

In the WN18RR dataset, RotatE achieves 0.476 mean MRR and the value of the standard deviation is 0.000749 while HRotatE achieves 0.483 mean MRR and the value of the stan-

dard deviation is 0.000927. We have performed the standard independent two-sample t-test on the MRR to test our hypothesis, which resulted in the $t - value = 22.75168$, which is higher than the $t - critical$. Thus, we rejected H0 in favor of H1. Figure 5.3 represent the normal distribution curve of the RotatE and HRotatE model on the WN18RR database.

5.2.4 Statistical Significance Test Results on FB15k-237 Dataset

In FB15k-237 dataset, RotatE achieves 0.336 mean MRR and the value of the standard deviation is 0.000508 while HRotatE achieves 0.338 mean MRR and the value of the standard deviation is 0.000517. We have performed the standard independent two-sample t-test on the MRR to test our hypothesis, which resulted in the $t - value = 10.58801$, which is higher than the $t - critical$. Thus, we rejected H0 in favor of H1. Figure 5.4 represent the normal distribution curve of the RotatE and HRotatE model on FB15k-237 database.

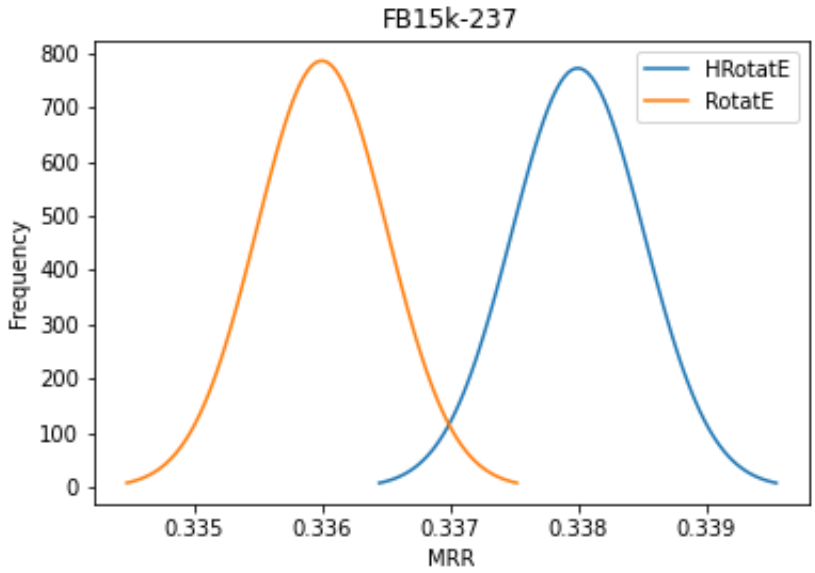


FIGURE 5.4: Normal distribution of the RotatE and HRotatE model on FB15k-237 dataset

5.2.5 Statistical Significance Test Results on Yago3-10 Dataset

In Yago3-10 dataset, RotatE achieves 0.495 mean MRR and the value of the standard deviation is 0.00162 while HRotatE achieves 0.497 mean MRR and the value of the standard deviation is 0.001844. We have performed the standard independent two-sample t-test on

the MRR to test our hypothesis, which resulted in the $t-value = 2.333249$, which is slightly higher than the $t - critical$. Thus, we rejected H0 in favor of H1. Figure 5.5 represent the normal distribution curve of the RotatE and HRotatE model on the Yago3-10 database.

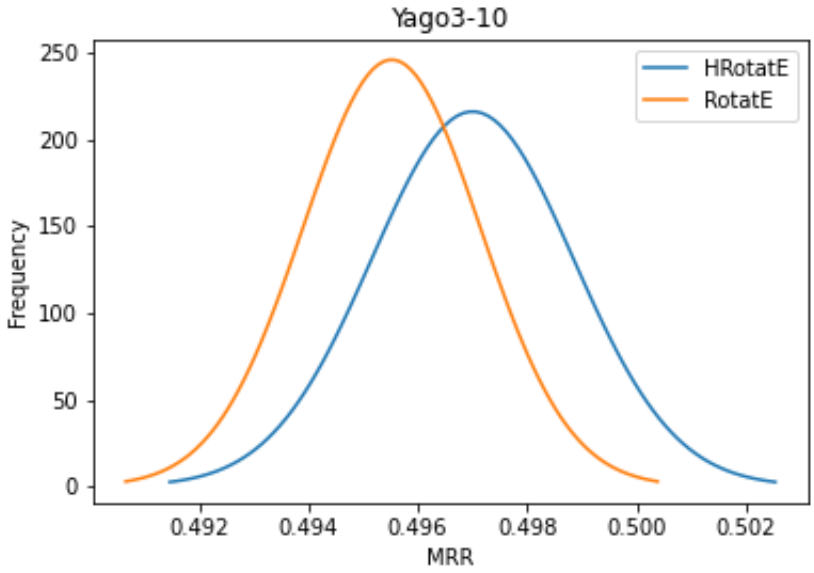


FIGURE 5.5: Normal distribution of the RotatE and HRotatE model on Yago3-10 dataset

5.3 Consistency in Results

We ran the experiment 15 times to ensure the consistency of our proposed approach result. The detailed result summary is displayed in Table 5.6. Based on the small values of standard deviation and variance of the different experiments, we can conclude that our algorithm can generate consistent results in all four measures.

5.4 Comparison with RotatE

Our proposed model, HRotatE, is efficient in learning training parameters; and it improves the result of the MRR by at least 0.002 in all five datasets. Moreover, it performs better compared to state-of-the-art baselines (benchmark models) in the MRR matrix in WN18RR, FB15k, and FB15k-237 dataset. Furthermore, we trained our model, HRotatE, using half of the number of training steps required to train the native RotatE. Thus, HRotatE achieves

nearly the same results as RotatE, and even higher in some datasets, with just half the number of RotatE's training steps. Table 5.7 shows the detailed comparison between HRotatE and RotatE. We got higher MRR scores in WN18, FB15k, and WN18RR datasets even with the half number of training steps.

This chapter examines the experimental results and comparison with state-of-the-art models. We conducted our experiment on five (5) benchmark datasets. We have also performed the statistical significance test (independent two-sample t-test) on all the five benchmark dataset results. In the end, we compared our proposed model (HRotatE) to the RotatE, where our model trained with just half the number of training steps required by the RotatE. The result indicates that our model not only gives a better result but it also converges faster.

Table 5.6: Summary of results

Dataset	Statistics	MRR	Hit@1	Hit@3	Hit@10
WN18	Mean	0.951	0.945	0.954	0.960
	Standard Deviation	0.000381	0.000464	0.000589	0.000666
	Variance	0.000000145	0.000000215	0.000000347	0.000000444
FB15k	Mean	0.799	0.751	0.833	0.832
	Standard Deviation	0.000600	0.000967	0.000644	0.000432
	Variance	0.000000361	0.000000935	0.000000415	0.000000186
WN18RR	Mean	0.483	0.438	0.499	0.572
	Standard Deviation	0.000927	0.001155	0.001275	0.001324
	Variance	0.000000859	0.000001330	0.000001630	0.000001750
FB15k-237	Mean	0.338	0.243	0.373	0.530
	Standard Deviation	0.000517	0.000758	0.001232	0.000815
	Variance	0.000000267	0.000000575	0.000001520	0.000000665
Yago3-10	Mean	0.497	0.399	0.554	0.681
	Standard Deviation	0.001844	0.002862	0.002567	0.001523
	Variance	0.000003400	0.000008190	0.000006590	0.000002320

Table 5.7: Comparison with RotatE

Model	HRotatE					RotatE				
Dataset	MRR	Hit@1	Hit@3	Hit@10	Max Step	MRR	Hit@1	Hit@3	Hit@10	Max Step
WN18	0.950	0.945	0.952	0.959	40,000	0.949	0.944	0.952	0.959	80,000
FB15k	0.797	0.746	0.830	0.884	75,000	0.797	0.746	0.830	0.884	150,000
WN18RR	0.483	0.442	0.497	0.559	40,000	0.476	0.428	0.492	0.571	80,000
FB15k-237	0.334	0.234	0.370	0.525	50,000	0.338	0.241	0.375	0.533	100,000

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We proposed a hybrid approach, namely HRotatE, to solve the Knowledge Graph Completion (KGC) problem. Various approaches exist to predict a missing link in a Knowledge Graph, but the most prominent approaches are based on tensor factorization and Knowledge-Graph embeddings. These approaches can be divided into three subcategories: translation-based approach, bilinear-based approach, and neural-network-based approach. The translation-based approaches use the relative distance existing between constituent entity embedding vectors of the Knowledge Graph (KG). In this approach, entities and relations are embedded into the embedding space, where the relation embedding is generally used to map the head entity to the tail entity. Additionally, the bilinear-based approaches employ tensor factorization with regard to the generation of embedding vectors in the KG. The neural-network-based approaches are essentially nonlinear models that vary according to different deep learning architectures. Each of the aforementioned approaches possesses its respective strengths and weaknesses.

Thus, we proposed the hybridized variant of the existing approaches RotatE and SimplE to solve the KGC problem. RotatE depicts the relation as a rotation between the head and tail entities in the complex vector space. However, In RotatE, the head and tail entities are derived from the same embedding-generation class. Thus, RotatE uses the same value

for an entity regardless of its usage in head embedding or tail embedding. On the other hand, the SimplE model is based on Canonical Polyadic (CP) decomposition. SimplE enhances CP via the addition of the inverse relation, while the head entity and tail entity are derived from different embedding-generation classes, which are interdependent. Hence, we employed the principle of inverse-relation embedding (from the SimplE model) onto the native RotatE model so as to yield a new hybrid resultant: HRotatE. Therefore, HRotatE boasts of efficiency as well as improved prediction scores.

We have evaluated our model, HRotatE, against five (5) benchmark datasets. Our approach achieves better performance on several benchmark datasets. Also, we have compared HRotatE with several state-of-the-art models (inclusive of neural-network-based approaches). We also evaluated the statistical significance of HRotatE performance by conducting an independent two-sample t-test on the Mean Reciprocal Rank (MRR) score of both HRotatE and RotatE in five benchmark datasets. The statistical test showed that there is the statistical significance that HRotatE performs better than RotatE. We have also tested and compared our approach, HRotatE, against RotatE. We show that our model, HRotatE, achieves approximately the same results as RotatE with much lesser (half the number) training steps. Thus, our proposed model can converge faster than the native RotatE model.

6.2 Future Work

In the future, we are planning to test our model on several benchmark datasets. The main drawback of HRotatE, at the moment, is that it cannot predict hierarchical relations effectively. In this regard, our HRotatE model does not perform well on the Yago3-10 dataset. In the near future, our goal is to update the score function so that it can effectively capture and predict the hierarchical relationship. Generally, KG represents a bilinear relationship, and this means that every relation possesses a pair of entities. In the future, we want to test our approach on graphs possessing non-binary relations or knowledge hypergraphs. Also, we are working on adapting our proposed model, HRotatE, for multi-task learning. The current HRotatE model calculates its score by computing the average of the main and inverse scores. However, HRotatE's performance can be improved by tuning the ratios between the

main score and the inverse score. We leave the work of modeling the uncertainties in KGs as our future work. Furthermore, this research can be extend to dynamic KG or temporal KG completion problem.

Bibliography

[1] James Joseph Sylvester. On an application of the new atomic theory to the graphical representation of the invariants and covariants of binary quantics, with three appendices. *American Journal of Mathematics*, 1(1):64–104, 1878.

[2] Bhavanari Satyanarayana and Kuncham Syam Prasad. *Discrete Mathematics and Graph Theory*. PHI Learning Pvt. Ltd., 2014.

[3] Shawn Gu, John Johnson, Fazle E Faisal, and Tijana Milenković. From homogeneous to heterogeneous network alignment via colored graphlets. *Scientific reports*, 8(1):1–16, 2018.

[4] Xander Wilcke, Peter Bloem, and Victor De Boer. The knowledge graph as the default data model for learning on heterogeneous knowledge. *Data Science*, 1(1-2):39–57, 2017.

[5] Tim Berners-Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, and David Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *Proceedings of the 3rd international semantic web user interaction workshop*, volume 2006, page 159. Athens, Georgia, 2006.

[6] Lisa Ehrlinger and Wolfram Wöß. Towards a definition of knowledge graphs. *SEMANTiCS (Posters, Demos, SuCCESS)*, 48:1–4, 2016.

[7] Ahmet Soylu, Oscar Corcho, Brian Elvesæter, Carlos Badenes-Olmedo, Francisco Yedro Martínez, Matej Kovacic, Matej Posinkovic, Ian Makgill, Chris Taggart, Elena Simperl, et al. Enhancing public procurement in the european union through constructing and exploiting an integrated knowledge graph. In *International Semantic Web Conference*, pages 430–446. Springer, 2020.

[8] Amit Singhal. Introducing the knowledge graph: things, not strings. *Official google blog*, 5:16, 2012.

[9] Zhe Chen, Yuehan Wang, Bin Zhao, Jing Cheng, Xin Zhao, and Zongtao Duan. Knowledge graph completion: A review. *IEEE Access*, 8:192435–192456, 2020.

[10] Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge base completion via search-based question answering. In *Proceedings of the 23rd international conference on World wide web*, pages 515–526, 2014.

[11] Bonaventure C. Molokwu, Shaon Bhatta Shuvo, N. Kar, and Ziad Kobti. Link prediction in social graphs using representation learning via knowledge-graph embeddings and convnet (rlvecn). *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 2173–2178, 2020.

[12] Bonaventure Molokwu, Shaon Bhatta Shuvo, Narayan C Kar, and Ziad Kobti. Node classification and link prediction in social graphs using rlvecn. In *32nd International Conference on Scientific and Statistical Database Management*, pages 1–10, 2020.

[13] Mohammad Al Hasan and Mohammed J Zaki. A survey of link prediction in social networks. In *Social network data analytics*, pages 243–275. Springer, 2011.

[14] Kumaran Ragunathan, Kalyani Selvarajah, and Ziad Kobti. Link prediction by analyzing common neighbors based subgraphs using convolutional neural network. In *ECAI*, 2020.

[15] Kalyani Selvarajah, Kumaran Ragunathan, Ziad Kobti, and Mehdi Kargar. Dynamic network link prediction by learning effective subgraphs using cnn-lstm. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[16] Jure Leskovec. Node representation learning, Feb 2020. URL `https://snap-stanford.github.io/cs224w-notes/machine-learning-with-networks/node-representation-learning`.

[17] Baoxu Shi and Tim Weninger. Open-world knowledge graph completion. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

[18] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[19] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *Advances in neural information processing systems*, pages 4284–4295, 2018.

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

[21] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. *Proceedings of the AAAI Conference on Artificial Intelligence*, 29(1), 2015.

[22] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2018.

[23] Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the International Conference on Learning Representations (ICLR) 2015*, May 2015. URL `https://www.microsoft.com/en-us/research/publication/embedding-entities-and-relations-for-learning-and-inference-in-knowledge-bases/`.

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

[25] Maximilian Nickel, Volker Tresp, and Hans-Peter Kmerriegel. A three-way model for collective learning on multi-relational data. In *Icml*, volume 11, pages 809–816, 2011.

[26] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

[27] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. *Advances in neural information processing systems*, 26:926–934, 2013.

[28] Deepak Nathani, Jatin Chauhan, Charu Sharma, and Manohar Kaul. Learning attention-based embeddings for relation prediction in knowledge graphs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4710–4723, 2019.

[29] Ivana Balazevic, Carl Allen, and Timothy Hospedales. Tucker: Tensor factorization for knowledge graph completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5188–5197, 2019.

[30] Frank L Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

[31] Théo Trouillon, Éric Gaussier, Christopher R Dance, and Guillaume Bouchard. On inductive abilities of latent factor models for relational learning. *Journal of Artificial Intelligence Research*, 64:21–53, 2019.

[32] Bonaventure C. Molokwu and Ziad Kobti. Social network analysis using rlvecn: Representation learning via knowledge-graph embeddings and convolutional neural-network. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 5198–5199. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/739. URL `https://doi.org/10.24963/ijcai.2020/739`. Doctoral Consortium.

[33] Zhouxia Wang, Tianshui Chen, Jimmy Ren, Weihao Yu, Hui Cheng, and Liang Lin. Deep reasoning with knowledge graph for social relationship understanding. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 1021–1028, 2018.

[34] Qi He, Jaewon Yang, and Baoxu Shi. Constructing knowledge graph for social networks in a deep and holistic way. In *Companion Proceedings of the Web Conference 2020*, pages 307–308, 2020.

[35] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 353–362, 2016.

[36] Enrico Palumbo, Giuseppe Rizzo, and Raphaël Troncy. Entity2rec: Learning user-item relatedness from knowledge graphs for top-n item recommendation. In *Proceedings of the eleventh ACM conference on recommender systems*, pages 32–36, 2017.

[37] Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. Complex query answering with neural link predictors. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=Mos9F9kDwkz`.

[38] Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander Smola, and Le Song. Variational reasoning for question answering with knowledge graph. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

[39] Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. Knowledge graph embedding based question answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 105–113, 2019.

[40] Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. K-bert: Enabling language representation with knowledge graph. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03):2901–2908, 2020.

[41] KM Annervaz, Somnath Basu Roy Chowdhury, and Ambedkar Dukkipati. Learning beyond datasets: Knowledge graph augmented neural networks for natural language processing. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 313–322, 2018.

[42] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1), 2014.

[43] Ralph Abboud, Ismail Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. Boxe: A box embedding model for knowledge base completion. In H. Larochelle,

M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9649–9661. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/6dbbe6abe5f14af882ff977fc3f35501-Paper.pdf.

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

[45] Ledyard R Tucker et al. The extension of factor analysis to three-dimensional matrices. *Contributions to mathematical psychology*, 110119, 1964.

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

[47] Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. A novel embedding model for knowledge base completion based on convolutional neural network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 327–333, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2053. URL https://www.aclweb.org/anthology/N18-2053.

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

[49] David Stipp. *A most elegant equation: Euler's formula and the beauty of mathematics*. Hachette UK, 2017.

[50] John H Conway and Richard Guy. *The book of numbers*. Springer Science & Business Media, 1998.

[51] C Edward Sandifer. *How Euler did it*, volume 3. MAA, 2007.

[52] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. 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. URL `https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf`.

[53] Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(2):1–49, 2021.

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

[55] Guido van Rossum and Python Dev Team. *Python 3.6 Language Reference*. Samurai Media Limited, London, GBR, 2016. ISBN 9789888406883.

[56] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, 2008.

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

[58] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd workshop on continuous vector space models and their compositionality*, pages 57–66, 2015.

[59] Dirk Taeger and Sonja Kuhnt. *Statistical hypothesis testing with SAS and R*. Wiley Online Library, 2014.

[60] Erich L Lehmann and Joseph P Romano. *Testing statistical hypotheses*. Springer Science & Business Media, 2006.

[61] S William. The probable error of a mean. *Biometrika*, 6(1):1–25, 1908.

[62] Zhen Tan, Xiang Zhao, Yang Fang, Bin Ge, and Weidong Xiao. Knowledge graph representation via similarity-based embedding. *Scientific Programming*, 2018, 2018.

[63] Timothée Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In *International Conference on Machine Learning*, pages 2863–2872. PMLR, 2018.

[64] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. Holographic embeddings of knowledge graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30 (1), 2016.

[65] Hanxiao Liu, Yuexin Wu, and Yiming Yang. Analogical inference for multi-relational embeddings. In *International conference on machine learning*, pages 2168–2178. PMLR, 2017.

[66] Takuma Ebisu and Ryutaro Ichise. Toruse: Knowledge graph embedding on a lie group. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

Appendices

Appendix A: HRotatE - Uniform Sampling

As mentioned in Chapter 3, the positive triples are the ground-truth triples given in the dataset [24, 52]. However, to train the modern efficiently, there needs to generate a negative sample. Generally, there are two popular ways to generate the negative sample: the Uniform way and the Adversarial Way. BoxE [43] and RotatE [22] show the advantages of adversarial sampling method over uniform method. In this regard, herein, we used the same self-adversarial negative sampling, as employed in RotatE [22, 52], to generate negative samples for training our model, HRotatE. However, to show the model performance, we also tested our approach with uniform sampling. In this section, we compare our approach with RotatE in uniform sampling. The final loss function for uniform sampling is shown in equation 1.

$$L = -\log \sigma(\gamma - d_r(h,t)) - \log \sigma(d_r(h'_i, t'_i) - \gamma) \tag{1}$$

Table 1: Comparison of RotatE and HRotatE (Uniform Sampling)

Dataset	Model	MRR	Hit@1	Hit@3	Hit@10
WN18	HRotatE	0.947	0.938	0.953	0.962
	RotatE	0.946	0.937	0.952	0.961
FB15k	HRotatE	0.707	0.614	0.774	0.860
	RotatE	0.699	0.584	0.789	0.872
WN18RR	HRotatE	0.474	0.428	0.492	0.593
	RotatE	0.471	0.424	0.490	0.565
FB15k-237	HRotatE	0.301	0.208	0.332	0.488
	RotatE	0.294	0.201	0.326	0.479
Yago3-10	HRotatE	0.454	0.357	0.502	0.645
	RotatE	0.448	0.347	0.496	0.647

Table 1 illustrates the result of the RotatE and HRotatE approach, which uses uniform sampling to generate the negative sample. We have used the same hyper-parameters that we used in self adversarial negative sampling. The result of uniform sampling is low, which satisfies the claim proposed by the RotatE [22], and BoxE [43]. However, this result can be improved by tuning the hyper-parameters.

Appendix B: Proof of the Inference Pattern

This appendix describes the proof of inference pattern discussed in Section 1.6 and Table 2.1. First, we will illustrate that how RotatE [22] can learn inference patter. Then, we will extend the proof to HRotatE.

Symmetry: if (e_i, r, e_j) and (e_j, r, e_i) hold, then a RotatE model makes

$$e_j = r \circ e_i \wedge e_i = r \circ r_j \implies r \circ r = 1$$

If $(e_i, r, e_j) \in \zeta$, then a HRotatE model makes (h_{e_i}, v_r, t_{e_j}) and $(h_{e_j}, v_r^{-1}, t_{e_i})$ positive. By, tying the parameters of v_r^{-1} to v_r, we can conclude that (h_{e_j}, v_r, t_{e_i}) and $(h_{e_i}, v_r^{-1}, t_{e_j})$ is also become positive. Therefore HRotatE can predicts $(e_j, r, e_i) \in \zeta$.

Anti-Symmetry: if (e_i, r, e_j) and $\neg(e_j, r, e_i)$ hold, then a RotatE model makes

$$e_j = r \circ e_i \wedge e_i \neq r \circ r_j \implies r \circ r \neq 1$$

If $(e_i, r, e_j) \in \zeta$, then a HRotatE model makes (h_{e_i}, v_r, t_{e_j}) and $(h_{e_j}, v_r^{-1}, t_{e_i})$ negative. By, tying the parameters of v_r^{-1} to v_r, we can conclude that (h_{e_j}, v_r, t_{e_i}) and $(h_{e_i}, v_r^{-1}, t_{e_j})$ is also become negative. Therefore HRotatE can predicts $(e_j, r, e_i) \in \zeta'$.

Inversion: if (e_i, r_1, e_j) and (e_j, r_2, e_i) hold, then a RotatE model makes

$$e_j = r_1 \circ e_i \wedge e_i = r_2 \circ r_j \implies r_1 = r_2^{-1}$$

If $(e_i, r_1, e_j) \in \zeta$, then a HRotatE model makes $(h_{e_i}, v_{r_1}, t_{e_j})$ and $(h_{e_j}, v_{r_1}^{-1}, t_{e_i})$ positive. By, tying the parameters of $v_{r_2}^{-1}$ to v_{r_1} and $v_{r_1}^{-1}$ to v_{r_2}, we can conclude that $(h_{e_j}, v_{r_2}, t_{e_i})$ and $(h_{e_i}, v_{r_2}^{-1}, t_{e_j})$ is also become negative. Therefore HRotatE can predicts $(e_j, r_2, e_i) \in \zeta$.

Composition: if (e_i, r_1, e_K), (e_i, r_2, e_j) and (e_j, r_3, e_k) hold, then a RotatE model makes

$$e_k = r_1 \circ e_i \wedge e_j = r_2 \circ e_i \wedge e_k = r_3 \circ e_j \implies r_1 = r_2 \circ r_3$$

If $(e_i, r_2, e_j) \in \zeta$, then a HRotatE model makes $(h_{e_i}, v_{r_2}, t_{e_j})$ and $(h_{e_j}, v_{r_2}^{-1}, t_{e_i})$ positive. and If $(e_j, r_3, e_k) \in \zeta$, then a HRotatE model makes $(h_{e_j}, v_{r_3}, t_{e_k})$ and $(h_{e_k}, v_{r_3}^{-1}, t_{e_j})$ positive. By, tying the parameters of v_{r_1} to v_{r_2} and v_{r_3}, we can conclude that $(h_{e_i}, v_{r_1}, t_{e_k})$ and $(h_{e_k}, v_{r_1}^{-1}, t_{e_k})$ is also become negative. Therefore HRotatE can predicts $(e_i, r_1, e_k) \in \zeta$.

Other Models: TransE [20] is not able to predict symmetric patterns. The reason for this is: $\forall e_i, e_j \in \mathcal{E}$ that satisfies $(e_i, r, e_j) \in \zeta$, $(e_j, r, r - i) \in \zeta$ must be true, which means TransE makes $\|e_i + r - e_j\| \approx 0$ and $\|e_j + r - e_i\| \approx 0$. Thus, TransE forcefully makes the $r = 0$ and $e_i = e_j$, which converts the symmetric relations into reflexive relations. However, e_i and e_j are different entities. Hence, they can not have the same value. BoxE[43] illustrates that TransE and RotatE are not able to predict hierarchical patterns because they enforce the relation equivalence.

TransH [42] cannot infer composition and inversion pattern due to their invertible matrix multiplications, and its symmetric nature [22]. Due to the full-rank relation matrix, DistMult [23] is not able to differentiate between the source entity and the target entity. For the given triplet (e_i, r, e_j) and the opposite triplet (e_j, r, e_i), DistMult assigns the same score for both triplets because its unable to differentiate entity in head or tail. The same reason can be applied to the inverse relation. ComplEx [24], DistMult [23] and TuckER [45] are not able to predict composition rules because they does not model a bijection mapping from the source node (head) to the destination node (tail) by relation.

Vita Auctoris

NAME:	Akshay Shah
PLACE OF BIRTH:	Gujarat, India
YEAR OF BIRTH:	1997
EDUCATION:	Bachelor of Engineering in Information Technology, Gujarat Technological University, Gujarat, India
	Master of Science in Computer Science, University of Windsor, Ontario, Canada