Doctoral Dissertation

Learning Continuous Representations for Knowledge Graphs

A dissertation presented
by
Caglar Demir
to the
Faculty for Computer Science,
Electrical Engineering and Mathematics
of
Paderborn University

in partial fulfillment of the requirements
for the degree of
Dr. rer. nat.
Paderborn, Germany
March 2023

Dissertation

Learning Continuous Representations for Knowledge Graphs
Caglar Demir, Paderborn University
Paderborn, Germany, 2023

Reviewers

Prof. Dr. Axel-Cyrille Ngonga Ngomo, Paderborn University
Prof. Dr. Sören Auer, Hannover University
Jun.-Prof. Dr. Sebastian Peitz, Paderborn University

Doctoral Committee

Prof. Dr. Axel-Cyrille Ngonga Ngomo, Paderborn University
Prof. Dr. Yasemin Acar, Paderborn University
Prof. Dr. Patricia Arias Cabarcos, Paderborn University
Prof. Dr. Sören Auer, Hannover University
Jun.-Prof. Dr. Sebastian Peitz, Paderborn University

Abstract

Learning Continuous Representations for Knowledge Graphs

For the last two decades, we have been witnessing a technological revolution, where Deep Learning undoubtedly plays a pivotal role. The idea of learning continuous vector representations for inputs has been instrumental in many recent scientific and industrial success stories.

This thesis is concerned with learning continuous vector representations for knowledge graphs. Although graph-structured data is ubiquitous in the nature, most machine learning algorithms cannot be directly applied on such discrete data. A feature engineering process is necessary to leverage most machine learning algorithms on learning problems defined over graph-structured data. Yet, this process is often is costly, arduous and sometimes even infeasible. In this thesis, we propose eight knowledge graph embedding models (Pyke, Shallom, ConEx, QMult, OMult, ConvQ, ConvO, and NeRo) that learn continuous vector representations to improve the state-of-the-art performance in benchmark tasks such as relation prediction, link prediction, and description logic concept learning. Through learning continuous representations, machine learning algorithms become amenable to graph-structured data. Additionally, we introduce a decomposition technique (KronE) that improves a model's parameter efficiency by learning compressed continuous vector representations. We also present a parameter ensemble technique (PPE) to boosts the generalization performance of a model in link prediction with virtually no additional cost. Finally, we demonstrate an industrial and scientific use cases based on our scientific and software contributions.

We conclude this with our future research directions that center around applying the Erlangen Programme mindset to unify "a veritable zoo of knowledge graph embedding models" on static, temporal and dynamic knowledge graphs.

Zusammenfassung

Learning Continuous Representations for Knowledge Graphs

In den letzten zwei Jahrzehnten haben wir eine technologische Revolution erlebt, bei der Deep Learning zweifellos eine zentrale Rolle spielt. Die Idee von lernenden kontinuierlichen Vektordarstellungen für Eingaben hat in den letzten Jahren zu vielen wissenschaftlichen und industriellen Erfolgen beigetragen.

Diese Arbeit befasst sich mit dem lernenden kontinuierlichen Vektordarstellungen für Wissensgraphen. Obwohl graphenstrukturierte Daten in der Natur allgegenwärtig sind, können die meisten Algorithmen für maschinelles Lernen nicht direkt auf solche diskreten Daten angewendet werden. Zur Lösung von Lernproblemen, die über graphenstrukturierte Daten definiert sind, ist daher ein Feature-Engineering-Prozess erforderlich, um die meisten maschinellen Lernalgorithmen zu nutzen. Dieser Prozess ist jedoch oft kostspielig, mühsam und manchmal sogar undurchführbar. In dieser Arbeit schlagen wir acht Modelle zur Einbettung von Wissensgraphen vor, die kontinuierliche Vektordarstellungen erlernen, um die State-of-the-Art-Leistung bei Benchmark-Aufgaben wie der Vorhersage von Beziehungen, der Vorhersage von Links und dem Lernen von Beschreibungslogikkonzepten zu verbessern. Durch lernende kontinuierliche Repräsentationen werden Algorithmen des maschinellen Lernens für grafisch strukturierte Daten zugänglich. Darüber hinaus stellen wir eine Dekompositionstechnik vor, die die Parametereffizienz eines Modells durch das Lernen komprimierter kontinuierlicher Vektordarstellungen verbessert. Außerdem stellen wir eine Parameter-Ensemble-Technik vor, die die Generalisierungsleistung eines Modells bei der Vorhersage von Verbindungen praktisch ohne zusätzliche Kosten steigert. Schließlich demonstrieren wir jeweils ein industrielles und ein wissenschaftliches Anwendungsbeispiel auf der Grundlage unserer wissenschaftlichen und softwaretechnischen Beiträge.

Abschließend geht es um zukünftige Forschungsanschlüsse, die sich auf die Anwendung der Denkweise des Erlangener Programms konzentrieren, um „einen wahren Zoo von Modellen zur Einbettung von Wissensgraphen“ auf statische, temporale und dynamische Wissensgraphen zu vereinheitlichen.

Acknowledgments

First of all, I would like to thank my family: Vanessa Demir, Saliha Demir, Öztekin Demir, Ümit Demir, and Anıl Öztürk. Without their unconditional love and support, this thesis would not have been possible.

I would like to wholeheartedly thank Prof. Dr. Axel-Cyrille Ngonga Ngomo. Without his guidance and constant support, our scientific contributions comprising this thesis would not have been possible. It has been a delight to share his intrinsic motivation to do a research for the sake of the research. I am grateful for having been his student.

Finally, I would like to thank to Dr. Diego Campos Moussallem, Dr. Stefan Heindorf, Dr. Pamela Heidi Douglas, Dr. Michael Röder, Dr. Mohamed Ahmed Sherif, N'dah Jean Kouagou, Alexander Bigerl, Arnab Sharma, Julian Lienen, Simon Bin, Anna Himmelhuber, Yushan Liu, and again Prof. Dr. Axel-Cyrille Ngonga Ngomo for our scientific discussions.

List of Publications and Declaration of Authorship

Physical Embedding Model for Knowledge Graphs

by Caglar Demir and Axel-Cyrille Ngonga Ngomo was accepted at the 9th Joint International Semantic Technology Conference 2019.

Declaration of Authorship: The original idea was introduced by Axel-Cyrille Ngonga Ngomo. The research contributions were further developed by Caglar Demir and discussed with Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Axel-Cyrille Ngonga Ngomo. A brief summary of this publication is provided in Section 1.3.1. This publication is presented in Chapter 3.

A shallow neural model for relation prediction

by Caglar Demir, Diego Moussallem, and Axel-Cyrille Ngonga Ngomo was accepted at the 15th IEEE International Conference on Semantic Computing 2021.

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Diego Moussallem and Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Diego Moussallem and Axel-Cyrille Ngonga Ngomo. A brief summary of this publication is provided in Section 1.3.2. This publication is presented in Chapter 4.

Convolutional Complex Knowledge Graph Embeddings

by Caglar Demir and Axel-Cyrille Ngonga Ngomo was accepted at the 17th European Semantic Web Conference 2021.

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Axel-Cyrille Ngonga Ngomo. A brief summary of this publication is provided in Section 1.3.3. This publication is presented in Chapter 5.

Convolutional Hypercomplex Embeddings for Link Prediction

by Caglar Demir, Diego Moussallem, Stefan Heindorf, Axel-Cyrille Ngonga Ngomo was accepted at the 13th Asian Conference on Machine Learning 2021.

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Diego Moussallem Stefan Heindorf, and Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Stefan Heindorf, Diego Moussallem, and Axel-Cyrille Ngonga Ngomo. A brief summary of this publication is provided in Section 1.3.4. This publication is presented in Chapter 6.

Kronecker Decomposition for Knowledge Graph Embeddings

by Caglar Demir, Julian Lienen, and Axel-Cyrille Ngonga Ngomo was accepted at the 33rd Conference on Hypertext and Social Media.

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Julian Lienen, and Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Julian Lienen, and Axel-Cyrille Ngonga Ngomo. A brief summary of this publication is provided in Section 1.3.5. This publication is presented in Chapter 7.

Polyak Parameter Ensemble for Knowledge Graph Embeddings

by Caglar Demir and Axel-Cyrille Ngonga Ngomo has been submitted to the 29th Conference on Knowledge Discovery and Data Mining.

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Axel-Cyrille Ngonga Ngomo. A brief summary of this publication is provided in Section 1.3.6. This publication is presented in Chapter 8.

Learning Permutation-Invariant Embeddings for Description Logic Concepts

by Caglar Demir and Axel-Cyrille Ngonga Ngomo was accepted at the 21th International Symposium on Intelligent Data Analysis 2023.

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Axel-Cyrille Ngonga Ngomo. A brief summary of this publication is provided in Section 1.3.7. This publication is presented in Chapter 9.

Hardware-agnostic computation for large-scale knowledge graph embeddings

by Caglar Demir and Axel-Cyrille Ngonga Ngomo was accepted at the Software Impacts Journal 2022.

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the framework, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Axel-Cyrille Ngonga Ngomo. This publication and the developed framework is presented in Chapter 10.

Rapid Explainability for Skill Description Learning

by Caglar Demir, Anna Himmelhuber, Yushan Liu, Alexander Bigerl, Diego Moussallem, and Axel-Cyrille Ngonga Ngomo was accepted at the 21st International Semantic Web Conference.

Declaration of Authorship: The original research idea was introduced by Axel-Cyrille Ngonga Ngomo in the BMWi-funded project RAKI (01MD19012D). Caglar Demir implemented the framework containing the inductive logic programming and the deep reinforcement learning modules, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and all authors revised it. This publication is presented in Chapter 13 as an industrial use case.

Deep Reinforcement Learning for Class Expression Learning in $\mathcal{ALC}$

by Caglar Demir and Axel-Cyrille Ngonga Ngomo has been submitted to the 32nd International Joint Conference On Artificial Intelligence 2023.

Declaration of Authorship: The original research idea was introduced by Axel-Cyrille Ngonga Ngomo. The original research contributions were developed by Caglar Demir and discussed with Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Axel-Cyrille Ngonga Ngomo. This publication is presented in Chapter 12 of this thesis.

Contents

Part I

Introduction

1 Introduction

This thesis is concerned with learning continuous vector representations for Knowledge Graphs (KGs). Our research interest centers around designing Knowledge Graph Embedding (KGE) models to effectively tackle benchmark tasks on large KGs. This thesis comprises four parts. Part I introduces the idea of learning continuous vector representations for KGs and elaborates current challenges along with our research questions. Part II presents our seven scientific contributions. Part III presents our software contributions and two use cases. Finally, Part IV concludes this thesis with a discussion and an overview of our ongoing works.

1.1 Motivation

For the last two decades, we have been witnessing a technological revolution, where Deep Learning (DL) undoubtedly plays a pivotal role. The 2018 Turing Award recipients Yann LeCun, Yoshua Bengio, and Geoffrey Hinton along with their peers have repeatedly shown that learning continuous vector representations has led to substantial improvements over using hand-crafted representations in many challenging tasks, including image recognition, machine translation, question answering, and sequential decision making (Krizhevsky et al., 2012; Mikolov et al., 2013b; Sutskever et al., 2014; Mnih et al., 2015; Silver et al., 2016; Vaswani et al., 2017; Devlin et al., 2018).[1]

[1] https://awards.acm.org/about/2018-turing accessed on the 21th of March, 2023.

Creating hand-crafted representations for inputs can be seen as a means to incorporate domain knowledge to extract discriminative information from inputs for a given learning problem (Bengio et al., 2013). Yet, this process is costly, arduous and sometimes even infeasible (Goodfellow et al., 2016). On the other hand, DL models can be trained without requiring hand-crafted representations (Bengio et al., 2013; LeCun et al., 2015, 2002). Although applying feature engineering was previously a standard technique, learning continuous vector representations has been consistently yielding state-of-the-art performances in the aforementioned tasks among many others (Jumper et al., 2021; Fawzi et al., 2022).

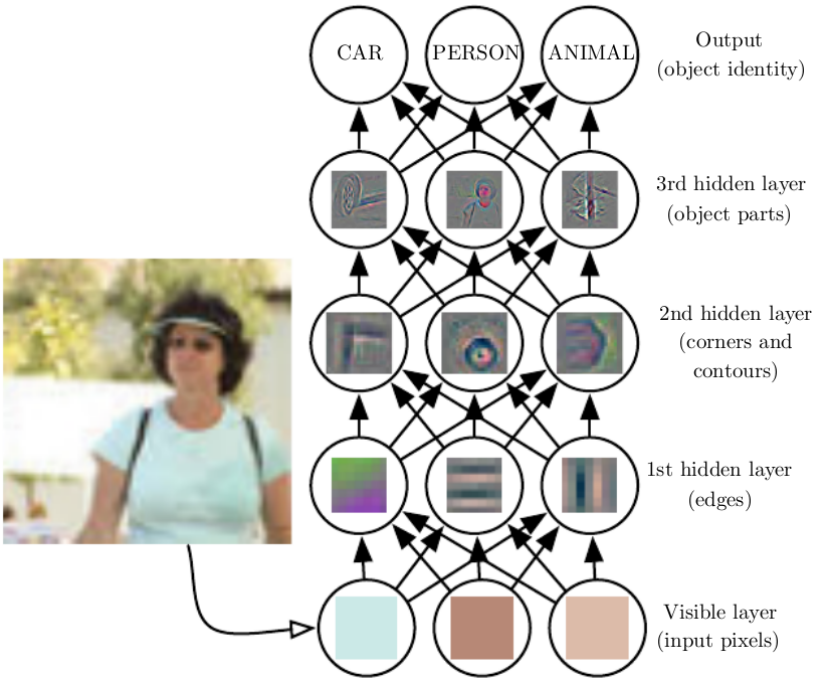


Figure 1.1: Learning hidden representations for images in an object recognition task. The image is taken from Goodfellow et al. (2016).

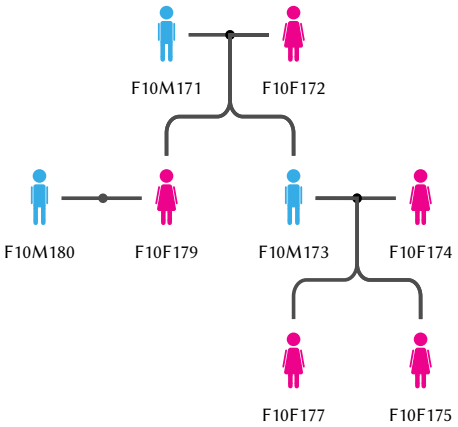
In Figure 1.1, we visualize a process of learning *hidden* continuous vector representations for images in an object recognition task. During training, parameters of this five-layered Convolutional Neural Network (CNN) are iteratively updated in the negative direction of the gradient of the loss function w.r.t. parameters. By this, true class labels of images can be accurately predicted directly from raw pixel values.

In this thesis, we focus on learning continuous vector representations (also called *embeddings*) for KGs. A KG represents a structured collection of assertions in the form of typed relationships between entities (Hogan et al., 2020). An assertion in a KG is a triple containing two entities (also called nodes) and a relation (also called an edge) (more details in Section 2.3.1).[2] These collections of assertions have been used in a wide range of applications, including question answering, machine translation, product recommendation, web search, and cancer research (Kulmanov et al., 2018; Saleem et al., 2014; Shen et al., 2014; Moussallem et al., 2018; Noy et al., 2019; Hogan et al., 2020). Yet, most Machine Learning (ML) algorithms cannot be directly applied on these valuable resources, since they are not designed to process graph-structured data (see Section 2.1.2). Consequently, feature engineering was previously a necessary step to apply most ML algorithms to graph-related problems. A simple remedy was to represent inputs with one-hot binary vector representations. For instance, Rumelhart et al. (1986) evaluated their now very well-known algorithm (the *backpropagation algorithm*) in a task similar to the link prediction problem, where a five-layered Neural Network (NN) was trained to predict an entity given one-hot binary vector representations of an entity and a relation. Therein, an entity and a relation are represented by a 24-dimensional and a 12-dimensional one-hot binary vector, respectively. During training, parameters of the five-layered NN were updated via the backpropagation algorithm to predict an entity from a given concatenated discrete representations of an entity and a relation. Such a discrete representation for inputs loses its applicability as the size of the unique nodes/entities and edges/relations in a graph-structured data grows. For instance, Bengio et al. (2000) show that learning a joint probability distribution over a large number of discrete variables becomes more challenging as the number of unique discrete variables grows. This partially stems from the fact that the Euclidean distance between one-hot binary vector representations of any two different entities is always same, i.e., $\sqrt{2}$. To mitigate this limitation among few others, Bengio et al. (2000) propose to learn a joint probability distribution over many discrete variables by simultaneously learning their continuous vector representations. Instead of using fixed one-hot binary vectors, discrete inputs are represented by randomly initialized continuous vectors. During training, such continuous vector representations are iteratively updated with the goal of minimizing a specified loss function.

[2]We use the term of entity to denote a node and the term of relation to denote an edge.

Their seminal work initiated word embedding/language models (e.g., Word2Vec, Glove, BERT, and GPT-3 (Mikolov et al., 2013b; Pennington et al., 2014; Devlin et al., 2018; Brown et al., 2020)) as well as KGE models (e.g., RDF2Vec and KG-Bert (Ristoski and Paulheim, 2016; Yao et al., 2019)). The idea of learning embeddings for discrete variables has now become a standard method for addressing difficult problems in various research areas. For instance, for the last decade, KGE models have been successfully applied to tackle a wide range of graph-related problems, including entity classification, community detection, link prediction, graph summarization, fact checking, and description logic concept learning (Bordes et al., 2013; Nickel et al., 2015; Cai et al., 2018; Ji et al., 2020; Zhou et al., 2020; Firmansyah et al., 2021; Silva et al., 2021; Kouagou et al., 2022b). Designing an effective KGE model in a benchmark task (e.g. link prediction), while retaining its a parameter efficiency is still an open question (Trouillon et al., 2016; Zhang et al., 2019; Balazevic et al., 2019). This non-trivial endeavor is the foundation for the first two research questions addressed this thesis (see Sections 1.2.1 and 1.2.2).

KGE models learn embeddings for inputs in a fashion akin to language embedding models, since both types of models learn embeddings for discrete inputs. Yet, from a data modeling perspective, KGE models can differ from language embedding models with an aspect. Specifically, a KG can explicitly contain a description of the world via a set of terminological axioms as shown in Figure 1.2.



TBox:
Brother ⊑Male
Brother ⊑ PersonWithASibling
Child ⊑ Person
Daughter ⊑ Child, Daughter ⊑ Female
Father ⊑ Male, Father ⊑ Parent
Female ⊑ Person
Grandchild ⊑ Child
Granddaughter ⊑ Female
Granddaughter ⊑ Grandchild
Grandfather⊑ Grandparent
Grandfather⊑ Male
Grandmother⊑Female
Grandmother⊑ Grandparent
Grandparent⊑Parent
Grandson⊑Grandchild,Grandson⊑Male
Male ⊑ Person
Mother ⊑ Person, Mother ⊑ Parent
Parent ⊑ Person
PersonWithASibling ⊑ Person
Sister ⊑ Female
Sister ⊑ PersonWithASibling
Son ⊑ Child, Son ⊑ Male

Figure 1.2: A visualization of the Family KG. Colors denote Male or Female class memberships assertions, while (·) and branching from (·) denote role assertions.

Although KGE research has mainly focused on designing KGE models without incorporating terminological axioms explicitly, there is a growing interest in designing models to learn embeddings for such logical expressions (Ren et al., 2020; Mondal et al., 2021; Zhang et al., 2022). Designing KGE models to learn learning embeddings for logical expressions (e.g., description logic concepts) constitutes the third and last research question addressed in this thesis (see Section 1.2.3).

1.2 Research Questions and Challenges

In the last decade, a plethora of KGE models has been proposed (Nickel et al., 2011; Bordes et al., 2013; Nickel et al., 2015; Trouillon et al., 2016; Wang et al., 2017; Cai et al., 2018; Balazevic et al., 2019; Ji et al., 2020; Zhu et al., 2021). Overall, KGE research has mainly focused on learning embeddings for entities and relations tailored towards the link prediction problem (see Section 2.3.2). Since a KG can contain billions of triples and millions of entities, the computation and parameter efficiency aspects of a KGE model play important roles in large-scale applications (Dettmers et al., 2018; Ren et al., 2022). Designing a KGE model to accurately predict missing triples on large KGs is still a major open question. Moreover, there is an ever growing interest in designing KGE models to learn embeddings for logical expressions, e.g., concepts (Mondal et al., 2021) in the DL $\mathcal{EL}^{++}$. In Sections 1.2.1 to 1.2.3, we elaborate the major challenges and articulate the research questions addressed in this thesis.

1.2.1 Effectiveness in Benchmark Tasks

Most KGs are incomplete, i.e., they contain missing triples (Nickel et al., 2015; Hogan et al., 2020). For instance, an assertion about the citizenship of Barack Obama (e.g. (`dbr:Barack_Obama`, `dbo:citizenship`, `dbr:United_States`)) is still missing in the most recent version of the DBpedia knowledge graph .[3] Consequently, accurately predicting missing triples is an important ability to leverage structured assertions in numerous applications, including approximate query answering over knowledge graphs (Trouillon et al., 2016; Hamilton et al., 2018; Ren and Leskovec, 2020; Arakelyan et al., 2021). Since designing effective KGE models for large KGs is still a major open question. Hence, we pose our first research question:

[3]https://dbpedia.org/page/Barack_Obama accessed on the 21th of March, 2023.

RQ1. Can we design a knowledge graph embedding model to predict missing triples on large knowledge graphs?

To address this research question, we investigate various techniques to design KGE models that advance the state of the art in benchmark tasks, including type prediction, relation prediction, and link prediction. Increasing the effectiveness in a benchmark task without increasing the number of parameters plays a central role in our investigation. Moreover, we also investigate techniques to further increase the effectiveness of KGE models with virtually no additional costs, e.g., applying ensemble learning, Polyak averaging, and designing a semantics based prediction constraint. In Chapters 3 to 8, we present our KGE models advancing the state of the art in the aforementioned benchmark tasks.

1.2.2 Efficiency in Parameters

According to the 2021 crawl of WebDataCommons, a giant joint KG containing at least 82 billion assertions can be extracted from the Web.[4] Learning on such a valuable resource at scale is hence crucial for the deployment of ML models on the Web—the world's largest shared information source with over 5 billion users. Consequently, designing efficient KGE models w.r.t. the number of parameters as well as computational costs is important in the aforementioned applications. Hence, we pose our second research question:

RQ2 Can we answer RQ1. while ensuring the parameter efficiency of our models?

To answer this question, we incorporate the parameter efficiency aspect in the design of our KGE models. For instance, we investigate learning complex-, quaternion-, and octonion-valued embeddings as well as leveraging a convolution operation to increase the parameter efficiency without decreasing the effectiveness (see Chapters 5 and 6). Importantly, we focus on designing KGE models that have linear time and space complexities in the embedding vector dimensions and the number of entities and/or relations. We also design a technique based on the Kronecker decomposition to increase the parameter efficiency of a KGE model, while retaining its effectiveness in the link prediction problem (see Chapter 7).

[4] http://webdatacommons.org/structureddata/#results-2021-1 accessed on the 21th of March, 2023.

We design a new training strategy to facilitate training KGE on large KGs (see Chapter 8). Finally, we develop a hardware-agnostic open-source software framework to facilitate large-scale applications of KGE models (see Chapter 10) without requiring extensive domain knowledge.

1.2.3 Learning Embeddings for Description Logic Concepts

Learning embeddings for First-Order Logic (FOL) queries to obtain approximate answers on incomplete KGs led significant improvements over traditional symbolic models (Hamilton et al., 2018; Ren et al., 2020; Ren and Leskovec, 2020; Arakelyan et al., 2021; Huang et al., 2022; Ren et al., 2022). Although Description Logics (DLs) are decidable fragments of FOL (Baader et al., 2003), learning embeddings for *expressive* DLs (e.g. $\mathcal{ALC}$) has not yet been extensively studied. Hence, we investigate KGE models to learn embeddings for $\mathcal{ALC}$ concepts tailored towards the DL concept learning problem. Concept Learning (CL) deals with learning DLs concepts from a background knowledge and input examples (see Section 2.5). Although symbolic and hybrid models have been successfully applied to tackle this problem, their large-scale applications have been hindered due to their impractical runtimes (see Section 2.5.3). Tackling CL at large scale (e.g. achieving low runtimes on large datasets) is an important problem since learning in DLs is ante-hoc and globally explainable (Heindorf et al., 2022), hence it has a potential for being a backbone for explainable Artificial Intelligence (AI) (Schockaert et al., 2021b). We pose our third research question:

RQ3. Can we design a model to learn embeddings for expressive DLs?

To answer this question, we leverage the knowledge acquired during our investigation of RQ1 and RQ2. For instance, in Chapter 9, we reformulate the CL problem as a multi-label classification problem and use a permutation-invariant neural embedding model. By learning permutation-invariant embeddings for $\mathcal{ALC}$ concepts, a goal concept can be found withing few retrieval operations. This significantly reduces total runtimes. Therefore, the CL problem can be tackled in applications requiring low latency. In Chapter 12, we introduce a scientific use case of leveraging a pre-trained KGE to tackle CL. Therein, we reformulate the definition of the standard symbolic search procedure as a sequential decision making problem in an embedding vector space and leverage deep Q-network to alleviated the impractical runtimes of CL models.

Treating myopic heuristic functions of CL models via a deep Q-network results in reducing total runtimes significantly. Finally, we develop an open-source software framework to facilitate large-scale applications of CL models (see Chapter 11).

1.3 Thesis Overview and Summary of Scientific Publications

In this section, we firstly give an overview of the structure of this thesis. Next, we summarize our scientific contributions, while elucidating relationships between them. Part I consists of two chapters. In Chapter 1, we motivate the idea of learning continuous vector representations for KGs and pose our research questions. In Chapter 2, we provide the background knowledge to make this thesis self-contained. In Part II, we present our scientific contributions. In Part III, we present our software contributions along with use cases. In Part IV, we conclude this thesis with a discussion and an overview of our ongoing works.

1.3.1 A Physical Embedding Model for Knowledge Graphs

In Chapter 3, we propose Pyke. Therein, our goal is to efficiently learn embeddings for entities and relations of a KG such that *similar* entities and relations are assigned with *similar* embeddings, i.e., so that the Euclidean distance between embeddings of similar entities and relations is low. To this end, we define Pyke as a combination of a physical model based on Hooke's law and its inverse with ideas from simulated annealing to efficiently learn embeddings. We prove that Pyke achieves a linear space complexity in the number of entities and relations and in the number of embedding dimensions. Although the time complexity for the initialization of Pyke is quadratic in the number of unique entities and relations, the time complexity of each of its iterations is linear in the size of the input KG. Pyke allows practitioners to incorporate their domain knowledge in the learning process by means of selecting a similarity function between entities and relations. We compare the performance of Pyke against six state-of-the-art KGE models on two benchmark datasets. Our experimental results suggest that Pyke outperforms six state-of-the-art models on benchmark datasets in the type prediction and cluster purity tasks, while maintaining superior runtimes. Our implementation and results are open-source and are available at http://github.com/dice-group/PYKE and https://github.com/dice-group/dice-embeddings. Our first scientific contribution serves as Chapter 3.

Trade-offs: The construction of a similarity matrix can be a bottleneck to apply Pyke on large KGs. This can be alleviated by samplings entities and relations uniformly at random at each parameter update. Yet, this prohibits the incorporation of domain knowledge in the learning process by means of selecting a similarity function. Although Pyke can be applied to efficiently learn embeddings for entities and relations, these embeddings can only be directly used to measure similarities between entities and relations. Hence, Pyke cannot predict missing triples on KGs, i.e., cannot tackle the link prediction or relation prediction problems *directly*. It is necessary to train a classifier (e.g., logistic regression (see Section 2.1.2)) on embeddings of entities and relations to predict missing triples. Although there have been KGE models proposed to solely learn embeddings for entities and relation (e.g., RDF2Vec (Ristoski and Paulheim, 2016), NBFNet (Zhu et al., 2021)), KGE research has mainly focused on KGE models with predictive abilities (see Section 2.4).

1.3.2 A Shallow Neural Model for Relation Prediction

In Chapter 4, we propose Shallom. We design this approach to predict missing relations given entities, while maintaining an efficient computation as in Pyke. To this end, we design the process of learning embeddings as a multi-label classification problem. For a given pair of entities, Shallom predicts missing relations via two affine transformations with non-linear activation functions. Hence, Shallom is analogous to the C-BOW variant of the well-known word embedding model Word2Vec (Mikolov et al., 2013a,b), since both model predict a central vocabulary term given surrounding terms. While a central term in the C-BOW model represents a word, a central term in Shallom represents a relation, hence surrounding terms represent entities. We evaluate Shallom on five benchmark datasets (WN18RR, FB15K-237, YAGO3-10, WN18, and FB15K). Our experimental results suggest that Shallom outperforms state-of-the-art models on many benchmark datasets, e.g., on FB15K-237 and WN18RR with margins of up to 3% and 8% (absolute), respectively, while requiring a maximum training time of 8 minutes on a commodity computer. An open-source implementation of Shallom, including training and evaluation scripts are available at https://github.com/dice-group/Shallom and https://github.com/dice-group/dice-embeddings. Our second scientific contribution serves as Chapter 4.

Trade-offs: Although Shallom addresses the inability of Pyke in predicting missing triples and maintains a strong runtime performance, Shallom cannot *directly* predict missing triples by means of predicting missing entities. This stems from the fact that Shallom learns a mapping from entities to relations.

1.3.3 Convolutional Complex Knowledge Graph Embeddings

In Chapter 5, we propose ConEx. Therein, we alleviate the limitations of Pyke and Shallom. Inspired by two impactful works (ComplEx (Trouillon et al., 2016) and ConvE (Dettmers et al., 2018)), we design ConEx as a multiplicative composition of a 2D convolution operation with a Hermitian inner product on complex-valued embeddings. Therefore, ConEx can be seen as a multiplicative composition of ComplEx and ConvE. ConEx learns a complex-valued embeddings for entities and relations as ConEx utilizes the Hadamard product to compose a 2D convolution followed by an affine transformation with a Hermitian inner product in $\mathbb{C}$. This combination endows ConEx with the capability of *controlling the impact of the convolution on the Hermitian inner product of embeddings* and *degenerating into ComplEx if such a degeneration is necessary to further minimize the incurred training loss.* We evaluate ConEx on five benchmark datasets (WN18RR, FB15K-237, YAGO3-10, WN18, and FB15K). Our experimental results suggest that ConEx outperforms state-of-the-art models on four of the five datasets w.r.t. Hits@1 and MRR even without extensive hyperparameter optimization. Our results also indicate that the generalization performance of state-of-the-art models can be further increased through prediction averaging, i.e., a simple form of ensemble learning. An open-source implementation of ConEx, including training and evaluation scripts are available at https://github.com/dice-group/Convolutional-Complex-Knowledge-Graph-Embeddings and https://github.com/dice-group/dice-embeddings. Our third scientific contribution serves as Chapter 5.

Trade-offs: To reach a new-state-of-the-art performance, ConEx often requires 100-200 more training epochs than DistMult and ComplEx. This runtime inferior performance of ConEx may stem from the fact that ConEx simultaneous learns embeddings as well as filters in the 2D convolution operation via the element-wise multiplication.

1.3.4 Convolutional Hypercomplex Embeddings for Link Prediction

After our third scientific contribution introduced in Section 1.3.3 and elucidated in Chapter 5, we were eager to find out whether our finding in $\mathbb{C}$ generalizes to the two largest normed division algebras. Many recent works including (Dettmers et al., 2018; Nguyen et al., 2018a; Balažević et al., 2019b) indicate that using convolutions on $\mathbb{R}$ and $\mathbb{C}$ leads to strong performance in the link prediction problem. Yet, using convolutions on hypercomplex numbers (Quaternions $\mathbb{H}$ and Octonions $\mathbb{O}$) has not been studied in the KGE literature. In Chapter 6, we propose the four KGE models QMult, OMult, ConvQ and ConvO. QMult and OMult can be considered as quaternion and octonion extensions of the previous state-of-the-art approaches DistMult and ComplEx. ConvQ and ConvO build upon QMult and OMult by including convolution operations in a way inspired by the residual learning framework. We evaluate our four models on seven benchmark datasets (WN18RR, FB15K-237, YAGO3-10, WN18, FB15K, Kinship, and UMLS). Our experimental results suggest that the benefits of learning hypercomplex-valued vector representations become more tangible as the size and link prediction difficulty of the KG grows. ConvO outperforms state-of-the-art approaches on FB15K-237 in MRR, Hit@1 and Hit@3, while QMult, OMult, ConvQ and ConvO outperform state-of-the-art approaches on YAGO3-10 in all metrics. Moreover, to boost the generalization performance, we apply ensemble learning by means of prediction averaging. Applying ensemble learning consistently improves the link prediction performance across models and datasets. We also design a prediction constraint rule based on the semantics of the input KG that detects and disregards implausible entities based on the range of a given relation at testing time. This technique further improves the link prediction performance across models and benchmark datasets. An open-source implementation of our models, including training and evaluation scripts are available at https://github.com/dice-group/Convolutional-Hypercomplex-Embeddings-for-Link-Prediction and https://github.com/dice-group/dice-embeddings. Our forth scientific contribution serves as Chapter 6.

Trade-offs: During our experiments, we observe that QMult, OMult, ConvQ, and ConvO often require more epochs to reach their peak performance. Increasing the link prediction performance via more complex operations (e.g., hypercomplex multiplication or 2D convolution operation) comes often with a cost of increased runtimes.

1.3.5 Kronecker Decomposition for Knowledge Graph Embeddings

In Chapter 7, we focus on designing a generic technique to increase the parameter efficiency of a KGE model. Therein, we propose KronE technique to reduce the number of explicitly stored parameters in a KGE model, while retaining its effectiveness. During training, *an embedding vector is not plainly retrieved but constructed on the fly* via the Kronecker product. Our experimental results suggest that KronE significantly reduces the number of parameters with no cost of predictive performance, while making a KGE model more robust against the noise. An open-source implementation of KronE is available at https://github.com/dice-group/dice-embeddings. Our fifth scientific contribution serves as Chapter 7.

Trade-offs: Although our experiments highlight the benefits of using KronE in terms of the parameter efficiency and robust predictions against noise in the input KG, using KronE consistently increases total runtimes, hence, the benefits of using KronE comes with a cost of increased runtimes.

1.3.6 Polyak Parameter Ensemble for Knowledge Graph Embeddings

In Chapter 8, we focus on designing a technique to increase the performance of a KGE model without increasing computational costs. Therein, we propose PPE technique to increase the generalization performance with virtually no additional cost through maintaining a running weighted average of parameters. This can be seen as leveraging ensemble learning by means of parameter averaging instead of prediction averaging. Prediction averaging technique comes with the computational overhead of training multiple models and increased latency and memory requirements at test time. By utilizing the noisy approximation of mini-batch gradients at each epoch interval, PPE constructs a high performing parameter ensemble model with an expense of training a single KGE model. Our experiments on 9 benchmark datasets suggest that PPE improves the performance across models and datasets. We also design a training strategy to facilitate large-scale KGE training. Our open-source implementation, including training and evaluation scripts, is available at https://github.com/dice-group/dice-embeddings. Our sixth scientific contribution serves Chapter 8.

Trade-offs: During our experiments, we did not observe any increase in runtimes, while applying PPE. Yet, as the size of a KGE model increases, this discrepancy may differ.

1.3.7 Learning Permutation-Invariant Embeddings for Description Logic Concepts

In Chapter 9, we focus on learning embeddings for expressive DLs. More specifically, we focus on learning embeddings for DL $\mathcal{ALC}$ concepts and individuals tailored towards the Concept Learning (CL). We formulate the CL problem as a multi-label classification problem from a set of DL individuals to DL concepts and propose NeRo—a permutation-invariant neural embedding model based on a deep set neural network. NeRo learns permutation-invariant embeddings tailored towards F_1 scores of pre-selected DL concepts w.r.t. input example DL individuals. By ranking such concepts in descending order of predicted scores, a possible goal concept can be detected within few retrieval operations, i.e., our model does not require excessive exploration. Importantly, top-ranked concepts can be used to start the search procedure of state-of-the-art symbolic models in multiple advantageous regions of a concept space, rather than starting it in the most general concept $\top$. Our experiments on five benchmark datasets with 770 learning problems firmly suggest that NeRo significantly (p-value $< 1\%$) outperforms the state-of-the-art models in terms of F_1 score, the number of explored concepts, and the total runtime. An open-source implementation of NeRo, including training and evaluation scripts are available at https://github.com/dice-group/Nero. Our seventh scientific contribution serves as Chapter 9.

Trade-offs: While NeRo effectively alleviates the well-known exploration problem in CL, hence, decreases the impractical runtimes, NeRo is not complete in CL as most CL models are. Consequently, NeRo may not find an existing goal concept, whereas symbolic models (e.g. CELOE) can find a goal an existing goal concept provided that there is no time and memory constraints.

2
Background

In this chapter, we provide necessary background knowledge to make this thesis self-contained. At the end of each subsection, we draw an explicit connection from an introduced concept to our research contributions. This chapter consists of seven sections. Section 2.1 covers a wide range of ML topics related to our research. Section 2.2 briefly introduces Reinforcement Learning (RL). Section 2.3 elucidates knowledge graph along with benchmark tasks and metrics. Section 2.4 introduces related works. Section 2.5 introduces description logic and the concept learning problem.

2.1 Machine Learning

2.1.1 Expected and Empirical Risk Minimization

In a supervised ML setting, our goal often is to minimize *the expected risk* defined as

$$Risk(h) = \int_{\mathcal{X}\times\mathcal{Y}} \ell(h(\mathbf{x}), \mathbf{y})\ dP_{data}(\mathbf{x}, \mathbf{y}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P_{data}}\Big[\ell(h(\mathbf{x}), \mathbf{y})\Big], \tag{2.1}$$

where $\mathcal{X}$ and $\mathcal{Y}$ denote the input space and the output space, respectively. $P_{data} : \mathcal{X} \times \mathcal{Y} \mapsto [0, 1]$ denotes an unknown joint probability distribution. An i.i.d. sampled observation/input-output pair is denoted by $(\mathbf{x}, \mathbf{y}) \sim P_{data}$. A predictor function is defined as $h : \mathcal{X} \mapsto \mathcal{Y}$ and $\ell : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}_{\geq 0}$ is a scalar function determining the

discrepancy between a prediction $\hat{\mathbf{y}} := h(\mathbf{x})$ and a true output $\mathbf{y}$. Hence, the problem of learning an *optimal* predictor function h can be defined as

$$h^* = \arg\min_h \ \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P_{data}}\Big[\ell(h(\mathbf{x}), \mathbf{y})\Big]. \tag{2.2}$$

Yet, finding h^* entails searching through all possible families of predictor functions (e.g., logistic regression, neural networks, and support vector machines). Instead, in most cases, we aim to find *an optimal predictor function belonging to a particular family of predictor functions* $\mathcal{H}$ defined as

$$h^*_{\mathcal{H}} = \arg\min_{h\in\mathcal{H}} \ \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P_{data}}\Big[\ell(h(\mathbf{x}), \mathbf{y})\Big]. \tag{2.3}$$

Since P_{data} is unknown in almost all machine learning problems, Equations (2.1) to (2.3) cannot be directly minimized (Bishop and Nasrabadi, 2006; Murphy, 2012; Goodfellow et al., 2016). If P_{data} were known, through the chain rule of probability $\big(P_{data}(\mathbf{x}, \mathbf{y})) = P_{data}(\mathbf{y} \mid \mathbf{x})\ P_{data}(\mathbf{x})\big)$, the conditional probability distribution could be obtained to describe to the relationship between input-output pairs, i.e., searching for $h^*_{\mathcal{H}}$ or h^* becomes unnecessary. Given that P_{data} is unknown and Equations (2.1) to (2.3) can not be directly minimized, we often attempt to minimize Equation (2.1) via minimizing a surrogate function (*the empirical risk*) that is defined as

$$Risk_{\mathcal{D}}(h) = \frac{1}{n}\sum_{i=1}^{n} \ell(h(\mathbf{x}_i), \mathbf{y}_i) = \mathbb{E}_{\mathcal{D}}\Big[\ell(h(\mathbf{x}), \mathbf{y})\Big], \tag{2.4}$$

where $\mathcal{D} := \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ denotes a training dataset that is $\mathcal{D} \sim_{i.i.d.} P^n_{data}$ and $n \in \mathbb{N}^+$. $Risk_{\mathcal{D}}(h)$ quantifies the empirical risk of h w.r.t. $\mathcal{D}$. Therefore, an *optimal empirical predictor function from a particular family of predictor functions* $\mathcal{H}$ can be defined as

$$h^*_{\mathcal{D},\mathcal{H}} = \arg\min_{h\in\mathcal{H}} \ \mathbb{E}_{\mathcal{D}}\Big[\ell(h(\mathbf{x}), \mathbf{y})\Big]. \tag{2.5}$$

The weak law of large numbers suggests that as the size of the dataset grows $|\mathcal{D}| \to +\infty$, the following holds $\mathbb{E}_{\mathcal{D}}[\ell(h(\mathbf{x}), \mathbf{y})] \approx \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim P_{data}}[\ell(h(\mathbf{x}), \mathbf{y})]$ (Murphy, 2012). This serves as a guarantee that $P_{data}(\mathcal{Y} \mid \mathcal{X})$ can be effectively approximated provided that

$\mathcal{D}$ and $\mathcal{H}$ are sufficiently large. The expected excess risk between h^* and $h^*_{\mathcal{D},\mathcal{H}}$ can be defined as

$$\text{EER} = \mathbb{E}\Big[Risk(h^*_{\mathcal{H}}) - Risk(h^*)\Big] + \mathbb{E}\Big[Risk(h^*_{\mathcal{D},\mathcal{H}}) - Risk(h^*_{\mathcal{H}})\Big]. \tag{2.6}$$

Equation (2.6) comprise the approximation and estimation error parts. The former signals the closeness between h^* and $h^*_{\mathcal{H}}$, while the latter signals the closeness between $h^*_{\mathcal{H}}$ and $h_{\mathcal{D},\mathcal{H}}$. As the size of $\mathcal{H}$ grows, the approximation error can be minimized. For instance, selecting sufficiently large $\mathcal{H}$ (e.g., neural networks) can lead to the minimization of the approximation error. Minimizing the estimation error is more challenging, since this error is determined by $\mathcal{D}$ and $\mathcal{H}$. More specifically, while selecting sufficiently complex $\mathcal{H}$ is desired to minimize the approximation error, this decision often leads to *the problem of overfitting* without having large $\mathcal{D}$, e.g., very low approximation error and high estimation error. Conversely, selecting simple $\mathcal{H}$ facilities the minimization of the estimation error, yet it often leads to *the problem of underfitting*, e.g., high approximation error and low estimation error. Finding a good ratio between the complexity of $\mathcal{H}$ w.r.t. $|\mathcal{D}|$ becomes a keystone in many successful ML applications (Bishop and Nasrabadi, 2006; Murphy, 2012).

Bottou and Bousquet (2007) show that as $|\mathcal{D}|$ grows, learning $h^*_{\mathcal{D},\mathcal{H}}$ becomes computationally challenging, sometimes even infeasible due to the required time and/or computational budgets. Yet, these aspects are not involved in Equation (2.6). To address this limitation, Bottou and Bousquet (2007) extends Equation (2.6) with *the optimization error* part as follows

$$\mathbb{E}\Big[Risk(h^*_{\mathcal{H}}) - Risk(h^*)\Big] + \mathbb{E}\Big[Risk(h^*_{\mathcal{D},\mathcal{H}}) - Risk(h^*_{\mathcal{H}})\Big] + \mathbb{E}\Big[Risk(\overline{h}_{\mathcal{D},\mathcal{H}}) - Risk(h^*_{\mathcal{D},\mathcal{H}})\Big], \tag{2.7}$$

where $\overline{h}_{\mathcal{D},\mathcal{H}} \in \mathcal{H}$ denotes a predictor, whose *iterative learning process* is terminated before reaching a minima of the empirical risk (see Equation (2.4)) w.r.t. $\mathcal{D}$. For instance, the early stopping technique can be considered as a technique to find a trade-off between the approximation, estimation, and the optimization error parts (LeCun et al., 2002; Prechelt, 2012).

Connection to our scientific contributions: The effectiveness of ML models is measured by their ability to generalize well on unseen data as shown in Equation (2.6).

For instance, the effectiveness of a KGE model is often measured by means of its ability to accurately predict missing links on a test dataset (see Chapter 2). Throughout our scientific contributions, we aim to incorporate the approximation, estimation and optimization error aspects in the design of our proposed models to generalize well on unseen data as well as to our experimental setups. By designing our models as neural networks, we ensure that **the approximation error** can be effectively minimized, since neural networks are universal approximators. By using ensemble learning, the dropout technique, the label smoothing, the batch normalization, the explicit complexity calibration of $\mathcal{H}$, designing a technique based on the Polyak averaging, we ensure that **the estimator error** can be minimized. Moreover, our experimental setups are designed in such a manner that **the optimization error** can be efficiently reduced subject to our computational and time budgets. The non-trivial endeavor of minimizing Equation (2.7) plays a central role behind of all our scientific contributions.

2.1.2 Linear Regression and Logistic Regression

Linear regression and logistic regression often serve as baseline models in regression and classification problems. Although they can be seen as simple models, they can generalize better than complex models, especially in cases where the training data is relatively small as shown in Equation (2.7) (Hastie et al., 2009).

Linear Regression

Setup: Let $\mathcal{D} \coloneqq \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ denote n i.i.d. sampled examples $\mathcal{D} \sim_{i.i.d.} P_{data}^{n}$, where $\mathbf{x} \in \mathcal{X} = \mathbb{R}^{d_x}, y \in \mathcal{Y} = \mathbb{R}^{d_y}$ having $d_y = 1$ and P_{data} denotes an unknown data distribution over $\mathcal{X}, \mathcal{Y}$. A linear regression predictor function can be seen as a parameterized linear mapping from inputs to outputs in the following form $h(\mathbf{x}; \mathbf{w}, b) = \mathbf{w}^\top \mathbf{x} + b$, where $\mathbf{w} \in \mathbb{R}^{d_x}$ and $b \in \mathbb{R}$. In practice, $\mathcal{X}$ is often augmented with an extra dimension of 1s to include b in $\mathbf{w}$ (Hastie et al., 2009). Hence, the underlying assumption in linear regression can be defined as

$$\forall (\mathbf{x}_i, y_i) \in \mathcal{D} \ , \ \exists \mathbf{w} \in \mathbb{R}^{d_x} \text{ such that } y_i = \mathbf{w}^\top \mathbf{x}_i + \epsilon_i, \tag{2.8}$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ denotes the zero centered Gaussian noise with constant variance. Equation (2.8) implies that (1) relationships between y_i and $\mathbf{x}_i$ are linear, (2) errors

$(y_i - \mathbf{w}^\top \mathbf{x}_i)$ are independent, (3) noises are zero centered with a constant variance σ^2. Expectedly, these assumptions often do not hold in practice (Hastie et al., 2009). A parameter vector $\hat{\mathbf{w}}$ leading to $\forall(\mathbf{x}, y) \in \mathcal{D}$, $\hat{\mathbf{w}}^\top \mathbf{x} \approx y$ can estimated/learned through minimizing incurred discrepancies between predictions and outputs, e.g., via the residual sum of squares/squared discrepancies or the average residual sum of squares

$$\mathcal{L}(\mathbf{w}) = \frac{1}{n} \sum_{i}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2. \tag{2.9}$$

Equation (2.9) is known as the mean squared error loss function Murphy (2012). Minimization of the residual sum of squares is known as the least squares estimation (see Section 7.3 Murphy (2012) for more details). Since Equation (2.9) is differentiable, quadratic and convex in $\mathbf{w}$ (hence, its minimum always exists, although it may not be a unique), a closed-form solution exists, e.g., the Moore-Penrose pseudo-inverse (Bishop and Nasrabadi, 2006; Hastie et al., 2009). Although a closed-form solution exists for Equation (2.9), in practice it is often more efficient to minimize it by iteratively updating $\mathbf{w}$ using the gradient of the loss function, $\mathcal{L}(\cdot)$, with respect to $\mathbf{w}$.

Maximum Likelihood Estimation (MLE): Linear regression can also be investigated in a probabilistic setting. More specifically, a $\hat{\mathbf{w}}$ leading to $\forall(\mathbf{x}, y) \in \mathcal{D}$, $\hat{\mathbf{w}}^T \mathbf{x} \approx y$ can be learned via MLE or Maximum a Posteriori Probability Estimation (MAP) methods. In MLE, the goal is to learn such $\hat{\mathbf{w}}$ that maximizes the likelihood of observing $\mathcal{D}$. In MAP, the goal is to maximize the likelihood of observing $\hat{\mathbf{w}}$ given $\mathcal{D}$.

Starting from Equation (2.10) to Equation (2.23), we derive the MLE solution for linear regression with a parametric probability distribution $P_{model}(\cdot; \mathbf{w})$.

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \; P_{model}(\mathcal{D}; \mathbf{w}) \qquad \text{Max. likel. of } \mathcal{D} \qquad (2.10)$$

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \prod_{i=1}^{n} P_{model}(\mathbf{x}_i, y_i; \mathbf{w}) \qquad \text{i.i.d. samples} \qquad (2.11)$$

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \prod_{i=1}^{n} P_{model}(y_i \mid \mathbf{x}_i; \mathbf{w}) \; P_{model}(\mathbf{x}_i; \mathbf{w}) \qquad \text{Chain Rule} \qquad (2.12)$$

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \prod_{i=1}^{n} P_{model}(y_i \mid \mathbf{x}_i; \mathbf{w}) \qquad \text{Remove Const.} \qquad (2.13)$$

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \sum_{i=1}^{n} \log\Big(P_{model}(y_i \mid \mathbf{x}_i; \mathbf{w})\Big) \qquad \text{Use Log} \qquad (2.14)$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} -\log\Big(P_{model}(y_i \mid \mathbf{x}_i; \mathbf{w})\Big) \qquad \text{Negative Log Likelihood} \qquad (2.15)$$

Maximizing the likelihood of observing $\mathcal{D}$ is equivalent to minimizing the negative log conditional likelihood of outputs given inputs. In Equation (2.15), the Gaussian distribution or the Laplace Gaussian can be used to model ($P_{model}(\cdot|\cdot; \mathbf{w})$) the linear relationship between inputs and noisy outputs. The conditional likelihood of outputs given inputs with the Gaussian distribution can be defined as follows

$$P_{model}(\mathcal{Y} = y \mid \mathcal{X} = \mathbf{x}; \mathbf{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{(\mathbf{w}^T\mathbf{x} - y)^2}{2\sigma^2}\Big), \qquad (2.16)$$

where $\sigma^2 \in \mathbb{R}_{>0}$ and π denote the variance and the pi constant. Similarly, the conditional likelihood of outputs given inputs with the Laplace distribution can be defined as follows

$$P_{model}(\mathcal{Y} = y \mid \mathcal{X} = \mathbf{x}; \mathbf{w}) = \frac{1}{2b} \exp\Big(-\frac{\mid y - \mathbf{w}^T\mathbf{x}_i \mid}{b}\Big), \qquad (2.17)$$

where $b \in \mathbb{R}_{>0}$ denotes the scale parameter. In Equations (2.18) and (2.23), we show the steps to derive MLE solution with the Gaussian distribution. In the last step, the incurred discrepancies are averaged to make the loss interpretable.

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} -\log\Big[\frac{1}{\sqrt{2\pi\sigma^2}}\exp\Big(-\frac{(\mathbf{w}^T\mathbf{x}_i - y_i)^2}{2\sigma^2}\Big)\Big] \quad \text{Insert Eq.2.16} \quad (2.18)$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} -\log\Big(\frac{1}{\sqrt{2\pi\sigma^2}}\Big) - \log\Big(\exp\big(-\frac{(\mathbf{w}^T\mathbf{x}_i - y_i)^2}{2\sigma^2}\big)\Big) \quad \text{Distribute Log} \quad (2.19)$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} -\log\Big(\exp\big(-\frac{(\mathbf{w}^T\mathbf{x}_i - y_i)^2}{2\sigma^2}\big)\Big) \quad \text{Remove Const.} \quad (2.20)$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{2\sigma^2}\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 \quad \text{Distribute Log} \quad (2.21)$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 \quad \text{Remove Const.} \quad (2.22)$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 \quad \text{Objective} \quad (2.23)$$

Important to note that Equation (2.23) and Equation (2.9) are *identical*. Hence, the least squares estimation is equivalent to the MLE with the Gaussian noise model, i.e., noises ϵ_i are normally distributed with equal variance. Moreover, it can be shown that using the Laplace distribution as the noise model (see Equation (2.17)) in MLE is equivalent of the least absolute deviation estimation, i.e., the minimization of the mean absolute error loss function defined as

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \frac{1}{n} \mid y_i - \mathbf{w}^T\mathbf{x}_i \mid \quad (2.24)$$

The Laplace distribution may be more suitable to increase robustness against outliers (see Section 13.3 in Murphy (2012)), since the Laplace distribution has heavier tails than the Gaussian distribution.

MAP: There is an extra assumption $P_{param}(\mathbf{w})$ that represents a prior belief about possible parameter vectors. Choosing a particular prior distribution incorporates a belief of a domain expert into the process of learning $\mathbf{w}$. For instance, through choosing

the Gaussian distribution with zero mean and an isotropic covariance ($\mathbf{w} \sim \mathbf{N}(\mathbf{0}, \tau^2\mathbf{I})$) defined in Equation (2.25), the absence of a prior belief can be incorporated in the given learning problem (Bishop and Nasrabadi, 2006; Murphy, 2012).

$$P_{param}(\mathbf{w}) = \frac{1}{\sqrt{2\pi\tau^2}} \exp\Big(- \frac{\mathbf{w}^T\mathbf{w}}{2\tau^2}\Big) \tag{2.25}$$

Note that using an isotropic covariance results in the same amount of variance in every direction. Starting from Equation (2.26) to Equation (2.34), we derive the MAP solution for linear regression. Using Equation (2.25) in MAP for linear regression results in learning smaller parameter vectors, since adding $\frac{1}{2\tau^2}\mathbf{w}^T\mathbf{w}$ encourages a $\mathbf{w}$ to decay towards zero, unless supported by $\mathcal{D}$ (Bishop and Nasrabadi, 2006). $\frac{1}{2\tau^2}$ can be used to control the size of possible parameters vectors, i.e., the complexity/size of possible prediction functions in $\mathcal{H}$.

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} P(\mathbf{w} \mid \mathcal{D}) \qquad \text{Max. likel. of } \mathbf{w} \tag{2.26}$$

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \frac{P_{model}(\mathcal{D} \mid \mathbf{w})P_{param}(\mathbf{w})}{P(\mathcal{D})} \qquad \text{Bayes Rule} \tag{2.27}$$

$$\mathbf{w} = \arg\max_{\mathbf{w}} P_{model}(\mathcal{D} \mid \mathbf{w})P_{param}(\mathbf{w}) \qquad \text{Remove Const.} \tag{2.28}$$

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \Big[\prod_{i=1}^{n} P_{model}(\mathbf{x}_i, y_i \mid \mathbf{w}) \Big] P_{param}(\mathbf{w}) \qquad \text{i.i.d.} \tag{2.29}$$

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \Big[\prod_{i=1}^{n} P_{model}(y_i \mid \mathbf{x}_i, \mathbf{w}) \Big] P_{param}(\mathbf{w}) \qquad \text{Remove Const.} \tag{2.30}$$

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \sum_{i=1}^{n} \log\Big(P_{model}(y_i \mid \mathbf{x}_i, \mathbf{w})\Big) + \log\Big(P_{param}(\mathbf{w})\Big) \qquad \text{Use Log} \tag{2.31}$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (\mathbf{w}^T\mathbf{x}_i - y_i)^2 + \log\Big(P_{param}(\mathbf{w})\Big) \qquad \text{Insert eq. (2.16)} \tag{2.32}$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (\mathbf{w}^T\mathbf{x}_i - y_i)^2 + \frac{1}{2\tau^2}\mathbf{w}^T\mathbf{w} \qquad \text{Insert 2.25} \tag{2.33}$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} (\mathbf{w}^T\mathbf{x}_i - y_i)^2 + \lambda\mathbf{w}^T\mathbf{w} \qquad \text{Objective} \tag{2.34}$$

An overview of commonly used likelihoods and priors is given in Table 2.1. Note that MAP with a uniform distribution corresponds to MLE and we refer Chapter 7 of Murphy (2012) for more details about MLE and MAP.

Table 2.1: Overview of linear regression model w/o probabilistic interpretation. A regularization constant denoted by λ.

$P_{model}(\cdot)$	$P_{param}(\cdot)$	Objective	Name	Regularization
Gaussian	-	$\frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}_i - y_i)^2$	Mean Squared Error	-
Gaussian	Laplace	$\frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 + \lambda\sum_j^d \lvert\mathbf{w}_j\rvert$	Mean Squared Error	l_1 norm
Gaussian	Gaussian	$\frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}_i - y_i)^2 + \lambda\sum_j^d \mathbf{w}_j^2$	Mean Squared Error	l_2 norm
Laplace	-	$\frac{1}{n}\sum_{i=1}^{n}\lvert\mathbf{w}^T\mathbf{x}_i - y_i\rvert$	Mean Absolute Error	-
Laplace	Laplace	$\sum_{i=1}^{n}\lvert\mathbf{w}^T\mathbf{x}_i - y_i\rvert + \lambda\sum_j^d \lvert\mathbf{w}_j\rvert$	Mean Absolute Error	l_1 norm
Laplace	Gaussian	$\sum_{i=1}^{n}\lvert\mathbf{w}^T\mathbf{x}_i - y_i\rvert + \lambda\sum_j^d \mathbf{w}_j^2$	Mean Absolute Error	l_2 norm

Logistic Regression

Setup: Let $\mathcal{D} := \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ denote n i.i.d. sampled examples $\mathcal{D} \sim_{i.i.d.} P_{data}^n$, where $\mathbf{x} \in \mathcal{X} = \mathbb{R}^{d_x}, y \in \mathcal{Y} = \{1, 0\}$ and P_{data} denotes an unknown data distribution over $\mathcal{X}, \mathcal{Y}$. Logistic regression can be seen as linear regression with two modifications: (1) the likelihood of an output given an input is modeled with a discrete probability distribution and (2) a prediction is confined within the unit interval (see Equation (2.36)). With Bernoulli distribution, (1) is computed as follows

$$P_{model}(y \mid \mathbf{x}; \mathbf{w}) = \hat{y}^y(1 - \hat{y})^{1-y}, \tag{2.35}$$

where

$$\hat{y} := \sigma(\mathbf{w}^\top\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^T\mathbf{x})}. \tag{2.36}$$

Therefore, $P_{model}(y = 1 \mid \mathbf{x}; \mathbf{w}) + P_{model}(y = 0 \mid \mathbf{x}; \mathbf{w}) = 1$. Note that the first four equations provided in the solution of MLE for linear regression hold in the derivation of the MLE solution for logistic regression.

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \prod_{i=1}^{n} P_{model}(y_i \mid \mathbf{x}_i; \mathbf{w}) \qquad \text{Maximize likelihood of i.i.d} \quad (2.37)$$

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \sum_{i=1}^{n} \log\Big(P_{model}(y_i \mid \mathbf{x}_i; \mathbf{w})\Big) \qquad \text{Use Log} \quad (2.38)$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} -\log\Big(P_{model}(y_i \mid \mathbf{x}_i; \mathbf{w})\Big) \qquad \text{Negative Log Likelihood} \quad (2.39)$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} -\log\Big(\hat{y}_i^{y}(1-\hat{y})^{1-y}\Big) \qquad \text{Bernoulli Dist.} \quad (2.40)$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} -y\log(\hat{y}_i) - (1-y)\log(1-\hat{y}) \qquad \text{Binary Cross-Entropy} \quad (2.41)$$

Maximizing the log-likelihood is equivalent to minimizing the binary cross-entropy. Although there is no closed-form solution for Equation (2.41), it can be effectively minimized through iteratively updating $\mathbf{w}$ in the negative direction of the gradient of Equation (2.41) w.r.t $\mathbf{w}$. This is possible, as Equation (2.41) is a convex and continuous function. Note that deriving the MAP estimation for logistic regression results in adding $(\mathbf{w}^T\mathbf{w})$ to Equation (2.41) in the MAP estimation for linear regression provided that the Gaussian prior is used.

Connection to our scientific contributions: Most knowledge graph embedding models are defined as scoring functions based on logistic regression (see Section 2.4). Through minimizing the binary cross-entropy function, parameters of knowledge graph embedding models are learned to discriminating positive triples ($\forall(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$) from negative triples ($\forall(\mathsf{x}, \mathsf{y}, \mathsf{z}) \notin \mathcal{G}$). Such parameters often contain embedding vectors for entities and relations as well as other trainable parameters (e.g., the batch normalization parameters (Ioffe and Szegedy, 2015)). In Chapter 12, we formulate the concept learning problem as a regression problem with the mean squared loss function, i.e, a problem of maximizing cumulative discounted reward.

2.1.3 Learning with Gradient Descent

Objective loss functions introduced in Section 2.1.2 can be written in a generic form

$$\mathcal{L}_{\mathcal{D}}(\mathbf{w}) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} \ell(h(\mathbf{x};\mathbf{w}), y), \tag{2.42}$$

where ℓ and h denote a loss function and a parameterized predictor function. $\mathcal{L}_{\mathcal{D}}(\mathbf{w})$ denotes the empirical risk of $h(\cdot;\mathbf{w})$ on $\mathcal{D}$, i.e., the average incurred loss of using h with $\mathbf{w}$ on $\mathcal{D}$. Provided that ℓ in Equation (2.42) is a continuous and *mostly* differentiable function, iteratively updating $\mathbf{w}$ in the direction of the negative gradient of the $\mathcal{L}_{\mathcal{D}}$ w.r.t. $\mathbf{w}$ may lead to finding a $\mathbf{w}$ minimizing Equation (2.42).[1] This formulation gives a rise to well known Gradient Descent (GD) algorithm (Robbins and Monro, 1951). There are three major different variations of GD: Full-batch GD, Stochastic Gradient Descent (SGD), and mini-batch SGD.

Full-Batch GD: At each the full-batch GD update, $\mathbf{w}$ is updated as follows

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x},y)\in\mathcal{D}} \nabla_{\mathbf{w}}\, \ell(h(\mathbf{x};\mathbf{w}_t), y) = \mathbf{w}_t - \eta\, \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{D}}(\mathbf{w}_t), \tag{2.43}$$

where $\eta \in \mathbb{R}_{>0}$ denotes the learning rate. The computational cost for a single GD update is $O(|\mathcal{D}|)$. Consequently, as $|\mathcal{D}|$ grows to billions, a single gradient step becomes prohibitively long (Goodfellow et al., 2016).

SGD: SGD can be seen as an extreme simplified version of the full-batch GD. At each SGD update, $\mathbf{w}$ is updated as follows

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla_{\mathbf{w}} \ell(h(\mathbf{x}_t);\mathbf{w}_t, y_t), \tag{2.44}$$

where $(\mathbf{x}_t, y_t) \in \mathcal{D}$ denotes a randomly sampled data point. Hence, SGD reduces the computational cost of the parameter update from $O(|\mathcal{D}|)$ to $O(1)$.

[1] Note that although the mean absolute error (see Table 2.1) is not differentiable at 0, it can be also minimized in practice.

Mini-batch SGD: The mini-batch SGD can be seen as a less simplified version of the full-batch GD compared to SGD. The parameter update is defined as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{x},y)\in\mathcal{B}} \nabla_{\mathbf{w}} \ell(h(\mathbf{x};\mathbf{w}_t), y) = \mathbf{w}_t - \eta \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}_t), \tag{2.45}$$

where $\mathcal{B} \subset \mathcal{D}$ denote a set of data points that are sampled uniformly at random without replacement. Hence, the mini-batch SGD reduces the computational cost for a parameter update from $O(|\mathcal{D}|)$ to $O(|\mathcal{B}|)$, where $|\mathcal{D}| >> |\mathcal{B}|$. The size of the batch size is often selected w.r.t. the available hardware (see Goodfellow et al. (2016) for more details). For the sake of brevity, we will omit the term of the mini-batch. We refer Section 1.5. of LeCun et al. (2002) for details about the convergence properties of SGD and GD.

Momentum Update: In Equations (2.43) and (2.45), gradients are elementwise averaged. This often leads to oscillating parameter updates provided that η is not sufficiently small (LeCun et al., 2002). Oscillating parameter updates can slow down convergence and sometimes even lead to divergence during training (Goodfellow et al., 2016). Although using sufficiently small η alleviates oscillating parameter updates, it results in slower convergence. The momentum update dampens this oscillating behavior by updating parameters with exponential moving average of previous gradients (LeCun et al., 2002; Sutskever et al., 2013). More specifically, SGD with the momentum update is defined as

$$\mathbf{v}_{t+1} = \mu\, \mathbf{v}_t + (1-\mu)\nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}_t) \qquad \text{Momentum Update} \tag{2.46}$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mathbf{v}_{t+1}, \qquad \text{Parameter Update} \tag{2.47}$$

where $\mu \in [0, 1)$ denotes the momentum decay controlling the impact of old gradients and $\mathbf{v}_t = \mathbf{0}$. If $\mu = 0$, then the impact of old gradients in the parameter update is discarded, i.e., previous gradients are scaled to 0. Conversely, setting $\mu \approx 0.99$ implies that $\mathbf{w}$ is not updated in the direction of the steepest descent. If the loss surface is highly non-spherical, using Momentum often accelerates the convergence as it smooths/dampens the size of the parameter updates along directions of high curvature. This results in larger parameter updates along the directions of low curvature (LeCun et al., 2002).

RMSprop and Adam: Tieleman and Hinton (2012) propose RMSprop that tackles the oscillating parameter update behavior by finding an adaptive learning rate for each parameter. More specifically, the parameter update in RMSprop is defined as

$$\mathbf{u}_{t+1} = \mu\mathbf{u}_t + (1-\mu)\big(\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}_t) \circ \nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}_t)\big) \quad \text{Moving Avg. of Sq. Grads.} \quad (2.48)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\frac{\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}_t)}{\sqrt{\mathbf{u}_{t+1}} + \epsilon}, \quad \text{Parameter Update} \quad (2.49)$$

where all operations in Equation (2.49) are performed in an elementwise fashion and $\epsilon = 10^{-8}$ (Tieleman and Hinton, 2012). Dividing the current gradient element-wise by running average of squared gradients results in individual learning rate per parameter. This permits using larger learning rates η. Kingma and Ba (2014) propose Adam that combines Momentum with RMSprop with bias correction terms as follows

$$\mathbf{v}_{t+1} = \beta_1\mathbf{v}_t + (1-\beta_1)\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}_t) \quad \text{Momentum} \quad (2.50)$$

$$\mathbf{u}_{t+1} = \beta_2\mathbf{u}_t + (1-\beta_2)\big(\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}_t) \circ \nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}}(\mathbf{w}_t)\big) \quad \text{RMSprop} \quad (2.51)$$

$$\hat{\mathbf{v}}_{t+1} = \frac{\mathbf{v}_{t+1}}{(1-\beta_1^t)} \quad \text{Momentum Correction} \quad (2.52)$$

$$\hat{\mathbf{u}}_{t+1} = \frac{\mathbf{u}_{t+1}}{(1-\beta_2^t)} \quad \text{RMSprop Correction} \quad (2.53)$$

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\frac{\hat{\mathbf{v}}_{t+1}}{\sqrt{\hat{\mathbf{u}}_{t+1}} + \epsilon}, \quad \text{Parameter Update} \quad (2.54)$$

where $\beta_1, \beta_2 \in [0, 1)$. Kingma and Ba (2014) show that initializing $\mathbf{u}$ and $\mathbf{v}$ with zero vectors biases weight updates towards zeros, hence it slows the learning process during the early phase of the training process.

Connection to our scientific contributions: In most of our scientific contributions, we apply the Adam optimizer in a stochastic mini-batch fashion (Kingma and Ba, 2014). Yet, as the size of the input knowledge graph grows, the extra memory overhead caused by storing $\mathbf{v}$ and $\mathbf{u}$ becomes more tangible w.r.t. our computational budget. Consequently, during our experiments, we carefully selected gradient-based optimizers w.r.t. the size of benchmark datasets as well as memory constraints. Moreover, we incorporate our domain knowledge in our open-source framework (see Chapter 10) to facilitate large-scale applications.

2.2 Reinforcement Learning

In recent years, RL has been successfully applied in learning policies for sequential decision-making problems. Notable examples range from deep Q-learning for Atari game-playing (Mnih et al., 2015) to protein folding (Jumper et al., 2021). Moreover, various RL models have been applied in diverse tasks on KGs, including question answering, link prediction, fact checking and knowledge graph completion (Zheng et al., 2018; Xian et al., 2019; Das et al., 2018; Lin et al., 2018; Xiong et al., 2017). Sequential decision problems in RL are often modeled as Markov Decision Processs (MDPs) applied to model the synchronous interaction between an *agent* and an *environment* in RL. Formally, an MDP is defined by a 5-tuple $\langle \mathbf{S}, \mathbf{A}, \mathbf{R}, \mathbf{T}, \gamma \rangle$, which comprises of a set of states $\mathbf{S}$, a set of actions $\mathbf{A}$, a reward function $\mathbf{R}$, a transition function $\mathbf{T}$ and the discount rate $\gamma \in [0, 1]$. Given a state $\mathbf{s}_t \in \mathbf{S}$ at a time t, an agent takes an action $\mathbf{a}_t$ from the set $\mathbf{A}(\mathbf{s}_t)$ of actions available on $\mathbf{s}_t$. Upon taking an action, the agent receives a reward r_t and reaches the next state $\mathbf{s}_{t+1}$. The probability of reaching $\mathbf{s}_{t+1}$ and receiving r_t by taking action $\mathbf{a}_t$ in a given $\mathbf{s}_t$ is assigned by $\mathbf{T}$. This synchronous interaction between an agent and an environment induces a *trajectory* τ. The *discounted return* of the t^{th} point in a trajectory is defined as

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{|\tau|-t} r_{|\tau|-t}, \tag{2.55}$$

where the discount rate $\gamma \in (0, 1]$ determines the present value of future rewards. Setting $\gamma = 0$ implies that the present value of future rewards is ignored, whereas setting $\gamma = 1$ implies that the present value of future rewards is equally important as the current reward. The goal of an RL agent is to select actions in a fashion that maximizes the cumulative discounted rewards (Sutton and Barto, 2018). A policy π prescribes which action to take in a given state. An optimal policy π_* hence prescribes actions on any state that maximize G_t. To obtain π_*, value functions are often used. The action-value function $Q_\pi : \mathbf{S} \times \mathbf{A} \mapsto \mathbb{R}$ is defined as

$$Q_\pi(\mathbf{s}, \mathbf{a}) = \mathbb{E}_\pi \left[G_t \mid \mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a} \right]. \tag{2.56}$$

Learning to act optimally: Using the Bellman equation, the optimal action-value function Q_* can be approximated:

$$Q_{i+1}(\mathbf{s},\mathbf{a}) \leftarrow Q_i(\mathbf{s},\mathbf{a}) + \eta\Big[r + \gamma \max_{\mathbf{a}'\in\mathbf{A}(\mathbf{s}')} Q_i(\mathbf{s}',\mathbf{a}') - Q_i(\mathbf{s},\mathbf{a})\Big], \tag{2.57}$$

where $\mathbf{s},\mathbf{a} \in \mathbf{A}(\mathbf{s}), \mathbf{s}', r$, and η denote the current state, the action taken on $\mathbf{s}$, the next stated reached by taking $\mathbf{a}$ on $\mathbf{s}$, the reward of reaching $\mathbf{s}'$, and the learning rate, respectively. Through using the iterative update defined in Equation (2.57), Q_i converges to Q_* as $i \rightarrow \infty$ (Sutton and Barto, 2018). However, iteratively approximating exact optimal values is often computationally infeasible as $|\mathbf{S}|$ or $|\mathbf{A}|$ increase. In practice, a neural network parameterized with Θ is commonly applied to approximate the optimal action-value function, $Q(\mathbf{s},\mathbf{a};\Theta) \approx Q_*(\mathbf{s},\mathbf{a})$ (Riedmiller, 2005). To this end, trajectories are often accumulated as a RL agent interacts with an environment. A training dataset $\mathcal{D}$ is iteratively built through appending trajectories. Θ is learned via minimizing the following loss function $(r + \gamma \max_{\mathbf{a}'\in\mathbf{A}(\mathbf{s}')} Q(\mathbf{s}',\mathbf{a}';\Theta) - Q(\mathbf{s},\mathbf{a};\Theta))^2$ (Riedmiller, 2005). Through introducing the experience replay mechanism and using the target network idea, Mnih et al. (2015) extend the previous work and design the following Q-loss function

$$\mathcal{L}(\Theta_i) = \mathbb{E}_{(\mathbf{s},\mathbf{a},r,\mathbf{s}')\sim U(\mathcal{D})}\left[\left(r + \gamma \max_{\mathbf{a}'\in\mathbf{A}(\mathbf{s}')} Q(\mathbf{s}',\mathbf{a}';\Theta_i^-) - Q(\mathbf{s},\mathbf{a};\Theta_i)\right)^2\right], \tag{2.58}$$

where $(\mathbf{s},\mathbf{a},r,\mathbf{s}') \sim U(\mathcal{D})$ denotes drawing a data point uniformly at random from a set of most recent data points. Hence, the size of $\mathcal{D}$ is fixed and it contains only most recent interactions between an RL agent and an RL environment. Moreover, Θ_i is the parameters of the neural network at iteration i and Θ_i^- is the parameters of the same neural network that is only updated at every few steps. This framework (known as deep Q-learning) allows approximating Q_* even in large state-action spaces (Sutton and Barto, 2018).

Connection to our research contributions: In Chapter 12, we use a pre-trained KGE model to represent a quasi-order DL concept space as a RL environment and use a deep-Q-network as a non myopic heuristic function. This approach plays a foundational role in an industrial use case (see Chapter 13).

2.3 Knowledge Graph and Link Prediction

2.3.1 Knowledge Graph

A Knowledge Graph (KG) represents structured collections of assertions describing the world (Hogan et al., 2020). These collections of assertions have been used in a wide range of applications, including web search, personal assistants, recommendation, question answering, fact checking, summarization, machine translation, and cancer research (Eder, 2012; Saleem et al., 2014; Zhang et al., 2016; Malyshev et al., 2018; Moussallem et al., 2019; Balog and Kenter, 2019; Hogan et al., 2020; Silva et al., 2021). Formally, a KG is often defined as a set of triples $\mathcal{G} := \{(\mathsf{h}, \mathsf{r}, \mathsf{t})\} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, where $\mathcal{E}$ and $\mathcal{R}$ represent a set of entities and a set of relations, respectively (Dettmers et al., 2018; Balažević et al., 2019c,b). Each triple $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$ represents a true assertion and is based on two entities $\mathsf{h}, \mathsf{t} \in \mathcal{E}$ and a relation $\mathsf{r} \in \mathcal{R}$. h and t are also known as head and tail entities, respectively. A relation r is *symmetric* iff. $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \iff (\mathsf{t}, \mathsf{r}, \mathsf{h})$ holds for $\mathsf{h}, \mathsf{t} \in \mathcal{E}$. Analogously, r is *anti-symmetric* iff $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G} \Rightarrow (\mathsf{t}, \mathsf{r}, \mathsf{h}) \notin \mathcal{G}$ for all $\mathsf{h} \neq \mathsf{t}$. A relation r is *transitive* iff $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G} \wedge (\mathsf{t}, \mathsf{r}, \mathsf{y}) \in \mathcal{G} \Rightarrow (\mathsf{h}, \mathsf{r}, \mathsf{y}) \in \mathcal{G}$ for all $\mathsf{h}, \mathsf{t}, \mathsf{y} \in \mathcal{E}$ (Sun et al., 2019; Kazemi and Poole, 2018). The inverse of a relation r, denoted r^{-1}, is a relation such that for any two entities h and t, $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G} \iff (\mathsf{t}, \mathsf{r}^{-1}, \mathsf{h}) \in \mathcal{G}$. Moreover, the domain and range of a relation are defined as $domain(r) = \{h \mid \forall\ (\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}\}$, $range(r) = \{t \mid \forall\ (\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}\}$, respectively.

2.3.2 Link Prediction

Link prediction on KGs refers to the problem of predicting missing triples (Dettmers et al., 2018). This task is also known as a single-hop reasoning task over KGs (Ren et al., 2022). In recent years, the link prediction problem has become a benchmark task to evaluate performance of KGE models. A missing triple can be predicted in one of the two following means: (1) by predicting a missing relation given two entities or (2) by predicting a missing entity given an entity and a relation. (1) and (2) correspond to *relation prediction* and *entity prediction*, respectively. (2) can be divided into into two subcategories: predicting a missing head entity given a relation and a tail entity, and predicting a missing tail entity given a head entity and a relation. (2) is known as the link prediction problem in the literature (Dettmers et al., 2018; Ruffinelli et al., 2019).

The link prediction problem is often tackled by learning a parameterized scoring function $\phi_\Theta : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \mapsto \mathbb{R}$ such that $\phi_\Theta(\mathsf{h}, \mathsf{r}, \mathsf{t})$ ideally signals a likelihood of (h, r, t) being true (Dettmers et al., 2018). The performance of a KGE model defined as a parameterized scoring function is quantified by its ability of generalizing well on unseen triples, i.e., assigning high likelihoods for $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}^{Test}$.

Link and Relation Prediction Evaluation: Mean Reciprocal Rank (MRR) and Hits@N are benchmark metrics to evaluate the performance of a KGE model in the link prediction problem. The former is defined as

$$\text{MRR} = \frac{1}{2|\mathcal{G}^{\text{Test}}|} \sum_{(\mathsf{h},\mathsf{r},\mathsf{t}) \in \mathcal{G}^{\text{Test}}} \frac{1}{\mathsf{rank}(S_{\text{h}}, h)} + \frac{1}{\mathsf{rank}(S_{\text{t}}, t)}, \tag{2.59}$$

where $\mathsf{rank}(\cdot, \cdot)$ returns the *filtered* rank of an entity and S denotes tuples of entities with corresponding scores sorted in descending order of scores, e.g., $S_{\text{h}} = [(x, score) | x \in \mathcal{E} \wedge score := \phi(x, r, t) \wedge (\mathsf{x}, \mathsf{r}, \mathsf{t}) \notin \mathcal{G}^{\text{Train}}]$, where the score in $S_{\text{h}}[i] \geq$ the score in $S_{\text{h}}[j]$ provided that $i < j$. Similarly, Hits@N is defined as

$$\text{Hits@N} = \frac{1}{2|\mathcal{G}^{\text{Test}}|} \sum_{(\mathsf{h},\mathsf{r},\mathsf{t}) \in \mathcal{G}^{\text{Test}}} \mathbb{I}(\mathsf{rank}(S_{\text{h}}, h) \leq N) + \big(\mathbb{I}(\mathsf{rank}(S_{\text{t}}, t) \leq N), \tag{2.60}$$

where $\mathbb{I}(\texttt{condition})$ returns 1 if `condition` is true, otherwise 0. MRR for the relation prediction task is defined as

$$\text{MRR} = \frac{1}{|\mathcal{G}^{\text{Test}}|} \sum_{(\mathsf{h},\mathsf{r},\mathsf{t}) \in \mathcal{G}^{\text{Test}}} \frac{1}{\mathsf{rank}(S_{\text{r}}, r)}, \tag{2.61}$$

where $\mathsf{rank}(\cdot, \cdot)$ returns the *filtered* rank of a relation and S denotes tuples of relations and scores sorted in descending order of scores. Similarly, Hits@N in relation prediction is computed as

$$\text{Hits@N} = \frac{1}{|\mathcal{G}^{\text{Test}}|} \sum_{(\mathsf{h},\mathsf{r},\mathsf{t}) \in \mathcal{G}^{\text{Test}}} \mathbb{I}(\mathsf{rank}(S_{\text{r}}, r) \leq N), \tag{2.62}$$

where $\mathbb{I}(\texttt{condition})$ returns 1 if `condition` is true, otherwise 0.

2.4 Related Works for Knowledge Graph Embeddings

Most Knowledge Graph Embedding (KGE) models are designed to learn continuous vector representations of entities and/or relations tailored towards link prediction. They are often formalized as parameterized scoring functions $\phi_\Theta : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \mapsto \mathbb{R}$ (Wang et al., 2017; Trouillon et al., 2016; Dettmers et al., 2018; Sun et al., 2019), where $\mathcal{E}, \mathcal{R}$ and Θ denote a set of entities, a set of relations and training parameters, e.g., an entity embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d}$, a relation embedding matrix $\mathbf{R} \in \mathbb{R}^{|\mathcal{R}| \times d}$, where d represents the number of dimensions. Note that there are also few KGE models that do not learn such relation embeddings explicitly, e.g., RESCAL represents each relation as a matrix or Shallom does not learn relation embeddings. An embedding vector for an entity $\mathtt{e} \in \mathcal{E}$ is denoted by $\mathbf{e}_e$. Similarly, an embedding vector for a relation $r \in \mathcal{R}$ is denoted by $\mathbf{e}_r$. A general training procedure for KGE models can be summarized as follows: During training, given a training data point consisting of a triple (h, r, t) and a binary label y, embedding vectors of h, r, and t are retrieved $\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{R}^d$. Thereafter, these three vectors are transformed into a scalar value, e.g., $z = \mathbf{e}_h \circ \mathbf{e}_r \cdot \mathbf{e}_t$. Next, the resulting scalar value z is mapped into the unit interval using the sigmoid function function, i.e., $\hat{y} = \frac{1}{1+\exp(-z)}$. This normalized scalar value $\hat{y}$ reflects the likelihood of (h, r, t) being true, i.e., the likelihood of being an element of $\mathcal{G}$. Finally, an incurred loss between y and $\hat{y}$ is computed. The resulting models are then evaluated w.r.t. their link ability of predicting missing entity rankings (Nickel et al., 2015; Dettmers et al., 2018; Ruffinelli et al., 2019). In Sections 2.4.1 and 2.4.2, we elucidates selected state-of-the-art KGE models and three commonly used training strategies.

2.4.1 Baseline Models

Here, we give a chronological overview of baseline KGE models. Table 2.2 provides a detailed overview of state-of-the-art KGE models. Nickel et al. (2011) propose a three-way factorization of a third-order binary tensor representing an input KG. The proposed approach (RESCAL) is limited in its scalability as it has a quadratic complexity in the factorization rank (Nickel et al., 2016). More specifically, as $|\mathcal{R}|$ increases, $|\Theta|$ of RESCAL grows quadratically in the entity embedding dimension. Balažević et al. (2019c) propose to apply the Tucker decomposition on third-order binary tensor representing an input KG.

Yang et al. (2015) propose DistMult that can be seen as an extension of RESCAL with a diagonal matrix per relation, where the dense core tensor is replaced with diagonal matrices. DistMult does not perform well on triples with antisymmetric relations (e.g., (`Barack`, `HasChild`, `Malia`)), whereas it performs well on symmetric relations (e.g., (`Barack`, `Married`, `Michelle`)). To avoid this shortcoming, Trouillon et al. (2016) extend DistMult by learning representations in a complex vector space. Their approach (ComplEx) can infer both symmetric and antisymmetric relations via complex vector multiplication followed by a Hermitian inner product of embeddings which involves the conjugate-transpose of one of the two input vectors. ComplEx yields state-of-the-art performance on the link prediction problem, while leveraging linear space and time complexity of the dot products (Ruffinelli et al., 2019). Sun et al. (2019) design RotatE that employs a rotational model by taking relations as rotations between entities in a complex space via the Hadamard product. RotatE performs well on transitive relations, while ComplEx performs poorly (Sun et al., 2019). Zhang et al. (2019) extend ComplEx into quaternions through applying the quaternion multiplication followed by an inner product to compute scores of triples. Cao et al. (2021) propose to learn dual quaternion valued embeddings by this they aim to universally models relations as the compositions of translation and rotation operations.

All aforementioned models learn embeddings of entities and/or relations via element-wise vector operations (e.g., the Hadamard product or inner product product). Although such models perform well in terms of predictive accuracy and computational complexity, to increase their expressiveness the embedding vector size is the only option. Dettmers et al. (2018); Nguyen et al. (2018a); Balažević et al. (2019b) show that convolution operation can be applied to increase the expressiveness of KGE models without significantly increasing the number of parameters. Dettmers et al. (2018) design ConvE that applies a 2D convolution to model the interactions between entities and relations. Through interactions captured by 2D convolution, ConvE yields a state-of-art performance in link prediction. Nguyen et al. (2018a) propose ConvKB that extends ConvE by omitting the reshaping operation. Similarly, HypER extends ConvE by applying relation-specific convolution filters as opposed to applying filters from concatenated subject and relation vectors (Balažević et al., 2019b).

Table 2.2: State-of-the-art KGE models with training strategies. $\mathbf{e}$ denotes embeddings, $\overline{\mathbf{e}} \in \mathbb{C}$ corresponds to the complex conjugate of $\mathbf{e}_\cdot$. $*, f$, concat(),vec() and $\triangleleft$ denote a convolution operation with ω kernel, rectified linear unit function, concatenation operation, flattening operation, and unit normalization, respectively. $\otimes, \bigstar, \circ, \cdot$ denote Hamilton/Quaternion, Octonion, Hadamard and inner products, respectively. In ConvE, the reshaping operation is omitted. The tensor product along the n-th mode is denoted by $\times_n$ and the core tensor is represented by $\mathcal{W}$. conv$(\cdot,\cdot)$ denotes 2D convolution operation followed by affine transformations. Bold capital letters denotes a weight matrix to perform linear transformation. MSE, BCE and MB denote mean squared error, binary cross entropy, margin based loss functions. NegSamp stands for negative sampling.

Model	Scoring Function	VectorSpace	Loss	Training
RESCAL (Nickel et al., 2011)	$\mathbf{e}_h \cdot \mathcal{W}_r \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_t \in \mathbb{R}$	MSE	Full
DistMult (Yang et al., 2015)	$\langle \mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \rangle$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{R}$	MB	NegSamp
ComplEx (Trouillon et al., 2016)	$\mathrm{Re}(\langle \mathbf{e}_h, \mathbf{e}_r, \overline{\mathbf{e}_t} \rangle)$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{C}$	BCE	NegSamp
ConvE (Dettmers et al., 2018)	$f(\mathrm{vec}(f([\mathbf{e}_h; \mathbf{e}_r] * \omega))\mathbf{W}) \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{R}$	BCE	KvsAll
HyPER (Balažević et al., 2019b)	$f(\mathrm{vec}(\mathbf{e}_h * \mathrm{vec}(\mathbf{e}_r\mathbf{H}))\mathbf{W})) \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{R}$	BCE	KvsAll
TuckER (Balažević et al., 2019c)	$\mathcal{W} \times_1 \mathbf{e}_h \times_2 \mathbf{e}_r \times_3 \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{R}$	BCE	KvsAll
RotatE (Sun et al., 2019)	$- \parallel \mathbf{e}_h \circ \mathbf{e}_r - \mathbf{e}_t \parallel$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{C}$	MB	NegSamp
QuatE (Zhang et al., 2019)	$\mathbf{e}_h \otimes \mathbf{e}_r^{\triangleleft} \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{H}$	BCE	NegSamp
Shallom (ours)	$\mathbf{W} \cdot f\big(\mathbf{H} \cdot \mathrm{concat}(\mathbf{e}_h, \mathbf{e}_t)\big)$	$\mathbf{e}_h, \mathbf{e}_t \in \mathbb{R}$	BCE	KvsAll
ConEx (ours)	$\mathrm{Re}\Big(\langle \mathrm{conv}(\mathbf{e}_h, \mathbf{e}_r), \mathbf{e}_h, \mathbf{e}_r, \overline{\mathbf{e}_t} \rangle\Big)$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{C}$	BCE	KvsAll
QMult (ours)	$\mathbf{e}_h \otimes \mathbf{e}_r \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{H}$	BCE	KvsAll
OMult (ours)	$\mathbf{e}_h \bigstar \mathbf{e}_r \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{O}$	BCE	KvsAll
ConvQ (ours)	$\mathrm{conv}(\mathbf{e}_h, \mathbf{e}_r) \circ (\mathbf{e}_h \otimes \mathbf{e}_r) \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{H}$	BCE	KvsAll
ConvO (ours)	$\mathrm{conv}(\mathbf{e}_h, \mathbf{e}_r) \circ (\mathbf{e}_h \bigstar \mathbf{e}_r) \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{O}$	BCE	KvsAll

2.4.2 Training Strategies

There are three commonly used training strategies for KGE models.

Negative Sampling: Bordes et al. (2013) design a negative sampling technique via perturbing an entity in a randomly sampled triple, i.e., replacing an entity given a triple with a randomly sampled entity. In this setting, a triple $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$ is considered as a positive example, whilst $\{(\mathsf{h}, \mathsf{r}, \mathsf{x}) | \forall x \in \mathcal{E}\} \cup \{(\mathsf{x}, \mathsf{r}, \mathsf{t}) | \forall x \in \mathcal{E}\}$ is considered as a set of possible candidate negative examples. For each positive triple $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$, a negative triple is sampled from the set of corresponding candidate negative triples.

1vsAll: Lacroix et al. (2018) discard the idea of randomly sampling negative triples and propose 1vsAll the training strategy. For each $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$, all possible tail perturbed set of triples are considered as negative triples regardless of whether a perturbed triple exists in an input KG, i.e., $\{(\mathsf{h}, \mathsf{r}, \mathsf{x}) | \forall x \in \mathcal{E} : x \neq t\}$. Given that this setting does not involve negative triples via head perturbed entities, a data augmentation technique is applied to add inverse triples (t, r^{-1}, h) for each (h, r, t). r^{-1} is a syntactically generated relation denoting the inverse of a relation r, e.g., for (h, `bornIn`, t), (t, `reverse_bornIn`, t) obtained. Therefore, the size of the data is doubled. Their results showed that even simpler models reached state-of-the-art performance in link prediction task via 1vsAll.

KvsAll: Dettmers et al. (2018) propose KvsAll by extending 1vsAll via constructing multi-label binary vectors for each (h, r) Ruffinelli et al. (2019).[2] More specifically, a training data point consists of a pair (h, r) and a binary vector containing 1 for $\{x | x \in \mathcal{E} \wedge (\mathsf{h}, \mathsf{r}, \mathsf{x}) \in \mathcal{G}\}$ and 0s for other entities. Recent KGE models are commonly trained with KvsAll (Balazevic et al., 2019; Nguyen et al., 2018a; Demir and Ngonga Ngomo, 2021a; Ruffinelli et al., 2019; Balažević et al., 2019b).

KvsSample: We design a new training strategy called KvsSample that allows practitioners to find a middle ground between the negative sampling and KvsAll methods w.r.t. available computational budgets. As in KvsAll, a training data point consists of a pair (h, r) and a binary vector containing 1s for all $\{x | x \in \mathcal{E} \wedge (\mathsf{h}, \mathsf{r}, \mathsf{x}) \in \mathcal{G}\}$ and 0s for other entities. At each mini-batch construction, a binary label vector is dynamically constructed by up or subsampling 1s or 0s. This technique is elucidated in Chapter 8.

Optimization: During training with 1vsAll or KvsAll, for a given pair (h, r), predicted scores for all entities are computed, i.e., $\forall x \in \mathcal{E} : \phi((\mathsf{h}, \mathsf{r}, \mathsf{x}))) =: \mathbf{z} \in \mathbb{R}^{|\mathcal{E}|}$. Through the logistic sigmoid function $\sigma(\mathbf{z}) = \frac{1}{1+\exp(-\mathbf{z})}$, scores are normalized to obtain

[2]Note that the KvsAll strategy is called 1-N scoring in Dettmers et al. (2018) Here, we follow the terminology of Ruffinelli et al. (2019).

predicted likelihood of entities denoted by $\hat{\mathbf{y}}$. An incurred binary cross-entropy loss on a training data point is computed as

$$\ell(\hat{\mathbf{y}}, \mathbf{y}) = -\frac{1}{|\mathcal{E}|} \sum_{i=1}^{|\mathcal{E}|} \mathbf{y}^{(i)} \log(\hat{\mathbf{y}}^{(i)}) + \big(1 - \mathbf{y}^{(i)}\big) \log\big(1 - \hat{\mathbf{y}}^{(i)}\big), \tag{2.63}$$

where $\mathbf{y} \in [0,1]^{|\mathcal{E}|}$ is a binary sparse label vector. If $(\mathsf{h}, \mathsf{r}, \mathcal{E}_i) \in \mathcal{G}$, then $\mathbf{y}^{(i)} = 1$, otherwise $\mathbf{y}^{(i)} = 0$. Recent works show that this optimization process often leads to state-of-the-art link prediction performance (Balazevic et al., 2019; Ruffinelli et al., 2019).

Connection to our scientific contributions: In Chapter 4, Chapter 5, Chapter 6, Chapter 7, and Chapter 8, we focus on knowledge graph embedding models tackling the link prediction and/or relation prediction problems. In most of experiments, we use KvsAll as the training strategy and the binary cross-entropy loss function. In Chapter 9, we formulate the description logic concept learning problem as a multi-label classification problem from sets of input examples to pre-selected concepts, where the binary cross-entropy function is used as a loss function.

2.5 Description Logics, Knowledge Base and Concept Learning

2.5.1 Description Logics

A Description Logic (DL) is a decidable fragment of the First-Order Logic (FOL) that uses only unary and binary predicates (Baader et al., 2003). The set of unary predicates, binary predicates and constants correspond to the set of named concepts N_C, roles N_R, and individuals N_I of the DLs respectively. Throughout this work, we focus on DL $\mathcal{ALC}$ (Attributive Language with Complements) (Schmidt-Schauß and Smolka, 1991) as in most recent works (Bühmann et al., 2016; Kouagou et al., 2022a; Tran et al., 2017). The model-theoretic semantics of $\mathcal{ALC}$ are given in Table 2.3.

2.5.2 Knowledge Base

A Knowledge Base (KB) is often defined as a tuple of terminological axioms and assertions $\mathcal{K} = (Tbox, Abox)$ (Lehmann, 2010; Kouagou et al., 2022b). Therein, $Tbox$ denotes a set of terminological axioms describing relationships between named concepts N_C. Every terminological axiom is in the form of $A \sqsubseteq B$ or $A \equiv B$ s.t. $A, B \in N_C$.

Tbox is also refereed as an ontology (Schockaert et al., 2021a). *Abox* denotes a set of assertions describing relationships among individuals $a, b \in N_I$ via roles $r \in N_R$ as well as instantiation/membership relationships between N_I and N_C. Every assertion in *Abox* must in the form of $A(x)/(\mathsf{x}, \mathtt{type}, \mathsf{A})$ and $r(x, y)/(\mathsf{x}, \mathsf{r}, \mathsf{y})$, where $A \in N_C$, $r \in N_R$, and $x, y \in N_I$. A KB can always be translated into a KG and vice-versa. In a KB, an information is categorized as a terminological axiom or an assertion, whereas such dichotomy does not exists in a KG (see Section 2.3.1). Most KBs on the Web provide a large collections of facts in the form of assertions (Nickel et al., 2015). Yet, they often lack well-structured ontologies (Nickel et al., 2012; Schockaert et al., 2021a).

Table 2.3: $\mathcal{ALC}$ syntax and semantics. $\mathcal{I}$ stands for an interpretation, $\Delta^{\mathcal{I}}$ for its domain.

Construct	Syntax	Semantics
Atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Role	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Top concept	$\top$	$\Delta^{\mathcal{I}}$
Bottom concept	$\bot$	$\emptyset$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Existential restriction	$\exists\, r.C$	$\{x \mid \exists\, y.(x, y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
Universal restriction	$\forall\, r.C$	$\{x \mid \forall\, y.(x, y) \in r^{\mathcal{I}}$ implies $y \in C^{\mathcal{I}}\}$

2.5.3 Concept Learning

We define Concept Learning (CL) in a fashion akin to Lehmann and Hitzler (2010). Let $\mathcal{K} = (Tbox, Abox)$ over $\mathcal{ALC}$, the set E^+ of positive examples, and the set E^- of negative examples be given, where $E^+, E^- \subset N_I \ \wedge E^+ \cap E^- = \emptyset$. The goal of CL is to find a H such that fulfills the following conditions

$$\forall p \in E^+, \forall n \in E^- \big(\mathcal{K} \models H(p)) \wedge (\mathcal{K} \not\models H(n)\big), \tag{2.64}$$

where $H \in C$ denotes an $\mathcal{ALC}$ concept and C denotes all valid $\mathcal{ALC}$ concepts under the construction rules: $C ::= A \mid \neg C \mid C \sqcap C \mid C \sqcup C \mid \exists r.C \mid \forall r.C \mid$, where $A \in N_C$ and $r \in N_R$.

$\mathcal{K} \models H(p)$ implies that an inference of the class membership $H(p)$ is a logical consequence of $\mathcal{K}$. Note that CL is also commonly called as Class Expression Learning (CEL) since a DL concept can be translated to an Web Ontology Language (OWL) class expression (Lehmann, 2010; Lehmann et al., 2011). Checking whether a H fulfills Equation (2.64) is performed by a retrieval function $\mathcal{R}: \ C \rightarrow 2^{N_I}$ defined under Open World Assumption (OWA) or Close World Assumption (CWA). The CL problem is often transformed into a search problem within a quasi-ordered state space $(\mathcal{S}, \preceq)$ (Bühmann et al., 2018; Fanizzi et al., 2019; Lehmann et al., 2011; Lehmann and Hitzler, 2010; Tran et al., 2017; Kouagou et al., 2022a), where each state is an $\mathcal{ALC}$ concept. Traversing in $\mathcal{S}$ is commonly conducted via top-down (also called downward) refinement operators, which are defined as $\rho : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ with

$$\forall A \in \mathcal{S} : \rho(A) \subseteq \{B \in \mathcal{S} \mid B \preceq A\}. \tag{2.65}$$

State-of-the-art CL models begin their search towards a H, after a search tree is initialized with the most general DL concept ($\top$) as a root node. This search tree is iteratively built by selecting a node containing a quasi-ordered DL concept with the highest heuristic value and adding its qualifying refinements as its children into a search tree (Lehmann and Hitzler, 2010).

Heuristics. A heuristic function is the key to an efficient search in $\mathcal{S}$ towards a H (Lehmann et al., 2011). The number of explored concepts and runtimes are used as proxy for the efficiency. Various heuristic functions have been investigated (Lehmann et al., 2011; Westphal et al., 2021). Most heuristic functions of state-of-the-art models can be considered as myopic functions favoring syntactically short and accurate concepts. Hence, they are prone to stuck in a local optimum (Westphal et al., 2021). For instance, the heuristic function of CELOE is defined as

$$\phi_{\text{CELOE}}(A, B) = \text{Q}(B) + \lambda \cdot \big[\text{Q}(B) - \text{Q}(A)\big] - \beta \cdot |B|, \tag{2.66}$$

where $A \in \mathcal{S}$, $B \in \rho(A)$. $\beta > \lambda \geq 0$ and $\mathrm{Q}(\cdot)$ denotes a quality function (e.g., the F_1 score). Through $\mathrm{Q}(\cdot)$ and $|\cdot|$, the search is steered based on solely A and B towards more accurate and syntactically shorter concepts. $F_1(\cdot)$ is defined as

$$F_1(A) = \frac{|\ E^+ \cap \mathcal{R}(A)\ |}{|\ E^+ \cap \mathcal{R}(A)\ | + 0.5(|\ E^- \cap \mathcal{R}(A)\ | + |\ E^+ \setminus \mathcal{R}(A)\ |)}. \tag{2.67}$$

As the size of KB grows, runtimes of performing retrieval operations $\mathcal{R}(\cdot)$ increase (Bin et al., 2016, 2017; Lehmann, 2010). Traversing in $\mathcal{S}$ becomes a computational bottleneck on large KBs. Therefore, reducing the number of explored concepts plays an important role to tackle to tackle CL on KBs. Although state-of-the-art models (e.g. CELOE) apply redundancy elimination and expression simplification rules to reduce the number of explored concepts, impractical long runtimes of state-of-the-art models still prohibit large-scale applications (d'Amato, 2020; Hitzler et al., 2020).

The selected assumption underlying $\mathcal{R}(\cdot)$ also plays a role to tackle CL on large KBs. Due to the incomplete nature of KBs, OWA seems to be a more suitable assumption (Rudolph, 2011). Yet, Using OWA often makes membership queries (e.g., $H(\cdot)$) computationally more challenging (Fanizzi et al., 2008; Lehmann et al., 2011). Consequently, CWA is often adopted in many recent works (Heindorf et al., 2022; Kouagou et al., 2022a; Tran et al., 2017).

Connection to our scientific contributions: In Chapter 9, we formulate the concept learning problem as a multi-label classification problem. To tackle this problem, we propose a permutation-invariant neural embedding model that learns embeddings for sets of examples tailored towards F_1 scores of pre-selected description logic concepts w.r.t. input examples. In Chapter 12, we introduce an use case for applying a pretrained KGE model to tackle the concept learning problem.

2.6 Hypercomplex Numbers

2.6.1 Quaternions

The quaternions are a 4-dimensional normed division algebra (Hamilton, 1844). A quaternion number $Q \in \mathbb{H}$ is defined as $Q = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$, where a, b, c, d are real numbers and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are imaginary units satisfying Hamilton's rule: $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$. The quaternion multiplication of Q_1 and Q_2 is defined as

$$\begin{aligned} Q_1 \otimes Q_2 = &(a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2) \\ &+ (a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)\,\mathbf{i} \\ &+ (a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)\,\mathbf{j} \\ &+ (a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)\,\mathbf{k}. \end{aligned}$$

The quaternion multiplication is also known as the Hamilton product (Zhang et al., 2021). The inner product of two quaternions is defined as

$$Q_1 \cdot Q_2 = \langle a_1a_2 \rangle + \langle b_1b_2 \rangle + \langle c_1c_2 \rangle + \langle d_1d_2 \rangle.$$

For a d-dimensional quaternion vector a + b $\mathbf{i}$ + c $\mathbf{j}$ + d $\mathbf{k}$ with $a, b, c, d \in \mathbb{R}^d$, the inner product and multiplication is defined accordingly.

2.6.2 Octonions

The Octonions are an 8-dimensional algebra where an octonion number $O_1 \in \mathbb{O}$ is defined as $O_1 = x_0 + x_1\mathbf{e}_1 + x_2\mathbf{e}_2 + x_3\mathbf{e}_3 + x_4\mathbf{e}_4 + x_5\mathbf{e}_5 + x_6\mathbf{e}_6 + x_7\mathbf{e}_7$, where $\mathbf{e}_1, \mathbf{e}_2 \ldots \mathbf{e}_7$ are imaginary units (Baez, 2002). The inner product of two octonions $O_1 \cdot O_2 \in \mathbb{R}$ is obtained as defined by taking the inner products between corresponding scalars and imaginary units:

$$O_1 \cdot O_2 = x_0y_0 + x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4 + x_5y_5 + x_6y_6 + x_7y_7. \tag{2.68}$$

A d-dimensional octonion-valued vector is defined as $\{x_0 + x_1\mathbf{e}_1 + \cdots + x_7\mathbf{e}_7 : x_0, \ldots, x_7 \in \mathbb{R}^d\}$ with the vector operations being defined correspondingly to quaternions.

The octonion multiplication of O_1 and O_2 is defined as

$$\begin{aligned} O_1 \bigstar O_2 = \; & (x_0y_0 - x_1y_1 - x_2y_2 - x_3y_3 - x_4y_4 - x_5y_5 - x_6y_6 - x_7y_7) \\ & + (x_0y_1 + x_1y_0 + x_2y_3 - x_3y_2 + x_4y_5 - x_5y_4 - x_6y_7 + x_7y_6)\, \mathbf{e}_1 \\ & + (x_0y_2 - x_1y_3 + x_2y_0 + x_3y_1 + x_4y_6 + x_5y_7 - x_6y_4 - x_7y_5)\, \mathbf{e}_2 \\ & + (x_0y_3 + x_1y_2 - x_2y_1 + x_3y_0 + x_4y_7 - x_5y_6 + x_6y_5 - x_7y_4)\, \mathbf{e}_3 \\ & + (x_0y_4 - x_1y_5 - x_2y_6 - x_3y_7 + x_4y_0 + x_5y_1 + x_6y_2 + x_7y_3)\, \mathbf{e}_4 \\ & + (x_0y_5 + x_1y_4 - x_2y_7 + x_3y_6 - x_4y_1 + x_5y_0 - x_6y_3 + x_7y_2)\, \mathbf{e}_5 \\ & + (x_0y_6 + x_1y_7 + x_2y_4 - x_3y_5 - x_4y_2 + x_5y_3 + x_6y_0 - x_7y_1)\, \mathbf{e}_6 \\ & + (x_0y_7 - x_1y_6 + x_2y_5 + x_3y_4 - x_4y_3 - x_5y_2 + x_6y_1 + x_7y_0)\, \mathbf{e}_7. \end{aligned}$$

Connection to our scientific contributions: In Chapter 5 and Chapter 6, we design KGE models to learn complex-, quaternion-, and octonion-valued embeddings for entities and relations to advance the state of the art in benchmark tasks.

2.7 Hadamard and Kronecker Products

For any matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ and $\mathbf{Y} \in \mathbb{R}^{m \times n}$, the Hadamard product $\mathbf{X} \circ \mathbf{Y}$ is defined as

$$\mathbf{X} \circ \mathbf{Y} = \begin{bmatrix} \mathbf{X}_{11}\mathbf{Y}_{11} & \dots & \mathbf{X}_{1n}\mathbf{Y}_{1n} \\ \vdots & \ddots & \vdots \\ \mathbf{X}_{m1}\mathbf{Y}_{m1} & \dots & \mathbf{X}_{mn}\mathbf{Y}_{mn} \end{bmatrix} \in \mathbb{R}^{m \times n}, \tag{2.69}$$

where $\mathbf{X}_{ij}$ only interacts with $\mathbf{Y}_{ij}$. For any matrix $\mathbf{Z} \in \mathbb{R}^{p \times q}$, the Kronecker Product (KP) $\mathbf{X} \otimes \mathbf{Z}$ is a block matrix defined as

$$\mathbf{X} \otimes \mathbf{Z} = \begin{bmatrix} \mathbf{X}_{11}\mathbf{Z} & \dots & \mathbf{X}_{1n}\mathbf{Z} \\ \vdots & \ddots & \vdots \\ \mathbf{X}_{m1}\mathbf{Z} & \dots & \mathbf{X}_{mn}\mathbf{Z} \end{bmatrix} \in \mathbb{R}^{mp \times nq}, \tag{2.70}$$

where every element of $\mathbf{X}$ interacts with every element of $\mathbf{Z}$. In contrast to the Hadamard product, the Kronecker product is not commutative, i.e., $\mathbf{X} \otimes \mathbf{Z} \neq \mathbf{Z} \otimes \mathbf{X}$ most commonly holds. For more details, we refer to (Van Loan, 2000; Graham, 2018).

Connection to our scientific contributions: In Chapter 7, we design a technique based on Kronecker product to decrease the number of parameters in a knowledge graph embedding model, while retaining its effectiveness in the link prediction problem.

Part II

Contributions

3

A Physical Embedding Model for Knowledge Graphs

Preamble: This chapter is based on Demir and Ngonga Ngomo (2020).

Declaration of Authorship: The original idea was introduced by Axel-Cyrille Ngonga Ngomo. The research contributions were further developed by Caglar Demir and discussed with Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Axel-Cyrille Ngonga Ngomo. The code is available at https://github.com/dice-group/PYKE and https://github.com/dice-group/dice-embeddings.

Research Questions: In this work, we are concerned with the following two research questions:

1. **RQ1.** Can we design a knowledge graph embedding model to predict missing triples on large knowledge graphs?
2. **RQ2.** Can we answer RQ1. while ensuring the parameter efficiency of our models?

3.1 Methodology

In this chapter, we introduce Pyke. Here, our goal is to learn real-valued continuous vector representations for entities and relations of a Knowledge Graph (KG) such that similar entities and relations are assigned with similar embedding vectors. To this end, we design Pyke that is a physical model based on Hooke's law and its inverse with ideas from simulated annealing to compute embeddings for KGs efficiently. We prove that Pyke achieves a linear time complexity of each of its iterations in the size of the input KG. The architecture of Pyke allows practitioners to incorporate their domain knowledge in the learning process my means of selecting a similarity function. Our findings suggest that Pyke outperforms many state-of-the-art models in type prediction and cluster purity downstream tasks, while attaining a superior runtime performance. In Table 3.1, we summarize the symbols used in this chapter.

Table 3.1: An overview of our notation used in Chapter 3.

Notation	Description
$\mathcal{G}$	An RDF knowledge graph
$\mathcal{R}, \mathcal{P}, \mathcal{B}, \mathcal{L}$	Set of all RDF resources, predicates, blank nodes and literals, respectively
$\mathcal{S}$	Set of all RDF subjects with type information
$\mathcal{V}, \sigma$	Vocabulary of $\mathcal{G}$ and similarity function on $\mathcal{V}$, respectively
$\overrightarrow{x}_t$	Embedding of x at time t
F_a, F_r	Attractive and repulsive forces based on the Hooke's law, respectively
$PPMI(\cdot, \cdot)$	The positive pointwise mutual information function
K	Threshold for positive and negative examples
P	Function mapping each $x \in \mathcal{V}$ to a set of attracting elements of $\mathcal{V}$
N	Function mapping each $x \in \mathcal{V}$ to a set of repulsive elements of $\mathcal{V}$
$\omega, \mathcal{E}$	Repulsive constant and System energy, respectively
ϵ	Upper bound on alteration of locations of $x \in \mathcal{V}$ across two iterations
Δe	Energy release
$\mathcal{J}$	Objective function

3.1.1 RDF Knowledge Graphs

In this work, we focus on computing embeddings for RDF KGs. Let $\mathcal{R}$ be the set of all RDF resources, $\mathcal{B}$ be the set of all RDF blank nodes, $\mathcal{P} \subseteq \mathcal{R}$ be the set of all properties

and $\mathcal{L}$ denote the set of all RDF literals. An RDF KG $\mathcal{G}$ is a set of RDF triples (s, p, o) where $s \in \mathcal{R} \cup \mathcal{B}$, $p \in \mathcal{P}$ and $o \in \mathcal{R} \cup \mathcal{B} \cup \mathcal{L}$. We aim to compute embeddings for resources and blank nodes. Hence, we define the *vocabulary* of an RDF knowledge graph $\mathcal{G}$ as $\mathcal{V} = \{x : x \in \mathcal{R} \cup \mathcal{P} \cup \mathcal{B} \wedge \exists(s, p, o) \in \mathcal{G} : x \in \{s, p, o\}\}$. Essentially, $\mathcal{V}$ stands for all the URIs and blank nodes found in $\mathcal{G}$. Finally, we define the *subjects with type information* of $\mathcal{G}$ as $\mathcal{S} = \{x : x \in \mathcal{R} \setminus \mathcal{P} \wedge (x, \texttt{rdf:type}, o) \in \mathcal{G}\}$, where `rdf:type` stands for the instantiation relation in RDF.

3.1.2 Hooke's Law

Hooke's law describes the relation between a deforming force on a spring and the magnitude of the deformation within the elastic regime of said spring. The increase of a deforming force on the spring is linearly related to the increase of the magnitude of the corresponding deformation. Formally, Hooke's law can be expressed as follows:

$$F = -k\,\Delta \tag{3.1}$$

where F is the deforming force, Δ is the magnitude of deformation and k is the spring constant. Let us assume two points of unit mass located at x and y respectively, e.g., embeddings of two entities in a vector space. We assume that the two points are connected by an ideal spring with a spring constant k, an infinite elastic regime and an initial length of 0. Then, the force they are subjected to has a magnitude of $k||x - y||$. Note that the magnitude of this force grows with the distance between the two mass points. The inverse of Hooke's law, where

$$F_{\text{inverse}} = -\frac{k}{\Delta} \tag{3.2}$$

has the opposite behavior. It becomes weaker with the distance between the two mass points it connects.

3.1.3 Positive Pointwise Mutual Information

The Positive Pointwise Mutual Information (PPMI) is a means to capture the strength of the association between two events (e.g., two entities appearing in a triple of a KG).

Let a and b be two events. Let $\mathbb{P}(a, b)$ stand for the joint probability of a and b, $\mathbb{P}(a)$ for the probability of a and $\mathbb{P}(b)$ for the probability of b. Then, $\text{PPMI}(a, b)$ is defined as

$$\text{PPMI}(a, b) = \max\left(0, \log \frac{\mathbb{P}(a, b)}{\mathbb{P}(a)\mathbb{P}(b)}\right), \tag{3.3}$$

3.1.4 Intuition

Pyke is an iterative approach that aims to represent each element x of the vocabulary $\mathcal{V}$ of an input KG $\mathcal{G}$ as a vector in the n-dimensional space $\mathbb{R}^n$. Our approach begins by assuming that each element of $\mathcal{V}$ is mapped to a single point (i.e., its *embedding*) of unit mass whose location can be expressed via an n-dimensional vector in $\mathbb{R}^n$ according to an initial (e.g., random) distribution at iteration $t = 0$. In the following, we will use $\overrightarrow{x}_t$ to denote the embedding of $x \in \mathcal{V}$ at iteration t. We also assume a similarity function $\sigma : \mathcal{V} \times \mathcal{V} \rightarrow [0, \infty)$ (e.g., a PPMI-based similarity) over $\mathcal{V}$ to be given. Simply put, our goal is to improve this initial distribution iteratively over a predefined maximal number of iterations (denoted T) by ensuring that

1. the embeddings of similar elements of $\mathcal{V}$ are close to each other while
2. the embeddings of dissimilar elements of $\mathcal{V}$ are distant from each other.

Let $d : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^+$ be the distance (e.g., the Euclidean distance) between two embeddings in $\mathbb{R}^n$. According to our goal definition, a good iterative embedding approach should have the following characteristics:

C_1: If $\sigma(x, y) > 0$, then $d(\overrightarrow{x}_t, \overrightarrow{y}_t) \leq d(\overrightarrow{x}_{t-1}, \overrightarrow{y}_{t-1})$. This means that the embeddings of similar terms should become more similar with the number of iterations. The same holds the other way around:

C_2: If $\sigma(x, y) = 0$, then $d(\overrightarrow{x}_t, \overrightarrow{y}_t) \geq d(\overrightarrow{x}_{t-1}, \overrightarrow{y}_{t-1})$.

We translate C_1 into our model as follows: If x and y are similar (i.e., if $\sigma(x, y) > 0$), then a force $F_a(\overrightarrow{x}_t, \overrightarrow{y}_t)$ of attraction must exist between the masses which stand for x and y at any time t. $F_a(\overrightarrow{x}_t, \overrightarrow{y}_t)$ must be proportional to $d(\overrightarrow{x}_t, \overrightarrow{y}_t)$, i.e., the attraction between must grow with the distance between ($\overrightarrow{x}_t$ and $\overrightarrow{y}_t$). These conditions are fulfilled by setting the following force of attraction between the two masses:

$$||F_a(\overrightarrow{x}_t, \overrightarrow{y}_t)|| = \sigma(x, y) \times d(\overrightarrow{x}_t, \overrightarrow{y}_t). \tag{3.4}$$

From the perspective of a physical model, this is equivalent to placing a spring with a spring constant of $\sigma(x, y)$ between the unit masses which stand for x and y. At time t, these masses are hence accelerated towards each other with a total acceleration proportional to $||F_a(\overrightarrow{x}_t, \overrightarrow{y}_t)||$. The translation of C_2 into a physical model is as follows: If x and y are not similar (i.e., if $\sigma(x, y) = 0$), we assume that they are dissimilar. Correspondingly, their embeddings should diverge with time. The magnitude of the repulsive force between the two masses representing x and y should be strong if the masses are close to each other and should diminish with the distance between the two masses. We can fulfill this condition by setting the following repulsive force between the two masses:

$$||F_r(\overrightarrow{x}_t, \overrightarrow{y}_t)|| = -\frac{\omega}{d(\overrightarrow{x}_t, \overrightarrow{y}_t)}, \tag{3.5}$$

where $\omega > 0$ denotes a constant, which we dub the repulsive constant. At iteration t, the embeddings of dissimilar terms are hence accelerated away from each other with a total acceleration proportional to $||F_r(\overrightarrow{x}_t, \overrightarrow{y}_t)||$. This is the inverse of Hooke's law, where the magnitude of the repulsive force between the mass points which stand for two dissimilar terms decreases with the distance between the two mass points.

Based on these intuitions, we can now formulate the goal of Pyke formally: We aim to find embeddings for all elements of $\mathcal{V}$ which minimize the total distance between similar elements and maximize the total distance between dissimilar elements. Let $P : \mathcal{V} \to 2^{\mathcal{V}}$ be a function which maps each element of $\mathcal{V}$ to the subset of $\mathcal{V}$ it is similar to. Analogously, let $N : \mathcal{V} \to 2^{\mathcal{V}}$ map each element of $\mathcal{V}$ to the subset of $\mathcal{V}$ it is dissimilar to. Pyke aims to optimize the following objective function:

$$J(\mathcal{V}) = \left(\sum_{x \in \mathcal{V}} \sum_{y \in P(x)} d(\overrightarrow{x}, \overrightarrow{y}) \right) - \left(\sum_{x \in \mathcal{V}} \sum_{y \in N(x)} d(\overrightarrow{x}, \overrightarrow{y}) \right). \tag{3.6}$$

Pyke implements the intuition described above as follows: Given an input KG $\mathcal{G}$, Pyke first constructs a symmetric similarity matrix $\mathcal{A}$ of dimensions $|\mathcal{V}| \times |\mathcal{V}|$. We will use $a_{x,y}$ to denotes the similarity coefficient between $x \in \mathcal{V}$ and $y \in \mathcal{V}$ stored in $\mathcal{A}$. Pyke truncates this matrix to (1) reduce the effect of oversampling and (2) accelerate subsequent computations. The initial embeddings of all $x \in \mathcal{V}$ in $\mathbb{R}^n$ are then determined. Subsequently, Pyke uses the physical model described above to improve the embeddings iteratively. The iteration is ran at most T times or until the objective

function $J(\mathcal{V})$ stops decreasing. In the following, we explain each of the steps of the approach in detail. We use the RDF graph shown in Figure 3.1 as a running example.[1]

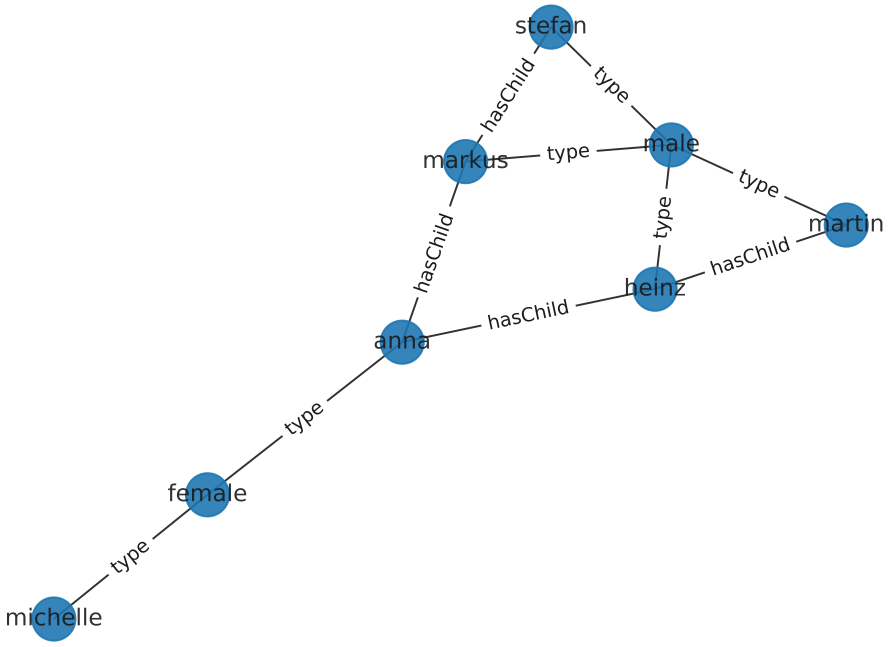


Figure 3.1: A visualization of an RDF knowledge graph.

Building the similarity matrix: For any two elements $x, y \in \mathcal{V}$, we set $a_{x,y} = \sigma(x, y) = \text{PPMI}(x, y)$ in our current implementation. We compute the probabilities $\mathbb{P}(x)$, $\mathbb{P}(y)$ and $\mathbb{P}(x, y)$ as follows:

$$\mathbb{P}(x) = \frac{|\{(s, p, o) \in \mathcal{G} : x \in \{s, p, o\}\}|}{|\{(s, p, o) \in \mathcal{G}\}|}. \tag{3.7}$$

Similarly,

$$\mathbb{P}(y) = \frac{|\{(s, p, o) \in \mathcal{G} : y \in \{s, p, o\}\}|}{|\{(s, p, o) \in \mathcal{G}\}|}. \tag{3.8}$$

Finally,

$$\mathbb{P}(x, y) = \frac{|\{(s, p, o) \in \mathcal{G} : \{x, y\} \subseteq \{s, p, o\}\}|}{|\{(s, p, o) \in \mathcal{G}\}|}. \tag{3.9}$$

[1]This example is provided as an example in the DL-Learner framework at http://dl-learner.org.

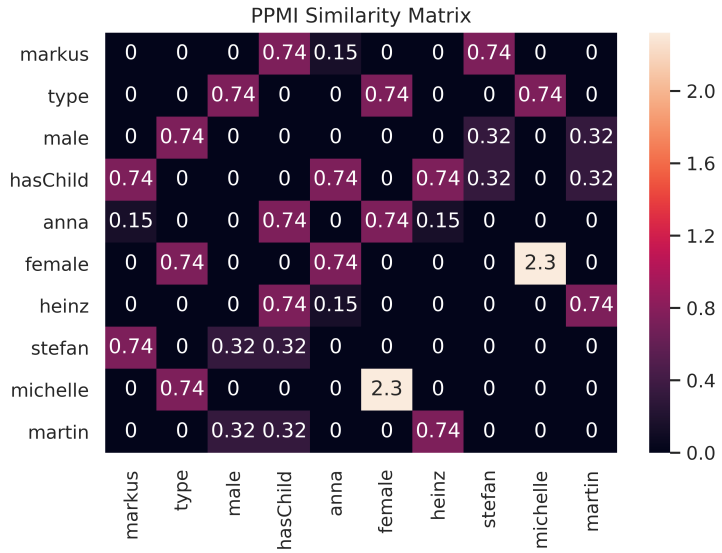


Figure 3.2: A visualization of the PPMI similarity matrix of resources in the RDF knowledge graph shown in Figure 3.1.

For our running example (see Figure 3.1), Pyke constructs the similarity matrix shown in Figure 3.2. Note that our framework can be combined with any similarity function σ. Exploring other similarity function is out the scope of this paper but will be at the center of future works.

Computing P and N: To avoid oversampling positive or negative examples, we only use a portion of $\mathcal{A}$ for the subsequent optimization of our objective function. For each $x \in \mathcal{V}$, we begin by computing $P(x)$ by selecting K resources which are most similar to x. Note that if less than K resources have a non-zero similarity to x, then $P(x)$ contains exactly the set of resources with a non-zero similarity to x. Thereafter, we sample K elements y of $\mathcal{V}$ with $a_{x,y} = 0$ randomly. We call this set $N(x)$. For all $y \in N(x)$, we set $a_{x,y}$ to $-\omega$, where ω is our repulsive constant. The values of $a_{x,y}$ for $y \in P(x)$ are preserved. All other values are set to 0. After carrying out this process for all $x \in \mathcal{V}$, each row of $\mathcal{A}$ now contains exactly $2K$ non-zero entries provided that each $x \in \mathcal{V}$ has at least K resources with non-zero similarity. Given that $K << |\mathcal{V}|$,

$\mathcal{A}$ is now sparse and can be stored accordingly.[2] The PPMI similarity matrix for our example graph is shown in Figure 3.2.

Initializing the embeddings: Each $x \in \mathcal{V}$ is mapped to a single point $\overrightarrow{x}_t$ of unit mass in $\mathbb{R}^n$ at iteration $t = 0$. As exploring sophisticated initialization techniques is out of the scope of this paper, the initial vector is set randomly.[3] Figure 3.3 shows a 3D projection of the initial embeddings for our running example (with $n = 50$).

Iteration: This is the crux of our approach. In each iteration t, our approach assumes that the elements of $P(x)$ attract x with a total force

$$F_a(\overrightarrow{x}_t) = \sum_{y \in P(x)} \sigma(x, y) \times (\overrightarrow{y}_t - \overrightarrow{x}_t). \tag{3.10}$$

On the other hand, the elements of $N(x)$ repulse x with a total force

$$F_r(\overrightarrow{x}_t) = - \sum_{y \in N(x)} \frac{\omega}{(\overrightarrow{y}_t - \overrightarrow{x}_t)}. \tag{3.11}$$

We assume that exactly one unit of time elapses between two iterations. The embedding of x at iteration $t + 1$ can now be calculated by displacing $\overrightarrow{x}_t$ proportionally to $(F_a(\overrightarrow{x}_t) + F_r(\overrightarrow{x}_t))$.However, implementing this model directly leads to a chaotic (i.e., non-converging) behavior in most cases. We enforce the convergence using an approach borrowed from simulated annealing, i.e., we reduce the total energy of the system by a constant factor Δe after each iteration. By these means, we can ensure that our approach always terminates, i.e., we can iterate until $J(\mathcal{V})$ does not decrease significantly or until a maximal number of iterations T is reached.

Implementation: Algorithm 1 shows the pseudocode of our approach. Pyke updates the embeddings of vocabulary terms iteratively until one of the following two stopping criteria is satisfied: Either the upper bound on the iterations T is met or a

[2] We use $\mathcal{A}$ for the sake of explanation. For practical applications, this step can be implemented using priority queues, hence making quadratic space complexity for storing $\mathcal{A}$ unnecessary.

[3] Preliminary experiments suggest that applying a singular value decomposition on $\mathcal{A}$ and initializing the embeddings with the latent representation of the elements of the vocabulary along the n most salient eigenvectors has the potential of accelerating the convergence of our approach.

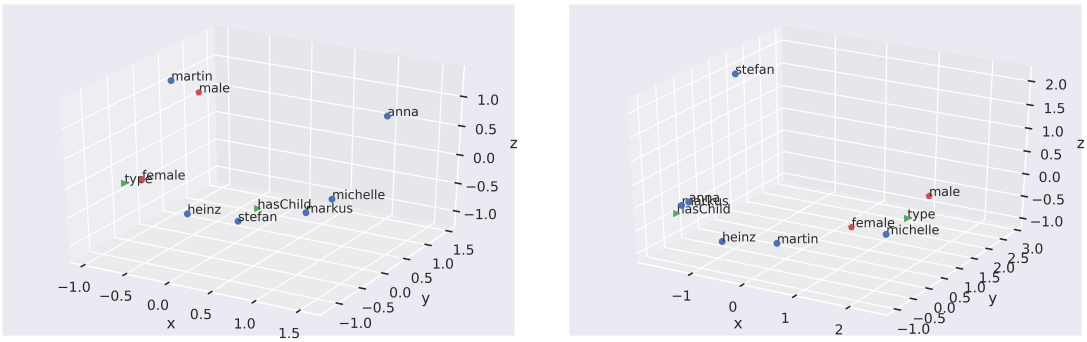


Figure 3.3: A visualization of the 2D PCA projected 50-dimensional embeddings for our running example. Left are the randomly initialized embeddings. The figure on the right shows the 50-dimensional Pyke embedding vectors for our running example after convergence. Pyke was configured with $K = 3$, $\omega = -0.3$, $\Delta e = 0.06$ and $\epsilon = 10^{-3}$.

lower bound ϵ on the total change in the embeddings (i.e., $\sum_{x \in \mathcal{V}} ||\overrightarrow{x}_t - \overrightarrow{x}_{t-1}||$) is reached. A gradual reduction in the system energy $\mathcal{E}$ inherently guarantees the termination of the process of learning embeddings. A 3D projection of the resulting embedding for our running example is shown in Figure 3.3.

3.1.5 Complexity Analysis

Space complexity: Let $m = |\mathcal{V}|$. We would need at most $\frac{m(m-1)}{2}$ entries to store $\mathcal{A}$, as the matrix is symmetric and we do not need to store its diagonal. However, there is actually no need to store $\mathcal{A}$ explicitly. $P(x)$ can be implemented as a priority queue of size K in which the indexes of K elements of $\mathcal{V}$ most similar to x as well as their similarity to x are stored. $N(x)$ can be implemented as a buffer of size K which contains only indexes. Once $N(x)$ reaches its maximal size K, then new entries (i.e., y with PPMI(x, y)) are added randomly. Hence, we need $O(Kn)$ space to store both P and N. Note that $K << m$. The embeddings require exactly $2mn$ space as we store $\overrightarrow{x}_t$ and $\overrightarrow{x}_{t-1}$ for each $x \in \mathcal{V}$. The force vectors F_a and F_r each require a space of n. Hence, the space complexity of Pyke lies clearly in $O(mn + Kn)$ and is hence linear w.r.t. the size of the unique entities and relations when the number n of dimensions of the embeddings and the number K of positive and negative examples are fixed.

Time complexity: Initializing the embeddings requires mn operations. The initialization of P and N can also be carried out in linear time. Adding an element to P and

Algorithm 1 Pyke

Require: $T, \mathcal{V}, K, \epsilon, \Delta e, \omega, n$
  //initialize embeddings
  for each x in $\mathcal{V}$ **do**
    $\overrightarrow{x}_0$ = random vector in $\mathbb{R}^n$;
  end for
  //initialize similarity matrix
  $\mathcal{A}$ = new Matrix[$|\mathcal{V}|$][$|\mathcal{V}|$];
  for each x in $\mathcal{V}$ **do**
    for each y in $\mathcal{V}$ **do**
      $\mathcal{A}_{xy} = PPMI(x, y)$;
    end for
  end for
  // perform positive and negative sampling
  for each x in $\mathcal{V}$ **do**
    $P(x)$ = getPositives($\mathcal{A}, x, K$) ;
    $N(x)$ = getNegatives($\mathcal{A}, x, K$) ;
  end for
  // iteration
  $t = 1$;
  $\mathcal{E} = 1$;
  while $t < T$ **do**
    for each x in $\mathcal{V}$ **do**
      // Simulation of elementwise pull
      $F_a = \sum_{y \in P(x)} \sigma(x, y) \times (\overrightarrow{y}_{t-1} - \overrightarrow{x}_{t-1})$;
      // Simulation of elementwise push
      $F_r = -\sum_{y \in N(x)} \frac{\omega}{\overrightarrow{y}_{t-1} - \overrightarrow{x}_{t-1}}$;
      // Parameter Update
      $\overrightarrow{x}_t = \overrightarrow{x}_{t-1} + \mathcal{E} \times (F_a + F_r)$;
    end for
    $\mathcal{E} = \mathcal{E} - \Delta e$;
    if $\sum_{x \in \mathcal{V}} ||\overrightarrow{x}_t - \overrightarrow{x}_{t-1}|| < \epsilon$ **then**
      break
    end if
    $t = t + 1$;
  end while
  return Embeddings $\overrightarrow{x}_t$

N is carried out at most m times. For each x, the addition of an element to $P(x)$ has a runtime of at most K. Adding elements to $N(x)$ is carried out in constant time, given that the addition is random. Hence the computation of $P(x)$ and $N(x)$ can be carried out in linear time w.r.t. m. This computation is carried out m times, i.e., once for each x. Hence, the overall runtime of the initialization for Pyke is on $O(m^2)$. Importantly, the update of the position of each x can be carried out in $O(K)$, leading to each iteration having a time complexity of $O(mK)$. The total runtime complexity for the iterations is hence $O(mKT)$, which is linear in m. This result is of central importance for our subsequent empirical results, as the iterations make up the bulk of Pyke's runtime. Hence, Pyke's runtime should be close to linear in real settings.

3.2 Experiments & Results

3.2.1 Experiments

Experimental Setup

The goal of our evaluation was to compare the quality of the embeddings generated by Pyke with the state of the art. We used two evaluation scenarios. In the first scenario, we measured the type homogeneity of the embeddings generated by the KGE approaches we considered. We achieved this goal by using a scalable approximation of DBSCAN dubbed HDBSCAN (Campello et al., 2013). In our second evaluation scenario, we compared the performance of Pyke on the type prediction task against that of 6 state-of-the-art algorithms. In both scenarios, we only considered embeddings of the subset $\mathcal{S}$ of $\mathcal{V}$ as done in previous works (Melo et al., 2016; Thoma et al., 2017). All experiments were carried out on Ubuntu 18.04 with 126 GB RAM with 16 Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz processors. We provide hyperparameter optimization, training and evaluation scripts at https://github.com/dice-group/PYKE.

Datasets

We used six datasets (2 real, 4 synthetic) throughout our experiments. An overview of the datasets used in our experiments is shown in Table 3.2. Drugbank[4] is a small-scale

[4]download.bio2rdf.org/#/release/4/drugbank

KG, whilst the DBpedia (version 2016-10) dataset is a large cross-domain dataset.[5] The *four synthetic datasets* were generated using the LUBM generator (Guo et al., 2005) with 100, 200, 500 and 1000 universities.

Table 3.2: An overview of RDF datasets used in our experiments. $|\mathcal{G}|$, $|\mathcal{V}|$, $|\mathcal{S}|$, and $|C|$ denote the number of triples, the number of unique entities and relations, the number of unique entities having type information, and the number of unique RDF classes, respectively.

Dataset	$\lvert\mathcal{G}\rvert$	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert C\rvert$
Drugbank	3,146,309	521,428	421,121	102
DBpedia	27,744,412	7,631,777	6,401,519	423
LUBM100	9,425,190	2,179,793	2,179,766	14
LUBM200	18,770,356	4,341,336	4,341,309	14
LUBM500	46,922,188	10,847,210	10,847,183	14
LUBM1000	93,927,191	21,715,108	21,715,081	14

Evaluation Metrics

We evaluated the homogeneity of embeddings by measuring the purity of the clusters generated by HDBSCAN (Campello et al., 2013). The original cluster purity equation assumes that each element of a cluster is mapped to exactly one class (Manning et al., 2010). Given that a single resource can have several types in a knowledge graph (e.g., `BarackObama` is a person, a politician, an author and a president in DBpedia), we extended the cluster purity equation as follows: Let $T = \{t_1, t_2, \ldots\}$ be the set of all class types found in $\mathcal{G}$. Each $x \in \mathcal{S}$ was mapped to a binary type vector $type(x)$ of length $|T|$. The i-th entry of $type(x)$ was 1 iff x was of type t_i. In all other cases, t_i was set to 0. Based on these premises, we computed the purity of a clustering as follows:

$$\text{Purity} = \sum_{i=1}^{|C|} \frac{1}{|C|^2} \sum_{x \in C_i} \sum_{y \in C_i} cos\Big(type(x), type(y)\Big), \tag{3.12}$$

[5] Note that we compile the DBpedia datasets by merging the dumps of `mapping-based objects`, `skos categories` and `instance types` provided in the DBpedia download folder for version 2016-10 at downloads.dbpedia.org/2016-10.

where $C := [C_1 \dots C_L]$ denotes the clusters computed by HDBSCAN, where $x \in C_i$ denotes a entity being element of in the i-th cluster. A high purity means that resources with similar type vectors (e.g., presidents who are also authors) are located close to each other in the embedding space, which is a wanted characteristic of a KGE.

In our second evaluation, we performed a type prediction experiment in a manner akin to (Melo et al., 2016; Thoma et al., 2017). For each resource $x \in \mathcal{S}$, we used the μ closest embeddings of x to predict x's type vector. We then compared the average of the types predicted with x's known type vector using the cosine similarity:

$$\text{prediction score} = \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} cos\Big(type(x), \sum_{y \in \mu nn(x)} type(y)\Big), \tag{3.13}$$

where $\mu nn(x)$ stands for the μ nearest neighbors of x.

Hyperparameter Optimization

We set $K = 45$, $\Delta e = 0.0414$ and $\omega = 1.45557$ throughout our experiments. The values were computed using a Sobol Sequence optimizer (Saltelli et al., 2010). We searched several different nearest neighbors at type prediction $\mu \in \{1, 3, 5, 10, 15, 30, 50, 100\}$ in our experiments. Preliminary experiments showed that performing the cluster purity and type prediction evaluations on embeddings of large KGs is prohibited by the long runtimes of the clustering algorithm. For instance, HDBSCAN did not terminate in 20 hours of computation when $|\mathcal{S}| > 6 \times 10^6$. Consequently, we had to apply HDBSCAN on embeddings on the subset of $\mathcal{S}$ on DBpedia which contained resources of type `Person` or `Settlement`. The resulting subset of $\mathcal{S}$ on DBpedia consists of 428, 289 RDF resources. For the type prediction task, we sampled 10^5 resources from $\mathcal{S}$ according to a random distribution and fixed them across the type prediction experiments for all KGE models.

3.2.2 Results

Table 3.3 reports cluster purity results for all competing approaches. Pyke achieves a cluster purity of 0.75 on Drugbank and clearly outperforms all other approaches. DBpedia turned out to be a more difficult dataset. Still, Pyke outperformed models by between 11% and 26% on Drugbank and between 9% and 23% on DBpedia.

Table 3.3: Cluster purity results. The best results are marked in bold. Experiments marked with * did not terminate after 24 hours of computation.

Approach	Drugbank	DBpedia
Pyke	**0.75**	**0.57**
Word2vec	0.43	0.37
ComplEx	0.64	*
RESCAL	*	*
TransE	0.60	0.48
CP	0.49	0.41
DistMult	0.49	0.34

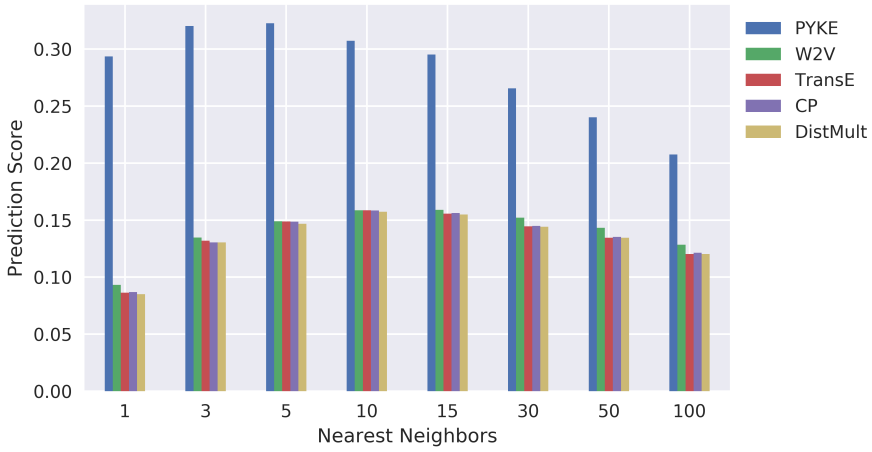


Figure 3.4: Mean results on type prediction scores on 10^5 randomly sampled entities of DBpedia.

Note that in 3 cases, the implementations available were unable to complete the computation of embeddings within 24 hours.

Figure 3.4 and Figure 3.5 show our type prediction results on the Drugbank and DBpedia datasets. Pyke outperforms all state-of-the-art approaches across all experiments. In particular, it achieves a margin of up to 22% (absolute) on Drugbank and 23% (absolute) on DBpedia. Like in the previous experiment, all KGE approaches perform worse on DBpedia, with prediction scores varying between < 0.1 and 0.32.

Table 3.4 show runtime performances of all models on the two real benchmark datasets, while Figure 3.6 display the runtime of Pyke on the synthetic LUBM datasets. Our results support our original hypothesis. The low space and time complexities of

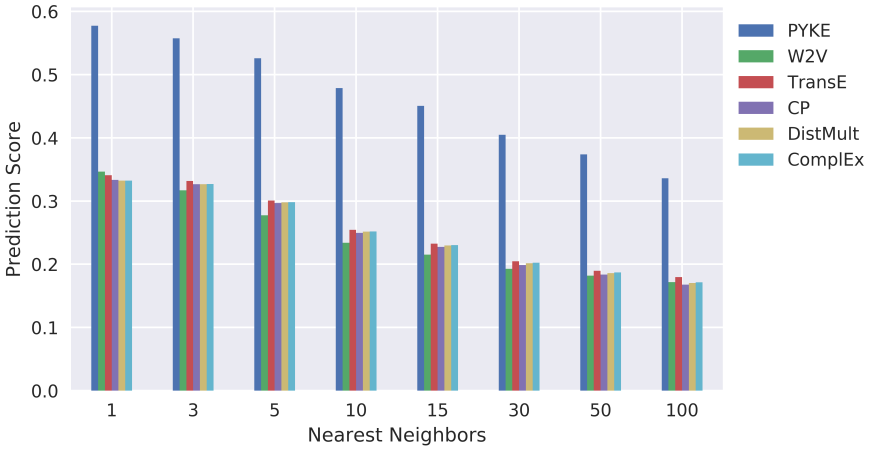


Figure 3.5: Mean of type prediction scores on all entities of Drugbank.

Table 3.4: Runtime performances (in minutes) of all competing approaches. All approaches were executed three times on each dataset. The reported results are the mean and standard deviation of the last two runs. The best results are marked in bold. Experiments marked with * did not terminate after 24 hours of computation.

Approach	Drugbank	DBpedia
Pyke	**25** ± 1	**309** ± 1
Word2vec	41 ±1	420 ±1
ComplEx	705 ± 1	*
RESCAL	*	*
TransE	68 ± 1	685 ± 1
CP	230 ± 1	1154 ± 1
DistMult	210 ± 1	1030 ± 1

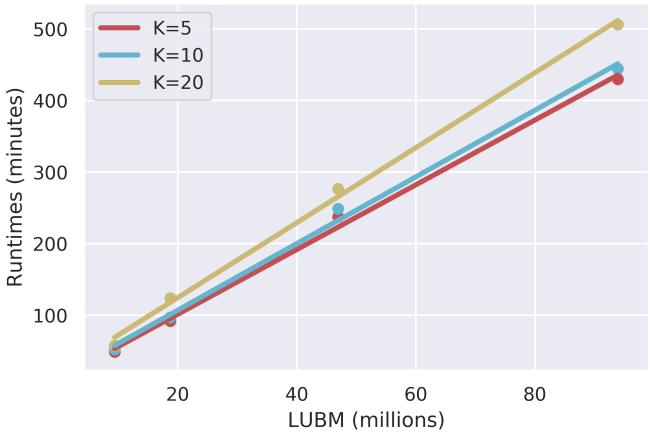


Figure 3.6: Runtime performances of Pyke on synthetic KGs. Colored lines represent fitted linear regressions with fixed K values of Pyke.

Pyke mean that it runs efficiently: Our approach achieves runtimes of only 25 minutes on Drugbank and 309 minutes on DBpedia, while outperforming all other approaches by up to 14 hours in runtime. In addition to evaluating the runtime of Pyke on synthetic data, we were interested in determining its behavior on datasets of growing sizes. We used LUBM datasets and computed a linear regression of the runtime. The runtime results for this experiment are shown in Figure 3.6. The linear fit shown in Table 3.5 achieves R^2 values beyond 0.99, which points to a clear linear fit between Pyke's runtime and the size of the input dataset.

We believe that the good performance of Pyke stems from (1) its sampling procedure and (2) its being akin to a physical simulation. Employing PPMI to quantify the similarity between resources seems to yield better sampling results than generating negative examples using the *local closed word assumption* that underlies sampling procedures of all of competing state-of-the-art KG models. More importantly, *positive and negative sampling occur in our approach per resource rather than per RDF triple.* Therefore, Pyke is able to leverage more from negative and positive sampling. By virtue of being akin to a physical simulation, Pyke is able to run efficiently even when each resource x is mapped to 45 attractive and 45 repulsive resources (see Table 3.4) whilst all state-of-the-art KGE required more computation time.

Table 3.5: Results of fitting linear regression on runtimes.

K	Coefficient	Intercept	R^2
5	4.52	10.74	0.997
10	4.65	13.64	0.996
20	5.23	19.59	0.997

3.2.3 Discussion

By virtue of being akin to a physical simulation, Pyke retains a linear space complexity. This was proven through a complexity analysis of our approach. While the time complexity of the approach is quadratic due to the computation of P and N, all other steps are linear in their runtime complexity. Hence, we expected our approach to behave closes to linearly. Our evaluation on LUBM datasets suggests that this is indeed the case and the runtime of Pyke grows close to linearly. This is an important result, as it means that our approach can be used on very large knowledge graphs and return results faster than popular algorithms such as Word2vec and TransE. However, time efficiency is not all. Our results suggest that Pyke outperforms state-of-the-art approaches in the two tasks of type prediction and clustering. Hence, our results confirm that the two research questions can be answered via Pyke.

Trade-offs & Possible Improvements: Most KGE models (see Section 2.4) can predict missing triples by means of assigning a likelihood score for a given triple. Although embeddings learned with Pyke can be directly used to quantify the similarity between entities and relations, Pyke cannot assign a score for a given triple.

In Algorithm 1, embeddings are updated with a gradually decreasing learning/update rate $\mathcal{E} - \Delta e$. Using line-search techniques to automatically find a good learning rate w.r.t. the current input may further improve the performance (Vaswani et al., 2019). Moreover, enriching this update rule with the bias corrected momentum update (i.e., Adam Equation (2.54)) may also improve the performance.

4

A Shallow Neural Model for Relation Prediction

Preamble: This chapter is based on Demir et al. (2021b).

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Diego Moussallem and Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Diego Moussallem and Axel-Cyrille Ngonga Ngomo. The code is available at https://github.com/dice-group/Shallom and https://github.com/dice-group/dice-embeddings.

Research Questions: In this work, we are concerned with the following two research questions:

1. **RQ1.** Can we design a knowledge graph embedding model to predict missing triples on large knowledge graphs?
2. **RQ2.** Can we answer RQ1. while ensuring the parameter efficiency of our models?

4.1 Methodology

In this chapter, we introduce Shallom. Here, our goal is to design a KGE model that efficiently learns real-valued continuous vector representations (embeddings) for entities and relations of a Knowledge Graph (KG), while alleviating the inability of Pyke in predicting missing triples. Shallom learns embeddings for entities and relations tailored towards relation prediction. Given a pair of entities, Shallom predicts missing relations via two affine transformations with non-linear activation functions. Hence, Shallom is analogous to a very-well-known word embedding model (Word2Vec (Mikolov et al., 2013b)), since both model predict a central vocabulary term given surrounding terms. In Table 4.1, we summarize the symbols used in this chapter.

Table 4.1: An overview of our notation used in Chapter 4.

Notation	Description
$\mathcal{G}$	A knowledge graph
$\mathcal{E}, \mathcal{R}$	Sets of unique entities and relations in $\mathcal{G}$, respectively
$\mathcal{D}$	Training dataset
$\sigma(\cdot)$	The sigmoid function
$\text{ReLU}(\cdot)$	The rectified linear unit function
d	The size of embedding vectors for entities
Θ	Model parameters
$\text{concat}(\cdot, \cdot)$	Concatenation operation
$\mathbf{H}, \mathbf{b}_1$	An affine transformation
$\mathbf{W}, \mathbf{b}_2$	An affine transformation
$\mathbf{E}$	An entity embedding matrix
$\ell(\cdot, \cdot)$	Binary cross-entropy loss function

We define Shallom as follows

$$\textsc{Shallom}(s, o)_\Theta = \sigma\Big(\mathbf{W} \cdot \text{ReLU}\big(\mathbf{H} \cdot \text{concat}(\mathbf{e}_s, \mathbf{e}_o) + \mathbf{b}_1\big) + \mathbf{b}_2\Big), \tag{4.1}$$

where $\text{concat}(s, o) \in \mathbb{R}^{2d}$, $\mathbf{H} \in \mathbb{R}^{k \times 2d}$, $\mathbf{W} \in \mathbb{R}^{|\mathcal{R}| \times k}$, $\mathbf{b}_1 \in \mathbb{R}^k$, and $\mathbf{b}_2 \in \mathbb{R}^{|\mathcal{R}|}$, $\sigma(\mathbf{z}) = \frac{1}{1+\exp(-\mathbf{z})}$, $\text{ReLU}(\mathbf{z}) = \max(\mathbf{z}, \mathbf{0})$. Hence, $\Theta = \{\mathbf{W}, \mathbf{b}_2, \mathbf{H}, \mathbf{b}_1, \mathbf{E}\}$. Given (s, o), $\text{concat}(s, o)$ returns concatenated embeddings of (s, o). Thereafter, we perform two affine transfor-

mations with the ReLU and the sigmoid function to obtain predicted probabilities for relation ($\hat{\mathbf{y}} \in [0,1]^{|\mathcal{R}|}$). Finally, the incurred loss is computed by the binary cross-entropy function:

$$\ell(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{|\mathcal{R}|} \sum_{i}^{|\mathcal{R}|} \mathbf{y}_i \cdot \log(\hat{\mathbf{y}}_i) + (1 - \mathbf{y}_i) \cdot \log(1 - \hat{\mathbf{y}}_i) \tag{4.2}$$

where $\hat{\mathbf{y}}$ is the vector of predicted probabilities and $\mathbf{y}$ is a binary vector of indicating multi labels. Note that Equation (4.2) computes an incurred loss on a single training data point. From the perspective of MLE, our goal is to learn such Θ so that the likelihood of observing i.i.d sampled training data points $\mathcal{D} := \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n}$ is maximized,

$$\Theta = \arg\max_{\Theta} \prod_{i=1}^{n} P(\mathbf{y}_i \mid \mathbf{x}_i; \Theta) \qquad \text{Max.like. of iid } \mathcal{D} \tag{4.3}$$

$$\Theta = \arg\max_{\Theta} \sum_{i=1}^{n} \log\Big(\P(\mathbf{y}_i \mid \mathbf{x}_i; \Theta)\Big) \qquad \text{Use Log} \tag{4.4}$$

$$\Theta = \arg\max_{\Theta} \sum_{i=1}^{n} \log\Big(\hat{\mathbf{y}}_i^{\mathbf{y}_i}\ (\mathbf{1} - \hat{\mathbf{y}}_i)^{\mathbf{1}-\mathbf{y}_i}\Big) \qquad \text{Bernoulli Dist.} \tag{4.5}$$

$$\Theta = \arg\min_{\Theta} -\sum_{i=1}^{n} \mathbf{y}_i \log(\hat{\mathbf{y}}_i) + (\mathbf{1} - \mathbf{y}_i) \log(1 - \hat{\mathbf{y}}_i) \qquad \text{Convert to min.} \tag{4.6}$$

$$\Theta = \arg\min_{\Theta} -\sum_{i=1}^{n} \mathbf{y}_i \log(\hat{\mathbf{y}}_i) + (\mathbf{1} - \mathbf{y}_i) \log(\mathbf{1} - \hat{\mathbf{y}}_i) \qquad \text{BCE} \tag{4.7}$$

$$\Theta = \arg\min_{\Theta} \sum_{i=1}^{n} -\sum_{j=1}^{|\mathcal{R}|} \mathbf{y}_i^j \log(\hat{\mathbf{y}}_i^j) + (1 - \mathbf{y}_i^j) \log(1 - \hat{\mathbf{y}}_i{}^j) \qquad \text{Objective.} \tag{4.8}$$

In Equation (4.5), the Bernoulli distribution is used to model binary labels elementwise. Figure 4.1 shows the architecture of Shallom. To obtain a composite representation of (s, o), we concatenate embeddings of entities as opposed to averaging them, since averaging embeddings loses the order of the input (as in the standard bag-of-words representation (Le and Mikolov, 2014)).

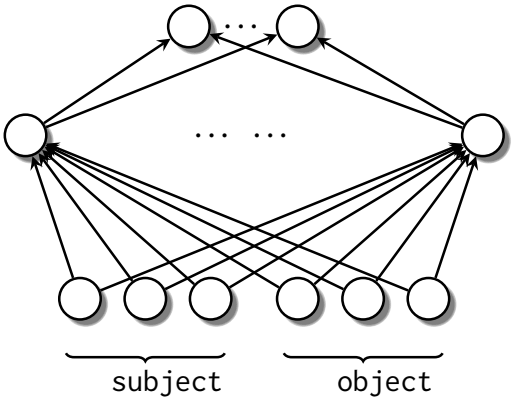


Figure 4.1: A visualization of Shallom.

4.2 Experiments & Results

4.2.1 Experiments

Experimental Setup

We compared Shallom against many state-of-the-art approaches and `Uniform Random Classifier` (URC) in the relation prediction task on benchmark datasets. All experiments were carried out on a single core of a server running Ubuntu 18.04 with 126 GB RAM with 16 Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz processors. To alleviate the hardware requirements for the reproducibility of our results and to foster further reproducible research, we provide hyperparameter optimization, training and evaluation scripts along with pre-trained models at https://github.com/dice-group/Shallom and https://github.com/dice-group/dice-embeddings.

Evaluation

We applied the Hits@N metric to evaluate the prediction performances. To evaluate runtime performances, we measured the elapsed runtime during the training phase. Ergo, we ignored the elapsed time during the data preprocessing since the training setup for Shallom is done on the fly while some approaches, including RDF2Vec (Ristoski and Paulheim, 2016), require additional computations such as applying the random walk technique. All approaches were trained four times on datasets. The reported runtimes (RT) of approaches are in seconds and the mean of the last three runs.

Datasets

We used five of the most commonly used benchmark datasets (WN18, WN18RR, FB15K, FB15K-237 and YAGO3-10) (Dettmers et al., 2018). An overview of the datasets is provided in Table 4.2.

Table 4.2: An overview of datasets in terms of number of entities, number of relations, and node degrees in the train split along with the number of triples in each split of the dataset.

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	Degr. (M±SD)	$\lvert\mathcal{G}^{\text{Train}}\rvert$	$\lvert\mathcal{G}^{\text{Validation}}\rvert$	$\lvert\mathcal{G}^{\text{Test}}\rvert$
YAGO3-10	123,182	37	9.6±8.7	1,079,040	5,000	5,000
FB15K	14,951	1,345	32.46±69.46	483,142	50,000	59,071
WN18	40,943	18	3.49±7.74	141,442	5,000	5,000
FB15K-237	14,541	237	19.7±30	272,115	17,535	20,466
WN18RR	40,943	11	2.2±3.6	86,835	3,034	3,134

Hyperparameter Optimization

We selected the hyperparameters of Shallom via grid search according to the Hits@1 on the validation set of each dataset. The hyperparameter ranges for the grid search were set as follows: embedding size $d = [30, 50, 100, 200]$, epochs = $[30, 50, 100]$, the width of the hidden layer $k = [.5d, d, 3d]$, batch size = $[256, 1000]$, dropout rate = $[.0, .2, .5]$ and L_2-normalizer = $[.0, .1]$. Initially, we used the default hyperparameters for all competing approaches provided in Trouillon et al. (2017). However, RESCAL, ComplEx, CP and DistMult did not terminate within three hours of computation. The long runtimes are corroborated by Trouillon et al. (2017). We hence optimized the hyperparameters of RESCAL, CP, TransE, DistMult and ComplEx via a grid search according to the Hits@1 on the validation set of each dataset. The hyperparameter ranges for the grid search were as follows: epochs = $[100, 200]$, negative ratio per valid triple = $[1, 5, 10, 50]$, and batch size = $[256, 512, |\mathcal{G}^{\text{Train}}|/100]$. We omitted d, regularization term and learning rate from grid-search and used the parameter settings provided in Trouillon et al. (2017). We selected the hyperparameters of RDF2Vec via grid search according to the Hits@1 on the validation set of each dataset. The hyperparameter ranges of RDF2Vec for the grid search were set as follows: embedding size $d = [50, 100]$, epochs = 100, number of negatives for Word2Vec = $[25, 100]$ and

random walk depth = [3, 5, 7]. After the embedding vectors are generated, we train the same scoring function defined in Equation (4.1) (look-up operation performed on RDF2Vec embeddings), by following the same optimization schema as our approach.

4.2.2 Results

Table 4.3, Table 4.4 and Table 4.5 report the HitsN relation prediction results on the five benchmark datasets. Overall, Shallom outperforms many state-of-the-art approaches while maintaining a superior runtime performance. The slightly superior (.018 absolute) performance of ProjE on the FB15K comes with the cost of more than 3 hours of computation. Shallom is significantly more time-efficient; it requires only 8 minutes, on average, a commodity computer. Since we could not reproduce the reported relation prediction results (Shi and Weninger, 2017), we could neither re-evaluate ProjE on FB15K nor include it on the other benchmark datasets. Approaches perform significantly better on WN18 than on FB15K. This may stem from the fact that WN18 contains (1) significantly fewer relations and (2) entity pairs having multiple relations than FB15K. More specifically, FB15K and WN18 datasets contain 63.856 and 277 number entity pairs, respectively, that occurred with multiple relations in the training splits.

Table 4.3: Hits@1 relation prediction results on FB15K and WN18. Results are taken from corresponding papers.

Method	FB15K	WN18
TransE (Shi and Weninger, 2017)	0.651	0.736
TransR (Xie et al., 2016a)	0.702	0.713
ProjE-listwise (Shi and Weninger, 2017)	**0.758**	-
PTransE (ADD, len-2 path) (Shi and Weninger, 2017)	0.695	-
DKLR(CNN) (Xie et al., 2016a)	0.698	-
TKRL (RHE) (Xie et al., 2016b)	0.711	-
RDFDNN (Onuki et al., 2017)	0.691	0.770
KGML (Onuki et al., 2019)	0.725	**0.975**
SSP (Xiao et al., 2017)	0.709	-
Shallom	0.734	0.970

Table 4.4 shows that Shallom outperforms all state-of-the-art approaches on the WN18RR and FB15K-237 datasets while maintaining an overall superior runtime performance. Note that the RT solely denotes the elapse training runtime. Initially, we trained RESCAL, TransE, ComplEx, CP and DistMult with hyperparameters provided in Trouillon et al. (2016). However, models other than TransE did not terminate within 3 hours of computation. Consequently, we selected the hyperparameters of approaches via grid search as explained in Section 4.2.1. TransE and DistMult yield a surprisingly better performance on WN18RR and FB15K-237 than on WN18 and FB15K. This may stem from (1) the hyperparameter optimization and (2) the fact that fewer numbers of entity pairs have multiple relations on training and testing datasets. The hyperparameters of TransE were not optimized in Shi and Weninger (2017); Onuki et al. (2017, 2019) where the Hit@1 performances of TransE were taken. CP performed poorly on the WN18RR due to the small number of relations as observed in Trouillon et al. (2017). During the training phase, the batch size was set to 32 in KGML and RDFDNN (Onuki et al., 2019, 2017). Although training models with a small-batch regime seemed to alleviate a possible degradation in the generalization performances of models Keskar et al. (2016), it came with the cost of increased runtime. By virtue of being a shallow NN, the error propagation was computationally more efficient in Shallom than KGML. Importantly, KGML and RDFDNN do not optimize the width of the hidden layers. Conversely, we optimized the width of Shallom, as per the suggestion in Nguyen et al. (2018b)—that optimizing the width of the network has an impact on the generalization performance. RDFDNN erroneously assumes one-to-one mapping between entity pairs to relations and possibly suffers from the hyperbolic tangent saturation as the hyperbolic tangent is applied in the hidden layer (Krizhevsky et al., 2012). RDF2Vec outperforms RESCAL, TransE, CP and DistMult w.r.t. Hits@3 and Hits@5 on WN18RR.

To confirm the performance of Shallom, we compared it with some of the best approaches in terms of runtime requirement and Hits@1 on a large benchmark dataset. Table 4.5 shows that Shallom reaches close to 1.0 Hits@5 and requires less than 10 minutes on the YAGO3-10. We could not evaluate KGML on YAGO3-10 due to its high memory consumption requiring more than 16 GB RAM.

Table 4.4: Average Hits@N relation prediction and runtime results on WN18RR and FB15K-237.

	WN18RR				**FB15K-237**			
		Hits				Hits		
	RT	@1	@3	@5	RT	@1	@3	@5
RESCAL	1860±6	0.331	0.529	0.734	5160±4	0.115	0.327	0.456
TransE	960±11	0.507	0.761	0.864	540±10	0.774	0.899	0.918
ComplEx	2160±15	0.515	0.652	0.758	5880±30	0.153	0.300	0.378
CP	840±15	0.332	0.518	0.659	8040±39	0.467	0.609	0.675
DistMult	780±13	0.497	0.677	0.799	1140±8	0.092	0.176	0.428
KGML	840±15	0.868	0.954	0.975	1080±10	0.921	0.960	0.976
RDFDNN	540±8	0.819	0.967	0.985	720±10	0.913	0.934	0.953
$\text{RDF2Vec}_{\text{Skip-Gram}}$	310±5	0.534	0.815	0.940	482±6	0.518	0.600	0.677
$\text{RDF2Vec}_{\text{CBOW}}$	337±10	0.451	0.785	0.932	472±8	0.522	0.608	0.687
URC		0.095	0.265	0.446		0.003	0.013	0.020
Shallom	610±13	**0.874**	**0.982**	**0.995**	404±8	**0.948**	**0.993**	**0.997**

Table 4.5: Average Hits@N relation prediction and runtime results on YAGO3-10.

	YAGO3-10			
		Hits		
	RT	@1	@3	@5
$\text{RDF2Vec}_{\text{Skip-Gram}}$	593±11	0.487	0.796	0.875
$\text{RDF2Vec}_{\text{CBOW}}$	625±12	0.491	0.803	0.873
Shallom	562±19	**0.630**	**0.983**	**0.996**

4.2.3 Discussion

The superior performance of Shallom stems from: (1) it being a shallow neural model, (2) optimizing the width of the hidden layer, (3) the task and evaluation measures used. By virtue of being a shallow NN, Shallom requires only 562 seconds to train on $|\mathcal{G}| > 10^6$ on a commodity computer. NNs are required to be wide enough (larger than the input dimension) to learn disconnected decision regions Nguyen et al. (2018b). Lastly, given the example (`Obama`, `Hawaii`), Shallom assigns high scores for `BirthPlace` and low scores for `SpouseOf`. This stems from the fact that input $\mathcal{G}$ does not involve triples such as (`SpouseOf`, `Hawaii`), while it involves many triples (`BirthPlace`, `Hawaii`). Shallom assigns presumably a high score (`Obama`, `BirthPlace`, `Paderborn`) although such a triple is not contained in $\mathcal{G}$. Since the test splits of the benchmark datasets do not involve such false triples, the Hit@N metric quantifies merely the performances of the relation prediction approaches on the valid triples. Ergo, the idea of corrupted triples is not necessary for relation prediction as each entity pair found in the test split is linked with a relation.

Trade-offs & Possible Improvements: Most KGE models can *directly* predict a missing head entity, relation or tail entity (see Section 2.4), whereas Shallom can only *directly* predict a missing relation. This stems from the fact that Shallom is defined as a sequence of non-linear mappings from a pair of entities to relations.

The relation prediction performance of Shallom may be further improved via the batch normalization technique (Ioffe and Szegedy, 2015). Referring to the running analogy between Shallom and Word2vec, increasing the window size of Shallom may further improve the performance. More specifically, a sequence of entities and relations can be used as input (e.g., $\mathbf{x} := (\texttt{entity22}, \texttt{relation1}, \texttt{entity2}, ?, \texttt{entity4})$), instead of using two entities as an input $\mathbf{x} := (\texttt{entity2}, ?, \texttt{entity4})$. By this, Shallom may incorporate multi-hop information in the relation prediction problem and learn entity embeddings tailored towards relation prediction.

5

Convolutional Complex Knowledge Graph Embeddings

Preamble: This chapter is based on Demir and Ngonga Ngomo (2021a).

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Axel-Cyrille Ngonga Ngomo. The code is available at https://github.com/dice-group/Convolutional-Complex-Knowledge-Graph-Embeddings and https://github.com/dice-group/dice-embeddings.

Research Questions: In this work, we are concerned with the following two research questions:

1. **RQ1.** Can we design a knowledge graph embedding model to predict missing triples on large knowledge graphs?
2. **RQ2.** Can we answer RQ1. while ensuring the parameter efficiency of our models?

5.1 Methodology

In this chapter, we introduce ConEx. Here, our goal is to design a knowledge graph embedding (KGE) model that effectively combines two state-of-the-art KGE models (ComplEx (Trouillon et al., 2016) and ConvE (Dettmers et al., 2018)) to advance the state of the art in link prediction. To this end, we design ConEx—a multiplicative composition of a 2D convolution operation with a Hermitian inner product on complex-valued embeddings. Previously, Sun et al. (2019) suggest that ComplEx is not able to model triples with transitive relations since ComplEx does not perform well on datasets containing many transitive relations (see Table 5 and Section 4.6 in Sun et al. (2019)). Motivated by this consideration, we propose ConEx, which applies the Hadamard product to compose a 2D convolution followed by an affine transformation with a Hermitian inner product in $\mathbb{C}$. By virtue of the proposed architecture (see Equation (5.1)), ConEx is endowed with the capability of

1. leveraging a 2D convolution operation and
2. degenerating to ComplEx if such degeneration is necessary to further minimize the incurred training loss.

ConEx benefits from the *parameter sharing* and *equivariant representation* properties of convolutions (Goodfellow et al., 2016). The parameter sharing property of the convolution operation allows ConEx to achieve parameter efficiency, while the equivariant representation allows ConEx to effectively integrate interactions captured in the stacked complex-valued embeddings of entities and relations into computation of scores. This implies that small interactions in the embeddings have small impacts on the predicted scores[1]. The rationale behind this architecture is to increase the expressiveness of our model without increasing the number of its parameters. As previously stated in (Trouillon et al., 2016), this nontrivial endeavor is the keystone of embedding models. Ergo, we aim to overcome the shortcomings of ComplEx in modeling triples containing transitive relations through combining it with a 2D convolutions followed by an affine transformation on $\mathbb{C}$. In Table 5.1, we summarize the symbols used in this chapter.

[1]We refer to Goodfellow et al. (2016) for further details of properties of convolutions.

Table 5.1: An overview of our notation used in Chapter 5.

Notation	Description
$\mathcal{G}$	A knowledge graph
$\mathcal{E}, \mathcal{R}$	Set of unique entities and relations in $\mathcal{G}$, respectively
Θ	Model parameters
$\mathbf{E}$	A complex-valued entity embedding matrix
$\mathbf{R}$	A complex-valued relation embedding matrix
$\mathrm{Re}(\cdot), \mathrm{Im}(\cdot)$	Real and imaginary parts of a complex number, respectively
$\sigma(\cdot), \mathrm{ReLU}(\cdot)$	Logistic sigmoid and rectified linear unit functions, respectively
$\mathrm{conv}(\cdot, \cdot)$	The convolution connection function in ConEx
$\mathbf{W}, \mathbf{b}$	Affine transformation used in $\mathrm{conv}(\cdot, \cdot)$
d	the size of embedding vectors for entities and relations
$\mathrm{vec}(\cdot)$	Flattening operation
$*$	Convolution operation
ω	Filters/kernels for the convolution operation
$\ell(\cdot, \cdot)$	Binary cross-entropy loss function

Approach: Given a triple $(\mathsf{h}, \mathsf{r}, \mathsf{t})$, ConEx : $\mathcal{E} \times \mathcal{R} \times \mathcal{E} \mapsto \mathbb{R}$ computes a triple score as

$$\text{ConEx}(\mathsf{h}, \mathsf{r}, \mathsf{t})_\Theta = \mathrm{Re}(\langle \mathrm{conv}(\mathbf{e}_h, \mathbf{e}_r), \mathbf{e}_h, \mathbf{e}_r, \overline{\mathbf{e}_t} \rangle), \tag{5.1}$$

where $\mathrm{conv}(\cdot, \cdot) : \mathbb{C}^{2d} \mapsto \mathbb{C}^d$ is defined as

$$\mathrm{conv}(\mathbf{e}_h, \mathbf{e}_r) = \mathrm{ReLU}\big(\mathrm{vec}(\mathrm{ReLU}([\mathbf{e}_h, \mathbf{e}_r] * \omega)) \cdot \mathbf{W} + \mathbf{b} \big), \tag{5.2}$$

where $\mathrm{ReLU}(\cdot)$ denotes the rectified linear unit function (ReLU), $\mathrm{vec}(\cdot)$ stands for a flattening operation, $*$ is the convolution operation, ω stands for kernels/filters in the convolution, and $(\mathbf{W}, \mathbf{b})$ characterize an affine transformation. $\Theta = \{\mathbf{E}, \mathbf{R}, \mathbf{W}, \mathbf{b}, \omega\}$ denotes the model parameters, where $\mathbf{E} \in \mathbb{C}^d$, $\mathbf{R} \in \mathbb{C}^d$, $\mathbf{W} \in \mathbb{C}^{|\omega| \times d}$, and $\mathbf{b} \in \mathbb{C}^d$. ConEx is enriched with the capability of controlling the impact of a 2D convolution and Hermitian inner product on the predicted scores.

The gradients of loss w.r.t. embeddings can be propagated in two ways, namely, via $\text{conv}(\mathbf{e}_h, \mathbf{e}_r)$ or $\text{Re}(\langle \mathbf{e}_h, \mathbf{e}_r, \overline{\mathbf{e}_t} \rangle)$. Equation (5.1) can be equivalently expressed by expanding its real and imaginary parts:

$$\begin{aligned} \text{ConEx}(h, r, t) &= \text{Re}\Big(\sum_{k=1}^{d} (\gamma)_k (\mathbf{e}_h)_k (\mathbf{e}_r)_k (\overline{\mathbf{e}_t})_k \Big) && (5.3) \\ &= \langle \text{Re}(\gamma), \text{Re}(\mathbf{e}_h), \text{Re}(\mathbf{e}_r), \text{Re}(\mathbf{e}_t) \rangle \\ &\quad + \langle \text{Re}(\gamma), \text{Re}(\mathbf{e}_h), \text{Im}(\mathbf{e}_r), \text{Im}(\mathbf{e}_t) \rangle \\ &\quad + \langle \text{Im}(\gamma), \text{Im}(\mathbf{e}_h), \text{Re}(\mathbf{e}_r), \text{Im}(\mathbf{e}_t) \rangle \\ &\quad - \langle \text{Im}(\gamma), \text{Im}(\mathbf{e}_h), \text{Im}(\mathbf{e}_r), \text{Re}(\mathbf{e}_t) \rangle && (5.4) \end{aligned}$$

where $\overline{\mathbf{e}_t}$ is the conjugate of $\mathbf{e}_t$ and γ denotes the output of $\text{conv}(\mathbf{e}_h, \mathbf{e}_r)$ for brevity. Such multiplicative inclusion of $\text{conv}(\cdot, \cdot)$ equips ConEx with two more degrees of freedom due the $\text{Re}(\gamma)$ and $\text{Im}(\gamma)$ parts.

Connection to ComplEx: During the optimization, $\text{conv}(\cdot, \cdot)$ is allowed to reduce its range into $\gamma \in \mathbb{C}$ such that $\text{Re}(\gamma) = 1 \wedge \text{Im}(\gamma) = 1$. This allows ConEx to degenerate into ComplEx as shown in Section 5.1. This multiplicative inclusion of $\text{conv}(\cdot, \cdot)$ is motivated by the scaling parameter in the batch normalization (see section 3 in Ioffe and Szegedy (2015)). Consequently, ConEx is allowed use a 2D convolution followed by an affine transformation as a scaling factor in the computation of scores.

Additive Connections: To measure an possible impact of multiplicative connections instead of additive connections, we define an additive variant of ConEx as follows

$$\begin{aligned} \text{AConEx}(h, r, t) &= \langle a + \text{Re}(\mathbf{e}_h), \text{Re}(\mathbf{e}_r), \text{Re}(\mathbf{e}_t) \rangle \\ &\quad + \langle b + \text{Re}(\mathbf{e}_h), \text{Im}(\mathbf{e}_r), \text{Im}(\mathbf{e}_t) \rangle \\ &\quad + \langle c + \text{Im}(\mathbf{e}_h), \text{Re}(\mathbf{e}_r), \text{Im}(\mathbf{e}_t) \rangle \\ &\quad - \langle d + \text{Im}(\mathbf{e}_h), \text{Im}(\mathbf{e}_r), \text{Re}(\mathbf{e}_t) \rangle && (5.5) \end{aligned}$$

where $[a, b, c, d] := \text{conv}(\mathbf{e}_h, \mathbf{e}_r)$, hence $\mathbf{W} \in \mathbb{C}^{|\omega| \times 2d}$ and $\mathbf{b} \in \mathbb{R}^{2d}$.

Training: We train our approach by following a standard setting (Dettmers et al., 2018; Balažević et al., 2019c). Similarly, we applied the standard data augmentation technique, the KvsAll training procedure[2]. After the data augmentation technique for a given pair (h, r), we compute scores for all $\mathsf{x} \in \mathcal{E}$ with $\textsc{ConEx}(\mathsf{h}, \mathsf{r}, \mathsf{x})$. We then apply the logistic sigmoid function $\sigma(\textsc{ConEx}((\mathsf{h}, \mathsf{r}, \mathsf{t})))$ to obtain predicted probabilities of entities. ConEx is trained to minimize the binary cross-entropy loss function ℓ that determines the incurred loss on a given pair (h, r) as defined in the following:

$$\ell(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{|\mathcal{E}|} \sum_{i=1}^{|\mathcal{E}|} \mathbf{y}^{(i)} \log(\hat{\mathbf{y}}^{(i)}) + (1 - \mathbf{y}^{(i)}) \log(1 - \hat{\mathbf{y}}^{(i)}), \tag{5.6}$$

where $\hat{\mathbf{y}} \in \mathbb{R}^{|\mathcal{E}|}$ is the vector of predicted probabilities and $\mathbf{y} \in [0, 1]^{|\mathcal{E}|}$ is the binary label vector.

5.2 Experiments & Results

5.2.1 Experiments

Experimental Setup

We compared ConEx against many state-of-the-art approaches in the link prediction task on benchmark datasets All experiments were carried out on a single core of a server running Ubuntu 18.04 with 126 GB RAM with 16 Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz processors. Throughout our experiments, the seed for the pseudo-random generator was fixed to 1. We provide hyperparameter optimization, training and evaluation scripts along with pre-trained models at https://github.com/dice-group/Convolutional-Complex-Knowledge-Graph-Embeddings and https://github.com/dice-group/dice-embeddings.

Datasets

We used five of the most commonly used benchmark datasets (WN18, WN18RR, FB15K, FB15K-237 and YAGO3-10). An overview of the datasets is provided in Table 5.2. WN18 and WN18RR are subsets of Wordnet, which describes lexical and semantic

[2]Note that the KvsAll strategy is called 1-N scoring in Dettmers et al. (2018). Here, we follow the terminology of Ruffinelli et al. (2019).

hierarchies between concepts and involves symmetric and antisymmetric relation types, while FB15K, FB15K-237 are subsets of Freebase, which involves mainly symmetric, antisymmetric and composite relation types (Sun et al., 2019).

Table 5.2: An overview of datasets in terms of number of entities, number of relations, and node degrees in the train split along with the number of triples in each split of the dataset.

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	Degr. (M±SD)	$\lvert\mathcal{G}^{\text{Train}}\rvert$	$\lvert\mathcal{G}^{\text{Validation}}\rvert$	$\lvert\mathcal{G}^{\text{Test}}\rvert$
YAGO3-10	123,182	37	9.6±8.7	1,079,040	5,000	5,000
FB15K	14,951	1,345	32.46±69.46	483,142	50,000	59,071
WN18	40,943	18	3.49±7.74	141,442	5,000	5,000
FB15K-237	14,541	237	19.7±30	272,115	17,535	20,466
WN18RR	40,943	11	2.2±3.6	86,835	3,034	3,134

Evaluation Metrics

We used the filtered MRR and Hits@N to evaluate link prediction performances, as in previous works (Sun et al., 2019; Trouillon et al., 2016; Dettmers et al., 2018; Balažević et al., 2019c). The filtered MRR and Hits@N metrics are defined in Equation (2.59), Equation (2.60), respectively.

Hyperparameter Optimization

We selected the hyperparameters of ConEx based on the MRR score obtained on the validation set of WN18RR. Hence, we evaluated the link prediction performance of ConEx on FB15K-237, YAGO3-10, WN18 and FB15K by using the best hyperparameter configuration found on WN18RR. This decision stems from the fact that we aim to reduce the impact of extensive hyperparameter optimization on the reported results and the CO_2 emission caused through relying on the findings of previously works (Ruffinelli et al., 2019). Strubell et al. (2019) highlighted the substantial energy consumption of performing extensive hyperparameter optimization. Moreover, Ruffinelli et al. (2019) showed that model configurations can be found by exploring relatively few random samples from a large hyperparameter space. With these considerations, we determined the ranges of hyperparameters for the grid search algorithm optimizer based on their best hyperparameter setting for ConvE (see Table 8 in Ruffinelli et al. (2019)). Specifically,

the ranges of the hyperparameters were defined as follows: d: $\{100, 200\}$; dropout rate:$\{.3, .4\}$ for the input; dropout rate: $\{.4, .5\}$ for the feature map; label smoothing: $\{.1\}$ and the number of output channels in the convolution operation: $\{16, 32\}$; the batch size: $\{1024\}$; the learning rate: $\{.001\}$. After determining the best hyperparameters based on the MRR on the validation dataset; we retrained ConEx with these hyperparameters on the combination of train and valid sets as applied in (Joulin et al., 2017). Motivated by the experimental setups for ResNet and AlexNet (He et al., 2016; Krizhevsky et al., 2017), we were interested in quantifying the impact of ensemble learning on the link prediction performances. Ensemble learning refers to learning a weighted combination of learning algorithms. In our case, we generated ensembles of models by averaging the predictions of said models.[3] To this end, we re-evaluated state-of-the-art models, including TucKER, DistMult and ComplEx on the combination of train and validation sets of benchmark datasets. Therewith, we were also able to quantify the impact of training state-of-the-art models on the combination of train and validation sets. Moreover, we noticed that link prediction performances of DistMult and ComplEx, on the YAGO3-10 dataset were reported without employing new training strategies (KvsAll, the reciprocal data augmentation, the batch normalization, and the ADAM optimizer). Hence, we trained DistMult, ComplEx on YAGO3-10 with these strategies.

5.2.2 Results

Table 5.3, Table 5.4, and Table 5.5 report the link prediction performances of ConEx on five benchmark datasets. Overall, ConEx outperforms state-of-the-art models on four out of five datasets. In particular, ConEx outperforms ComplEx and ConvE on all five datasets. This supports our original hypothesis, i.e., that the composition of a 2D convolution with a Hermitian inner product improves the prediction of relations in complex spaces. We used the Wilcoxon signed-rank test to measure the statistical significance of our link prediction results. Moreover, we performed an ablation study (see Table 5.9) to obtain confidence intervals for prediction performances of ConEx. Bold and underlined entries denote best and second-best results in all tables. Table 5.3 reports that ConEx outperforms all state-of-the-art models on WN18 and FB15K, whereas such distinct superiority is not observed on WN18RR and FB15K-237. Table 5.4 shows that ConEx outperforms many state-of-the-art models, including RotatE, ConvE, HypER,

[3]Ergo, the weights for models were set to 1 (see Cection 16.6 in Murphy (2012) for more details.)

ComplEx, NKGE, in all metrics on WN18RR and FB15K-237. This is an important result for two reasons: (1) ConEx requires significantly fewer parameters to yield such superior results (e.g., ConEx only requires 26.63M parameters on WN18RR, while RotatE relies on 40.95M parameters), and (2) we did not tune the hyperparameters of ConEx on FB15K-237. Furthermore, the results reported in Table 5.4 corroborate the findings of Ruffinelli et al. (2019): training DistMult and ComplEx with KvsAll, the reciprocal data augmentation, the batch normalization, and the ADAM optimizer leads to a significant improvement, particularly on FB15K-237.

Table 5.3: Link prediction results on WN18 and FB15K. Results are obtained obtained from Balažević et al. (2019c); Zhang et al. (2019).

	WN18				**FB15K**			
	MRR	Hits@10	Hits@3	Hits@1	MRR	Hits@10	Hits@3	Hits@1
DistMult	0.822	0.936	0.914	0.728	0.654	0.824	0.733	0.546
ComplEx	0.941	0.947	0.936	0.936	0.692	0.840	0.759	0.599
ANALOGY	0.942	0.947	0.944	0.939	0.725	0.854	0.785	0.646
R-GCN	0.819	0.964	0.929	0.697	0.696	0.842	0.760	0.601
TorusE	0.947	0.954	0.950	0.943	0.733	0.832	0.771	0.674
ConvE	0.943	0.956	0.946	0.935	0.657	0.831	0.723	0.558
HypER	0.951	0.958	0.955	0.947	0.790	0.885	0.829	0.734
SimplE	0.942	0.947	0.944	0.939	0.727	0.838	0.773	0.660
TuckER	0.953	0.958	0.955	0.949	0.795	0.892	0.833	0.741
QuatE	0.950	0.962	0.954	0.944	0.833	0.900	0.859	0.800
ConEx	**0.976**	**0.980**	**0.978**	**0.973**	**0.872**	**0.930**	**0.896**	**0.837**

During our experiments, we observed that many state-of-the-art models are not evaluated on YAGO3-10. This may stem from the fact that the size of YAGO3-10 prohibits performing extensive hyperparameter optimization even with the current state-of-the-art hardware systems. Note that YAGO3-10 involves 8.23 and 8.47 times more entities than FB15K and FB15K-237, respectively. Table 5.5 indicates that DistMult and ComplEx perform particularly well on YAGO3-10, provided that KvsAll, the reciprocal data augmentation, the batch normalization, and the ADAM optimizer are employed. Our results support findings of Ruffinelli et al. (2019). During training, we observed that the training loss of DistMult and ComplEx seemed to converge within 400 epochs, whereas

Table 5.4: Link prediction results on WN18RR and FB15K-237. Results are obtained from corresponding papers. ‡ denotes the recently reported results of corresponding models.

	WN18RR				FB15K-237			
	MRR	Hits@10	Hits@3	Hits@1	MRR	Hits@10	Hits@3	Hits@1
DistMult (Dettmers et al., 2018)	0.430	0.490	0.440	0.390	0.241	0.419	0.263	0.155
ComplEx (Dettmers et al., 2018)	0.440	0.510	0.460	0.410	0.247	0.428	0.275	0.158
ConvE (Dettmers et al., 2018)	0.430	0.520	0.440	0.400	0.335	0.501	0.356	0.237
RESCAL† (Ruffinelli et al., 2019)	0.467	0.517	0.480	0.439	0.357	0.541	0.393	0.263
DistMult† (Ruffinelli et al., 2019)	0.452	0.530	0.466	0.413	0.343	0.531	0.378	0.250
ComplEx† (Ruffinelli et al., 2019)	0.475	0.547	0.490	0.438	0.348	0.536	0.384	0.253
ConvE† (Ruffinelli et al., 2019)	0.442	0.504	0.451	0.411	0.339	0.521	0.369	0.248
HypER (Balažević et al., 2019b)	0.465	0.522	0.477	0.436	0.341	0.520	0.376	0.252
NKGE (Zhang et al., 2019)	0.450	0.526	0.465	0.421	0.330	0.510	0.365	0.241
RotatE (Sun et al., 2019)	0.476	0.571	0.492	0.428	0.338	0.533	0.375	0.241
TuckER (Balažević et al., 2019c)	0.470	0.526	0.482	0.443	0.358	0.544	0.394	0.266
QuatE (Zhang et al., 2019)	**0.482**	**0.572**	**0.499**	0.436	**0.366**	**0.556**	0.401	**0.271**
DistMult	0.439	0.527	0.455	0.399	0.353	0.539	0.390	0.260
ComplEx	0.453	0.546	0.473	0.408	0.332	0.509	0.366	0.244
TuckER	0.466	0.515	0.476	0.441	0.363	0.553	0.400	0.268
ConEx	0.481	0.550	0.493	**0.448**	**0.366**	0.555	**0.403**	**0.271**

the training loss of TuckER seemed to continue decreasing. Ergo, we conjecture that TuckER is more likely to benefit from increasing the number of epochs than DistMult and ComplEx. Table 5.5 shows that the superior performance of ConEx against state-of-the-art models including DistMult, ComplEx, HypER can be maintained on the largest benchmark dataset for the link prediction.

Delving into the link prediction results, we observed an inconsistency in the test splits of WN18RR and FB15K-237. Specifically, the test splits of WN18RR and FB15K-237 contain many out-of-vocabulary entities.[4] For instance, 6% of the test set on WN18RR involves out-of-vocabulary entities. During our experiments, we did not remove such triples to obtain fair comparisons on both datasets. To quantify the impact of unseen entities on link prediction performances, we conducted an additional experiment. We reported a detail analysis on the impact of unseen entities in Demir and Ngonga Ngomo (2021c).

[4] github.com/TimDettmers/ConvE/issues/66

Table 5.5: Link prediction results on YAGO3-10.

	YAGO3-10			
	MRR	Hits@10	Hits@3	Hits@1
DistMult (Dettmers et al., 2018)	0.340	0.540	0.380	0.240
ComplEx (Dettmers et al., 2018)	0.360	0.550	0.400	0.260
ConvE (Dettmers et al., 2018)	0.440	0.620	0.490	0.350
HypER (Balažević et al., 2019b)	0.533	0.678	0.580	0.455
RotatE (Sun et al., 2019)	0.495	0.670	0.550	0.402
DistMult	0.543	0.683	0.590	0.466
ComplEx	0.547	0.690	0.594	0.468
TuckER	0.427	0.609	0.476	0.331
ConEx	**0.553**	**0.696**	**0.601**	**0.474**

Link Prediction per Relation

Table 5.6 reports the link prediction per relation performances on WN18RR. Overall, models perform particularly well on triples containing symmetric relations such as `also_see` and `similar_to`. ConEx performs better on triples containing transitive relations (e.g., `hypernym` and `has_part`) compared to RotatE, DistMult, ComplEx and TuckER. Allen et al. (2021) ranked the complexity of type of relations as R > S > C in the link prediction task. Based on this ranking, superior performance of ConEx becomes more apparent as the complexity of relations increases.

Ensemble Learning

Table 5.7 reports the link prediction performances of ensembles based on pairs of models. These results suggest that ensemble learning can be applied as an effective means to boost the generalization performance of existing approaches including ConEx. These results may also indicate that models may be further improved through optimizing the impact of each model on the ensembles, e.g., by learning two scalars α and β in $\big(\alpha\text{ConEx}(\mathsf{s},\mathsf{p},\mathsf{o}) + \beta\text{TuckER}(\mathsf{s},\mathsf{p},\mathsf{o})\big)$ instead of averaging predicted scores.

Table 5.6: MRR link prediction on each relation of WN18RR. Results of RotatE are taken from Zhang et al. (2019). The complexity of type of relations in the link prediction task is defined as R > S > C Allen et al. (2021).

Relation Name	Type	RotatE	DistMult	ComplEx	TuckER	ConEx
hypernym	S	0.148	0.102	0.106	0.121	**0.149**
instance_hypernym	S	0.318	0.218	0.292	0.375	**0.393**
member_meronym	C	**0.232**	0.129	0.181	0.181	0.171
synset_domain_topic_of	C	0.341	0.226	0.266	0.344	**0.373**
has_part	C	0.184	0.143	0.181	0.171	**0.192**
member_of_domain_usage	C	**0.318**	0.225	0.280	0.213	**0.318**
member_of_domain_region	C	0.200	0.095	0.267	0.284	**0.354**
derivationally_related_form	R	0.947	0.982	0.984	0.985	**0.986**
also_see	R	0.585	0.639	0.557	**0.658**	0.647
verb_group	R	0.943	**1.000**	**1.000**	**1.00**	**1.000**
similar_to	R	**1.000**	**1.000**	**1.000**	**1.00**	**1.000**

Parameter Analysis

Table 5.8 indicates the robustness of ConEx against the overfitting problem. Increasing the number of parameters in ConEx does not lead to a significant decrease in the generalization performance. In particular, ConEx achieves similar generalization performance, with $p = 26.63M$ and $p = 70.66M$, as the difference between MRR scores are less than absolute 1%. This cannot be explained with convolutions playing no role as ConEx would then degrade back to ComplEx and achieve the same results (which is clearly not the case in our experiments).

Statistical Hypothesis Testing

We performed the Wilcoxon signed-rank test to check whether our results are significant. Our null hypothesis was that the link prediction performances of ConEx, ComplEx and ConvE come from the same distribution. The alternative hypothesis was correspondingly that these results come from different distributions. To perform the Wilcoxon signed-rank test (two-sided), we used the differences of the MRR, Hits@1, Hits@3, and Hits@10 performances on WN18RR, FB15K-237 and YAGO3-10.

Table 5.7: Link prediction results of ensembled models on WN18RR and FB15K-237. Second rows denote link prediction results without triples containing out-of-vocabulary entities. ConEx-ConEx stands for ensembling two ConEx trained with the dropout rate 0.4 and 0.5 on the feature map.

	WN18RR				FB15K-237			
	MRR	Hits@10	Hits@3	Hits@1	MRR	Hits@10	Hits@3	Hits@1
DistMult-ComplEx	0.446	0.545	0.467	0.398	0.359	0.546	0.397	0.265
	0.475	0.579	0.497	0.426	0.359	0.546	0.397	0.265
DistMult-TuckER	0.446	0.533	0.461	0.405	0.371	0.563	0.410	0.275
	0.476	0.569	0.492	0.433	0.371	0.563	0.411	0.275
ConEx-DistMult	0.454	0.545	0.471	0.410	0.371	0.563	0.409	0.275
	0.484	0.580	0.501	0.439	0.367	0.556	0.403	0.272
ConEx-ComplEx	0.470	0.554	0.487	0.428	0.370	0.559	0.407	0.276
	0.501	0.589	0.518	0.456	0.360	0.547	0.397	0.267
ConEx-TuckER	0.483	0.549	0.494	0.449	0.375	0.568	0.414	0.278
	0.514	0.583	**0.526**	**0.479**	0.375	0.568	0.414	0.278
ConEx-ConEx	0.485	0.559	0.495	0.450	**0.376**	0.569	**0.415**	**0.279**
	0.517	**0.594**	**0.526**	**0.479**	**0.376**	**0.570**	**0.415**	**0.279**

We performed two hypothesis tests between ConEx and ComplEx as well as between ConEx and ConvE. In both tests, we were able to reject the null hypothesis with a p-value $< 1\%$. Ergo, the superior performance of ConEx is statistically significant.

Ablation Study

We conducted our ablation study in a fashion akin to Dettmers et al. (2018). We evaluated 2 different parameter initializations to compute confidence intervals, that is defined as $\bar{x} \pm 1.96 \cdot \frac{s}{\sqrt{n}}$, where $\bar{x} = \frac{1}{n}\sum_i^n x_i$ and $s = \sqrt{\frac{\sum_i^n (x_i - \bar{x})^2}{n}}$, respectively. Hence, the mean and the standard deviation are computed without Bessel's correction. Our results suggest that the initialization of parameters does not play a significant role in the link performance of ConEx. The dropout technique is the most important component in the generalization performance of ConEx. This is also observed in Dettmers et al. (2018). Moreover, replacing the Adam optimizer with the RMSprop optimizer (Tieleman and Hinton, 2012) leads to slight increases in the variance of the link prediction results.

Table 5.8: Influence of different hyperparameter configurations for ConEx on WN18RR. d, c and p stand for the dimensions of embeddings in $\mathbb{C}$, number of output channels in 2D convolutions and number of free parameters in millions, respectively.

			WN18RR			
d	c	p	MRR	Hits@10	Hits@3	Hits@1
300	64	70.66M	0.475	0.540	0.490	0.442
250	64	52.49M	0.475	0.541	0.488	0.441
300	32	47.62M	0.480	0.548	0.491	0.447
250	32	36.39M	0.479	0.545	0.490	0.446
300	16	36.10M	0.479	0.550	0.494	0.445
250	16	28.48M	0.477	0.544	0.489	0.443
200	32	26.63M	0.481	0.550	0.493	0.447
100	32	10.75M	0.474	0.533	0.480	0.440
100	16	9.47M	0.476	0.536	0.486	0.441
50	32	4.74M	0.448	0.530	0.477	0.401

During our ablation experiments, we were also interested in decomposing ConEx through removing $\text{conv}(\cdot,\cdot)$, after ConEx is trained with it on benchmark datasets. By doing so, we aim to observe the impact of a 2D convolution in the computation of scores. Table 5.10 indicates that the impact of $\text{conv}(\cdot,\cdot)$ differs depending on the input knowledge graph. As the size of the input knowledge graph increases, the impact of $\text{conv}(\cdot,\cdot)$ on the computation of scores of triples increases.

5.2.3 Multiplicative vs. Additive Connections

We conducted further experiments to observe the impact of replacing the multiplicative connection of 2D convolution operation with an additive connection of 2D convolution operation. We selected the following configuration: Embeddings are represented with 32 real-valued numbers, the mini-batch size is set to 1024, Adam with 0.1 learning rate is used as optimizer and 32 kernels of shape 3 by 3 are used.

Tables 5.11 and 5.12 suggest that AConEx outperforms ConEx in all metrics with all configuration settings. We conjecture that using the last ReLU activation function in $\text{conv}(\mathbf{e}_h, \mathbf{e}_r)$ leads to such performance differences.

Table 5.9: Ablation study for ConEx on FB15K-237. *dp* and *ls* denote the dropout technique and the label smoothing technique, respectively.

	FB15K-237			
	MRR	Hits@10	Hits@3	Hits@1
Full	0.366±0.000	0.556±0.001	0.404±0.001	0.270±0.001
With RMSprop	0.361±0.004	0.550±0.007	0.400±0.005	0.267±0.003
No *dp* on inputs	0.282±0.000	0.441±0.001	0.313±0.001	0.203±0.000
No *dp* on feature map	0.351±0.000	0.533±0.000	0.388±0.001	0.259±0.001
No *ls*	0.321±0.001	0.498±0.001	0.354±0.001	0.232±0.002

Table 5.10: Link prediction results on benchmark datasets. ConEx$^-$ stands for removing $\text{conv}(\cdot,\cdot)$ in ConEx during the evaluation.

	ConEx				ConEx$^-$			
	MRR	Hits@10	Hits@3	Hits@1	MRR	Hits@10	Hits@3	Hits@1
WN18RR	0.481	0.550	0.493	0.448	0.401	0.494	0.437	0.346
FB15K-237	0.366	0.555	0.403	0.271	0.284	0.458	0.314	0.198
YAGO3-10	0.553	0.696	0.601	0.477	0.198	0.324	0.214	0.136

Each negative number in the output vector of the affine transformation in $\text{conv}(\mathbf{e}_h, \mathbf{e}_r)$ is mapped to 0 by the last ReLU activation function. Therefore, 0s in $\gamma := \text{conv}(\mathbf{e}_h, \mathbf{e}_r)$ more heavily influences the final score in ConEx than AConEx. In ConEx, the impact of $\text{conv}(\mathbf{e}_h, \mathbf{e}_r)$ in the final score is ignored iff $\text{conv}(\mathbf{e}_h, \mathbf{e}_r) =: \gamma \in \mathbf{1}$, whereas, in AConEx, the impact of $\text{conv}(\mathbf{e}_h, \mathbf{e}_r)$ is ignored $\text{conv}(\mathbf{e}_h, \mathbf{e}_r) =: \gamma \in \mathbf{0}$. Due to the last ReLU activation function, the latter is arguably more easily learned than the former.

5.2.4 Discussion

The superior performance of ConEx stems from the composition of a 2D convolution with a Hermitian inner product of complex-valued embeddings. Trouillon et al. (2016) show that a Hermitian inner product of complex-valued embeddings can be effectively used to tackle the link prediction problem. Applying the convolution operation on complex-valued embeddings of subjects and predicates permits ConEx to recognize interactions between subjects and predicates in the form of complex-valued feature maps.

Table 5.11: Link prediction results with multiplicative and additive connections. AConEx denotes ConEx with an additive connection of the 2D convolution operation.

	KINSHIP				UMLS			
	MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
ConEx-train	0.219	0.113	0.223	0.438	0.935	0.884	0.984	0.993
ConEx-val	0.188	0.084	0.183	0.415	0.830	0.725	0.924	0.974
ConEx-test	0.185	0.082	0.183	0.395	0.820	0.704	0.926	0.979
AConEx-train	**0.855**	**0.579**	**0.926**	**0.986**	**0.999**	**0.999**	**1.000**	**1.000**
AConEx-val	**0.711**	**0.579**	**0.808**	**0.951**	**0.733**	**0.597**	**0.847**	**0.956**
AConEx-test	**0.711**	**0.582**	**0.823**	**0.952**	**0.743**	**0.611**	**0.845**	**0.973**

Through the affine transformation of feature maps and their inclusion into a Hermitian inner product involving the conjugate-transpose of complex-valued embeddings of objects, ConEx can accurately infer various types of relations. Moreover, the number and shapes of the kernels permit to adjust the expressiveness, while ConEx retains the parameter efficiency due to the parameter sharing property of convolutions. By virtue of the design, the expressiveness of ConEx may be further improved by increasing the depth of the $\text{conv}(\cdot, \cdot)$ via the residual learning block (He et al., 2016).

Trade-offs & Possible Improvements: During our experiments, we observe that DistMult often requires less runtime than ComplEx and ComplEx requires less runtime than ConEx per epoch. This may stem from the fact that, a single score for a given triple requires less elementwise operations per embedding dimension in DistMult and ComplEx than in ConEx. Moreover, during our experiments, we observe that DistMult often requires fewer epoch than ComplEx to reach its peak performance, i.e., the trajectory of the loss function stabilizes. We conjecture that it may be beneficial to train models longer that based on more complex operations (e.g., an Hermitian inner product or a convolution operation). Using the last ReLU activation function in Equation (5.2) leads to ignoring the impact of negative complex numbers in computed scores for ConEx. Omitting the ReLU activation or replacing it with the Leaky ReLU function may improve the performance.

Table 5.12: Link prediction performance comparison ConEx and AConEx on FB15k-237 with different training epochs.

		ConEx				**AConEx**			
N		MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
100	train	0.164	0.111	0.178	0.266	**0.490**	**0.382**	**0.548**	**0.697**
	val	0.180	0.126	0.194	0.287	**0.264**	**0.187**	**0.288**	**0.417**
	test	0.178	0.123	0.194	0.283	**0.259**	**0.182**	**0.285**	**0.412**
300	train	0.189	0.129	0.204	0.306	**0.584**	**0.483**	**0.644**	**0.774**
	val	0.209	0.149	0.226	0.328	**0.271**	**0.192**	**0.294**	**0.428**
	test	0.208	0.146	0.228	0.328	**0.266**	**0.187**	**0.290**	**0.422**
500	train	0.203	0.138	0.218	0.326	**0.584**	**0.484**	**0.642**	**0.772**
	val	0.219	0.157	0.236	0.340	**0.265**	**0.189**	**0.288**	**0.420**
	test	0.215	0.152	0.233	0.339	**0.265**	**0.187**	**0.290**	**0.421**

6

Convolutional Hypercomplex Embeddings for Link Prediction

Preamble: This chapter is based on Demir et al. (2021a).

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Diego Moussallem Stefan Heindorf, and Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Stefan Heindorf, Diego Moussallem, and Axel-Cyrille Ngonga Ngomo. A brief summary of this publication is provided The code is available at https://github.com/dice-group/Convolutional-Hypercomplex-Embeddings-for-Link-Prediction and https://github.com/dice-group/dice-embeddings.

Research Questions: In this work, we are concerned with the following two research questions:

1. **RQ1.** Can we design a knowledge graph embedding model to predict missing triples on large knowledge graphs?
2. **RQ2.** Can we answer RQ1. while ensuring the parameter efficiency of our models?

6.1 Methodology

In this chapter, we introduce four hypercomplex KGE models QMult,OMult, ConvQ and ConvO. Here, our goal is to confirm whether our finding in using convolutions on complex numbers $\mathbb{C}$ (see Chapter 5) generalizes to to the two largest normed division algebras, i.e., hypercomplex numbers (Quaternions $\mathbb{H}$ and Octonions $\mathbb{O}$). To this end, we propose four KGE models—QMult,OMult, ConvQ and ConvO. The former two model learn quaternion- and octonion-valued embeddings based on element-wise multiplications followed by an inner product in Quaternions and Octonions. respectively. The latter two model extend the former two by including 2D convolution operation.

Motivation: Dettmers et al. (2018) suggest that indegree and PageRank can be used to quantify the difficulty of predicting missing links in KGs. Results indicate that the superiority of ConvE becomes more apparent against DistMult and ComplEx as the complexity of the knowledge graph increases, i.e., indegree and PageRank of a KG increase (see Table 6 in Dettmers et al. (2018)). In turn, Zhang et al. (2019) show that learning quaternion-valued embeddings via multiplicative interactions can be a more effective means of predicting missing links than learning real and complex-valued embeddings. Although learning quaternion-valued embeddings through multiplicative interactions yields promising results, the only way to further increase the expressiveness of such models is to increase the number of dimensions of embeddings. This does not scale to larger knowledge graphs (Dettmers et al., 2018). Increasing parameter efficiency while retaining effectiveness is a desired property in many applications (Zhang et al., 2021; Trouillon et al., 2016, 2017).

Motivated by findings of aforementioned works, we investigate the composition of convolution operations with hypercomplex multiplications. The rationale behind this composition is to increase the expressiveness without increasing the number of parameters. This nontrivial endeavor is the keystone of embedding models (Trouillon et al., 2016). The sparse connectivity property of the convolution operation endows models with parameter efficiency, which helps to scale to larger knowledge graphs. Additionally, different configurations of the number of kernels and their shapes can be explored to find the best ratio between expressiveness and the number of parameters. Although increasing the number of feature maps results in increasing the number of parameters, we are able to benefit from the parameter sharing property of convolu-

tions (Goodfellow et al., 2016). In Table 6.1, we summarize the symbols used in this chapter.

Table 6.1: An overview of our notation used in Chapter 6.

Notation	Description
$\mathcal{G}$	Knowledge graph
$\mathcal{E}, \mathcal{R}$	Set of unique entities and relations in $\mathcal{G}$, respectively
Θ	Model parameters
$\mathbf{E}$	A hypercomplex-valued entity embedding matrix
$\mathbf{R}$	A hypercomplex-valued relation embedding matrix
$\circ$	Hadamard product
$\otimes$	Hamilton/Quaternion product
$\bigstar$	Octonion product
$\sigma(\cdot), \text{ReLU}(\cdot)$	Logistic sigmoid and rectified linear unit functions, respectively
$\text{conv}(\cdot, \cdot)$	The convolution connection function
$\mathbf{W}, \mathbf{b}$	Affine transformation used in $\text{conv}(\cdot, \cdot)$
d	the size of embedding vectors for entities and relations
$\text{vec}(\cdot)$	Flattening operation
$*$	Convolution operation
ω	Filters/kernels for the convolution operation
$\ell(\cdot, \cdot)$	Binary cross-entropy loss function

Approaches: Given a triple $(\mathsf{h}, \mathsf{r}, \mathsf{t})$, $\text{QMult} : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \mapsto \mathbb{R}$ computes a triple score through the quaternion multiplication of head entity embeddings $\mathbf{e}_h$ and relation embeddings $\mathbf{e}_r$ followed by the inner product with tail entity embeddings $\mathbf{e}_t$ as

$$\text{QMult}(h, r, t)_\Theta = \mathbf{e}_h \otimes \mathbf{e}_r \cdot \mathbf{e}_t, \tag{6.1}$$

where $\Theta = \{\mathbf{E}, \mathbf{R}\}$ denotes the model parameters, where $\mathbf{E} \in \mathbb{H}^d$, and $\mathbf{R} \in \mathbb{H}^d$, hence, $\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{H}^d$.

Similarly, OMult : $\mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightarrow \mathbb{R}$ performs the octonion multiplication followed by the inner product as

$$\text{OMult}(h, r, t)_\Theta = \mathbf{e}_h \bigstar \mathbf{e}_r \cdot \mathbf{e}_t, \tag{6.2}$$

where $\Theta = \{\mathbf{E}, \mathbf{R}\}$ denotes the model parameters, where $\mathbf{E} \in \mathbb{O}^d$, and $\mathbf{R} \in \mathbb{O}^d$, hence, $\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{O}^d$. Computing scores of triples in this setting can be illustrated in two consecutive steps: (1) rotating $\mathbf{e}_h$ through $\mathbf{e}_r$ by applying quaternion/octonion multiplication, and (2) squishing $(\mathbf{e}_h \otimes \mathbf{e}_r)$ and $\mathbf{e}_t$ into a real number line by taking the inner product. During training, the degree between $(\mathbf{e}_h \otimes \mathbf{e}_r)$ and $\mathbf{e}_t$ is minimized provided $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$. Motivated by the response of John T. Graves to W. R. Hamilton,[1] we combine 2D convolutions with QMult and OMult as defined

$$\text{ConvQ}(h, r, t)_\Theta = \text{conv}(\mathbf{e}_h, \mathbf{e}_r) \circ (\mathbf{e}_h \otimes \mathbf{e}_r) \cdot \mathbf{e}_t, \tag{6.3}$$

$$\text{ConvO}(h, r, t)_\Theta = \text{conv}(\mathbf{e}_h, \mathbf{e}_r) \circ (\mathbf{e}_h \bigstar \mathbf{e}_r) \cdot \mathbf{e}_t, \tag{6.4}$$

where $\text{conv}(\cdot, \cdot) : \mathbb{H}^{2d} \rightarrow \mathbb{H}^d$ (respectively : $\mathbb{O}^{2d} \rightarrow \mathbb{O}^d$) is defined as

$$\text{conv}(\mathbf{e}_h, \mathbf{e}_r) = \text{ReLU}\big(\text{vec}(\text{ReLU}([\mathbf{e}_h, \mathbf{e}_r] * \omega)) \cdot \mathbf{W} + \mathbf{b}\big). \tag{6.5}$$

$\text{ReLU}(\cdot)$, $\text{vec}(\cdot)$, $*$, ω, $[\cdot, \cdot]$, and $(\mathbf{W}, \mathbf{b})$ denote the rectified linear unit function, a flattening operation, convolution operation, kernel in the convolution, stacking operation, and an affine transformation, respectively.

Connection to ComplEx and DistMult: During training, $\text{conv}(\cdot, \cdot)$ can reduce its range into $\gamma \in 1$ if such reduction is necessary to further decrease the training loss. In the following Equations (6.6) to (6.10), we elucidate the reduction of ConvQ into QMult and ComplEx:

$$\text{ConvQ}(\mathsf{h}, \mathsf{r}, \mathsf{t})_\Theta = \gamma \circ (\mathbf{e}_h \otimes \mathbf{e}_r) \cdot \mathbf{e}_t. \tag{6.6}$$

[1] "If with your alchemy you can make three pounds of gold, why should you stop there?" Baez (2002).

Equation (6.6) corresponds to QMult provided that $\text{conv}(\mathbf{e}_h, \mathbf{e}_r) = \gamma = 1$. ConvQ can be further reduced into ComplEx by setting the imaginary parts $\mathbf{j}$ and $\mathbf{k}$ of $\mathbf{e}_h$, $\mathbf{e}_r$ and $\mathbf{e}_t$ to zero:

$$\gamma \circ \big((a_h + b_h\mathbf{i}) \otimes (a_r + b_r\mathbf{i})\big) \cdot (a_t + b_t\mathbf{i}). \tag{6.7}$$

Computing the quaternion multiplication of two quaternion-valued vectors corresponds to Equation (6.8):

$$\gamma \circ [(a_h \circ a_r - b_h \circ b_r) + (a_h \circ b_r + b_h \circ a_r)\mathbf{i}] \cdot (a_t + b_t\mathbf{i}). \tag{6.8}$$

The resulting quaternion-valued vector is scaled with $\gamma = [\gamma_1, \gamma_2]$:

$$[(\gamma_1 \circ a_h \circ a_r - \gamma_1 \circ b_h \circ b_r) + (\gamma_2 \circ a_h \circ b_r + \gamma_2 \circ b_h \circ a_r)\mathbf{i}] \cdot (a_t + b_t\mathbf{i}). \tag{6.9}$$

Through taking the inner product of the former vector with $(a_t + b_t\mathbf{i})$, we obtain

$$\begin{aligned} \text{ConvQ}(\mathsf{h}, \mathsf{r}, \mathsf{t}) = &\langle \gamma_1, a_h, a_r, a_t \rangle \\ &+\langle \gamma_2, b_h, a_r, b_t \rangle \\ &+\langle \gamma_2, a_h, b_r, b_t \rangle \\ &-\langle \gamma_1, b_h, b_r, a_t \rangle, \end{aligned} \tag{6.10}$$

where $\langle a, b, c, d \rangle = \sum_k a_k b_k c_k d_k$ corresponds to the multi-linear inner product. Equation (6.10) corresponds to ComplEx provided that $\gamma = 1$. In the same way, ConvQ can be reduced into DistMult by setting all imaginary parts $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$ to zero for $\mathbf{e}_h$, $\mathbf{e}_r$, and $\mathbf{e}_t$ yielding

$$\text{ConvQ}(\mathsf{h}, \mathsf{r}, \mathsf{t}) = \langle \gamma_1, a_h, a_r, a_t \rangle. \tag{6.11}$$

Connection to residual learning: The residual learning framework facilitates the training of deep neural networks. A simple residual learning block consists of two weight layers denoted by $\mathcal{F}(x)$ and an identity mapping of the input x (see Figure 2 in He et al. (2016)). Increasing the depth of a neural model via stacking residual learning blocks led to significant improvements in many domains. In our setting, $\mathcal{F}(\cdot)$ and x correspond to $\text{conv}(\cdot, \cdot)$ and $[\mathbf{e}_h, \mathbf{e}_r]$, respectively. We replaced the identity mapping of the input with the hypercomplex multiplication. To scale the output, we replaced the element-wise vector addition with the Hadamard product.

By virtue of such inclusion, ConvQ and ConvO are endowed with the ability of controlling the impact of $\text{conv}(\cdot, \cdot)$ on predicted scores as shown in Equation (6.10). Ergo, the gradients of loss w.r.t. head entity and relation embeddings can be propagated in two ways, namely, via $\text{conv}(\mathbf{e}_h, \mathbf{e}_r)$ or hypercomplex multiplication. Moreover, the number of feature maps and the shape of kernels can be used to find the best ratio between expressiveness and the number of parameters. Hence, the expressiveness of models can be adjusted without necessarily increasing the embedding size. Although increasing the number of feature maps results in increasing the number of parameters in the model, we are able to benefit from the parameter sharing property of convolutions.

6.2 Experiments & Results

Experimental Setup

We compare our models against many state-of-the-art approaches in the link prediction task on benchmark datasets. All experiments are carried out on a single core of a server running Ubuntu 18.04 with 126 GB RAM with 16 Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz processors and a single NVIDIA GeForce RTX 3090. Throughout our experiments, the seed for the pseudo-random generator is fixed to 1. We provide hyperparameter optimization, training and evaluation scripts along with pre-trained models at https://github.com/dice-group/Convolutional-Hypercomplex-Embeddings-for-Link-Prediction and https://github.com/dice-group/dice-embeddings.

Datasets

We use seven benchmark datasets: WN18RR, FB15K-237, YAGO3-10, FB15K, WN18, UMLS and Kinship. An overview of the datasets is provided in Table 6.2. The latter four datasets are included for the sake of the completeness of our evaluation. Since the KvsAll training strategy is applied, the training, validation and test sets are augmented with reciprocal triples $((\mathtt{t}, \mathtt{r}^{-1}, \mathtt{h}))$ as done in previous works (Dettmers et al., 2018; Balažević et al., 2019b,c). For link prediction based on only tail entity ranking experiments (see Table 6.6), we omit the data augmentation on the test set, as in (Bansal et al., 2019).

Table 6.2: An overview of benchmark datasets in terms of entities, relations, average node degree plus/minus standard deviation.

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	Degree	$\lvert\mathcal{G}^{\text{Train}}\rvert$	$\lvert\mathcal{G}^{\text{Val.}}\rvert$	$\lvert\mathcal{G}^{\text{Test}}\rvert$
YAGO3-10	123,182	37	9.6±8.7	1,079,040	5,000	5,000
FB15K	14,951	1,345	32.5±69.5	483,142	50,000	59,071
WN18	40,943	18	3.5±7.7	141,442	5,000	5,000
FB15k-237	14,541	237	19.7±30	272,115	17,535	20,466
WN18RR	40,943	11	2.2±3.6	86,835	3,034	3,134
KINSHIP	104	25	82.2±3.5	8,544	1,068	1,074
UMLS	135	46	38.6±32.5	5,216	652	661

Evaluation Metrics

We use the standard metrics (*filtered* MRR and Hits@N) to evaluate link prediction performances, as in previous works (Trouillon et al., 2016; Dettmers et al., 2018; Sun et al., 2019; Balažević et al., 2019b,c). The filtered MRR and Hits@N metrics are defined in Equation (2.59), Equation (2.60), respectively. For the sake of completeness, we also report link prediction performances based on *only tail rankings*, i.e., without including triples with reciprocal relations into test data, as done in Bansal et al. (2019).

Hyperparameter Optimization

We apply the standard training strategy as Dettmers et al. (2018); Balažević et al. (2019b,c): Following the KvsAll training procedure,[2] for a given pair (h, r), we compute scores for all $x \in \mathcal{E}$ with $\phi(h, r, x)$ and apply the logistic sigmoid function $\sigma(\phi(h, r, x))$. Models are trained to minimize the binary cross entropy loss function, where $\hat{\mathbf{y}} \in \mathbb{R}^{|\mathcal{E}|}$ and $\mathbf{y} \in [0, 1]^{|\mathcal{E}|}$ denote the predicted scores and binary label vector, respectively.

$$\ell(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{|\mathcal{E}|} \sum_{i=1}^{|\mathcal{E}|} \mathbf{y}^{(i)} \log(\hat{\mathbf{y}}^{(i)}) + (1 - \mathbf{y}^{(i)}) \log(1 - \hat{\mathbf{y}}^{(i)}). \tag{6.12}$$

We employ the Adam optimizer (Kingma and Ba, 2014), dropout (Srivastava et al., 2014), label smoothing and batch normalization (Ioffe and Szegedy, 2015), as in the

[2]Here, we follow the terminology of Ruffinelli et al. (2019).

literature (Balažević et al., 2019b,c; Dettmers et al., 2018). Moreover, we selected hyperparameters of our approaches by random search based on validation set performances (Balažević et al., 2019c). Notably, we did not search a good random seed for the random number generator, and we fixed the seed to 1 throughout our experiments.

6.2.1 Results

Table 6.3 reports link prediction results on the WN18RR, FB15K-237, and YAGO3-10 datasets. Overall, the superior performance of our approaches becomes more and more apparent as the size and complexity of the knowledge graphs grow. On the smallest benchmark dataset (WN18RR), QMult, OMult, ConvQ and ConvO outperform many approaches including DistMult, ConvE and ComplEx in all metrics. However, QuatE, TuckER, and RotatE yield the best performance. On the second-largest benchmark dataset (FB15K-237 is **3.1**× larger than WN18RR), ConvO outperforms all state-of-the-art approaches in 3 out of 4 metrics. Additionally, QMult and ConvQ outperform all state-of-the-art approaches except for TucKER in terms of MRR, H@1 and H@3. On the largest benchmark dataset (YAGO3-10 is **12.4**× larger than WN18RR), QMult, ConvO, ConvQ outperform all approaches in all metrics. Surprisingly, QMult and OMult reach the best and second-best performance in all metrics, whereas ConvO does not perform particularly well compared to our other approaches. ConvO outperforms QMult, OMult, and ConvQ in 8 out of 12 metrics, whereas QMult yields better performance on YAGO3-10. Overall, these results suggest that superiority of learning hypercomplex embeddings becomes more apparent as the size and complexity of the input knowledge graph increases, as measured by indegree (see Table 6.2) and PageRank (see Table 6 in Dettmers et al. (2018)). In Table 6.4, we compare some of the best performing approaches on WN18RR, FB15K-237 and YAGO3-10 in terms of the number of trainable parameters. Results indicate that our approaches yield competitive (if not better) performance on all benchmark datasets.

Ensemble Learning

Table 6.5 reports link prediction results of ensembled models on benchmark datasets. Two models are ensembled by averaging their predictions at testing time. Averaging the predicted scores of models improved the performance by circa 1–2% in MRR.

Table 6.3: Link prediction results on WN18RR, F15K-237 and YAGO3-10. Results are obtained from corresponding papers. Bold and underlined entries denote best and second-best results. The dash (-) denotes values missing in the papers.

	WN18RR				**FB15K-237**				**YAGO3-10**			
	MRR	@1	@3	@10	MRR	@1	@3	@10	MRR	@1	@3	@10
TransE (Ruffinelli et al., 2019)	0.228	0.053	0.368	0.520	0.313	0.221	0.347	0.497	-	-	-	-
ConvE (Ruffinelli et al., 2019)	0.442	0.411	0.451	0.504	0.339	0.248	0.359	0.521	-	-	-	-
TuckER (Balažević et al., 2019c)	0.470	**0.443**	0.482	0.526	0.358	0.266	0.394	**0.544**	-	-	-	-
A2N (Bansal et al., 2019)	0.450	0.420	0.460	0.510	0.317	0.232	0.348	0.486	-	-	-	-
QuatE (Zhang et al., 2019)	**0.482**	0.436	**0.499**	**0.572**	0.311	0.221	0.342	0.495	-	-	-	-
HypER (Balažević et al., 2019b)	0.465	0.436	0.477	0.522	0.341	0.252	0.376	0.520	0.533	0.455	0.580	0.678
DistMult (Dettmers et al., 2018)	0.430	0.390	0.440	0.490	0.240	0.160	0.260	0.420	0.340	0.240	0.380	0.540
ConvE (Dettmers et al., 2018)	0.430	0.400	0.440	0.520	0.335	0.237	0.356	0.501	0.440	0.350	0.490	0.620
ComplEx (Dettmers et al., 2018)	0.440	0.410	0.460	0.510	0.247	0.158	0.275	0.428	0.360	0.260	0.400	0.550
REFE (Chami et al., 2020)	0.455	0.419	0.470	0.521	0.302	0.216	0.330	0.474	0.370	0.289	0.403	0.527
ROTE (Chami et al., 2020)	0.463	0.426	0.477	0.529	0.307	0.220	0.337	0.482	0.381	0.295	0.417	0.548
ATTE (Chami et al., 2020)	0.456	0.419	0.471	0.526	0.311	0.223	0.339	0.488	0.374	0.290	0.410	0.538
ComplEx-N3 (Chami et al., 2020)	0.420	0.390	0.420	0.460	0.294	0.211	0.322	0.463	0.336	0.259	0.367	0.484
MuRE (Chami et al., 2020)	0.458	0.421	0.471	0.525	0.313	0.226	0.340	0.489	0.283	0.187	0.317	0.478
RotatE (Sun et al., 2019)	0.476	0.428	0.492	0.571	0.338	0.241	0.375	0.533	0.495	0.402	0.550	0.670
QMult	0.438	0.393	0.449	0.537	0.346	0.252	0.383	0.535	**0.555**	**0.475**	**0.602**	**0.698**
OMult	0.449	0.406	0.467	0.539	0.347	0.253	0.383	0.534	0.543	0.461	0.592	0.692
ConvQ	0.457	0.424	0.470	0.525	0.343	0.251	0.376	0.528	0.539	0.459	0.587	0.687
ConvO	0.458	0.427	0.473	0.521	**0.366**	**0.271**	**0.403**	0.543	0.489	0.395	0.546	0.664

Table 6.4: Number of parameter comparisons on the WN18RR, FB15K-237 and YAGO3-10 datasets. The dash (-) denotes values missing in the papers.

	WN18RR	**FB15K-237**	**YAGO3-10**
QuatE (Zhang et al., 2019)	16.38M	5.82M	-
RotatE (Sun et al., 2019)	40.95M	29.32M	123.22M
QMult	16.38M	6.01M	49.30M
OMult	16.38M	6.01M	49.30M
ConvQ	21.51M	11.13M	54.42M
ConvO	21.51M	11.13M	54.42M

Table 6.5: Link prediction results via ensemble learning.

	WN18RR				FB15K-237				YAGO3-10			
	MRR	@1	@3	@10	MRR	@1	@3	@10	MRR	@1	@3	@10
Q-OMult	0.444	0.399	0.458	0.544	0.356	0.260	0.393	0.545	0.557	0.478	0.601	0.700
QMult-ConvQ	0.446	0.406	0.455	0.538	0.357	0.263	0.392	0.546	**0.561**	**0.483**	**0.606**	**0.703**
QMult-ConvO	0.449	0.410	0.459	0.536	**0.372**	0.275	**0.411**	0.564	0.543	0.460	0.594	0.693
OMult-ConvQ	0.444	0.403	0.453	0.537	0.357	0.262	0.391	0.547	0.558	0.478	0.602	0.700
OMult-ConvO	0.462	**0.425**	**0.475**	**0.539**	**0.372**	**0.277**	**0.411**	0.564	0.535	0.450	0.588	0.692
ConvQ-O-OMult	**0.463**	**0.425**	**0.475**	**0.539**	**0.372**	0.275	**0.411**	**0.567**	0.552	0.470	0.599	0.702

We conjecture that link prediction performance may be further improved through optimizing the impact of each model in the constructed ensemble.

Impact of Tail Entity Rankings

During our experiments, we observed that models often perform more accurately in predicting missing tail entities compared to predicting missing head entities, which was also observed in Bansal et al. (2019). Table 6.6 indicates that MRR performance based on *only tail entity rankings* are on average absolute 10% higher than MRR results based on *head and tail entity rankings* on FB15K-237 while such difference was not observed on WN18RR.

Link Prediction Per Relation and Direction

We reevaluate link prediction performance of some of the best-performing models from Table 6.3 in Tables 6.7 and 6.8. Allen et al. (2021) distinguish three types of relations: Type S relations are specialization relations such as `hypernym`, type C denote so-called generalized context-shifts and include `has_part` relations, and type R relations include so-called highly-related relations such as `similar_to`. Our results show that our approaches accurately rank missing tail and head entities for type R relations. For instance, our approaches perfectly rank (1.0 MRR) missing entities of symmetric relations (`verb_group` and `similar_to`). However, the direction of entity prediction has a significant impact on the results for non-symmetric type C relations. For instance, MRR performances of QMult, ConvQ, OMult, and ConvO differ by up to absolute 0.63 for the relation `member_of_domain_region`.

Table 6.6: Link prediction results based on only tail entity rankings.

	WN18RR				FB15K-237				YAGO3-10			
	MRR	@1	@3	@10	MRR	@1	@3	@10	MRR	@1	@3	@10
DistMult	0.430	0.410	0.440	0.480	0.370	0.275	0.417	0.568	-	-	-	-
ComplEx	0.420	0.380	0.430	0.480	0.394	0.303	0.434	0.572	-	-	-	-
ConvE	0.440	0.400	0.450	0.520	0.410	0.313	0.457	0.600	-	-	-	-
MINERVA	0.450	0.410	0.460	0.510	0.293	0.217	0.329	0.456	-	-	-	-
A2N	**0.490**	**0.450**	**0.500**	0.550	0.422	0.328	0.464	0.608	-	-	-	-
QMult	0.451	0.403	0.472	0.553	0.439	0.341	0.485	0.636	0.692	0.626	0.736	0.799
OMult	0.461	0.414	0.482	.559	0.440	0.340	0.486	0.636	0.689	0.623	0.733	0.801
ConvQ	0.470	0.437	0.482	0.538	0.441	0.344	0.482	0.632	0.674	0.612	0.715	0.783
ConvO	0.473	.442	0.491	0.535	0.465	0.367	0.512	0.654	0.622	0.535	0.682	774
Q-OMult	0.457	0.401	0.48	0.559	0.448	0.347	0.495	0.644	0.696	0.632	0.734	**0.804**
QMult-ConvQ	0.466	0.424	0.480	0.557	0.451	0.353	0.496	0.646	**0.697**	**0.635**	**0.737**	0.803
QMult-ConvO	0.474	0.435	0.487	0.558	0.467	0.367	0.516	**0.662**	0.680	0.610	0.727	0.800
OMult-ConvQ	0.471	0.430	0.487	**0.560**	0.452	0.354	0.495	0.647	0.696	0.632	0.735	0.803
OMult-ConvO	0.476	0.436	0.488	0.559	.466	0.366	0.515	**0.662**	0.676	0.602	0.724	0.803
ConvQ-O	0.477	0.442	0.494	0.548	**0.468**	**0.370**	**0.515**	0.661	0.675	0.603	0.724	0.795

Table 6.7: MRR link prediction results per relations on WN18RR. Ensemble refers to averaging predictions of ConvQ-ConvO-OMult.

Relation Name	Rel. Type	RotatE	QMult	ConvQ	OMult	ConvO	Ensemble
hypernym	S	**0.15**	0.10	0.14	0.11	0.13	0.13
instance_hypernym	S	0.32	0.35	0.37	0.36	0.37	**0.39**
member_meronym	C	**0.23**	0.22	0.20	**0.23**	0.20	**0.23**
synset_domain_topic_of	C	**0.34**	0.31	0.31	0.32	0.33	**0.34**
has_part	C	0.18	**0.19**	0.17	0.18	0.18	**0.19**
member_of_domain_usage	C	0.32	0.29	0.28	0.27	**0.33**	0.29
member_of_domain_region	C	0.20	0.25	**0.38**	0.30	0.37	**0.38**
derivationally_related_form	R	0.95	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**
also_see	R	0.59	**0.67**	0.65	0.66	0.66	0.66
verb_group	R	0.94	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
similar_to	R	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

Table 6.8: Link prediction results depending on the direction of prediction (head vs. tail prediction) on WN18RR. Ensemble refers to averaging predictions of ConvQ-ConvO-OMult.

	QMult	ConvQ	OMult	ConvO	Ensemble
(h, r, x)					
hypernym	0.12	**0.18**	0.13	**0.18**	0.17
instance_hypernym	0.53	0.56	0.53	0.57	**0.58**
member_meronym	**0.17**	0.09	0.16	0.08	0.14
synset_domain_topic_of	0.49	0.47	0.51	**0.52**	**0.52**
has_part	**0.15**	0.12	0.15	0.14	0.14
member_of_domain_usage	0.04	0.02	0.07	**0.08**	0.05
member_of_domain_region	0.05	**0.06**	0.05	**0.06**	0.05
derivationally_related_form	0.98	0.98	0.98	0.98	0.98
also_see	**0.67**	0.63	0.65	0.63	0.63
verb_group	**1.0**	**1.0**	**1.0**	**1.00**	**1.0**
similar_to	**1.0**	**1.0**	**1.0**	**1.00**	**1.0**
(x, r, t)					
hypernym	0.07	0.09	0.09	0.08	**0.10**
instance_hypernym	0.17	0.18	**0.19**	0.17	**0.19**
member_meronym	0.27	0.31	0.29	**0.33**	0.32
synset_domain_topic_of	0.13	0.14	0.13	**0.15**	**0.15**
has_part	0.22	0.22	0.22	0.22	**0.24**
member_of_domain_usage	0.53	0.54	0.47	**0.59**	0.54
member_of_domain_region	0.45	0.69	0.55	0.67	**0.70**
derivationally_related_form	0.98	0.98	0.98	**0.99**	0.98
also_see	0.68	0.67	0.66	**0.69**	0.68
verb_group	**1.00**	**1.0**	**1.0**	**1.0**	**1.0**
similar_to	**1.00**	**1.0**	**1.0**	**1.0**	**1.0**

Table 6.9: Link prediction results with the batch and unit normalization.

	Kinship				UMLS			
	MRR	H@1	@3	@10	MRR	@1	@3	@10
GNTP-Standard (Minervini et al., 2020)	0.72	0.59	0.82	0.96	0.80	0.70	0.88	0.95
GNTP-Attention (Minervini et al., 2020)	0.76	0.64	0.85	0.96	0.86	0.76	0.95	0.98
NTP (Minervini et al., 2020)	0.35	0.24	0.37	0.57	0.80	0.70	0.88	0.95
NeuralLP (Minervini et al., 2020)	0.62	0.48	0.71	0.91	0.78	0.64	0.87	0.96
MINERVA (Minervini et al., 2020)	0.72	0.60	0.81	0.92	0.83	0.73	0.90	0.97
ConvE (Dettmers et al., 2018)	0.83	0.74	0.92	0.98	0.94	0.92	0.96	0.99
QMult (batch)	**0.88**	**0.81**	**0.94**	**0.99**	**0.96**	**0.93**	**0.98**	**1.0**
QMult (unit)	0.69	0.58	0.78	0.90	0.77	0.69	0.82	0.93
OMult (batch)	0.87	0.80	**0.94**	**0.99**	0.95	0.91	**0.98**	**1.0**
OMult (unit)	0.69	0.57	0.77	0.89	0.76	0.66	0.82	0.93
ConvQ (batch)	0.86	0.77	0.93	0.98	0.92	0.86	**0.98**	**1.0**
ConvQ (unit)	0.61	0.49	0.68	0.85	0.55	0.45	0.59	0.75
ConvO (batch)	0.86	0.77	0.93	0.98	0.90	0.82	**0.98**	**1.0**
ConvO (unit)	0.65	0.53	0.72	0.86	0.56	0.46	0.61	0.78

The low performance of `hypernym` (type S) may stem from the fact that there are 184 triples in the test split of WN18RR where `hypernym` occurs with entities of which at least one did not occur in the training split (Demir and Ngonga Ngomo, 2021c). Models often perform poorly on type C relations but considerably better on type R relations corroborating findings by Allen et al. (2021).

Batch vs. Unit Normalization

We investigate the effect of using batch normalization, instead of unit normalization as previously proposed by Zhang et al. (2019). Table 6.9 indicates that the scaling effect of hypercomplex multiplications can be effectively alleviated by using the batch normalization technique. Replacing unit normalization with the batch normalization technique allows benefiting (1) from its regularization effect and (2) from its numerical stability. Through batch normalization, our models are able to control the rate of normalization and benefit from its implicit regularization effect (Ioffe and Szegedy, 2015).

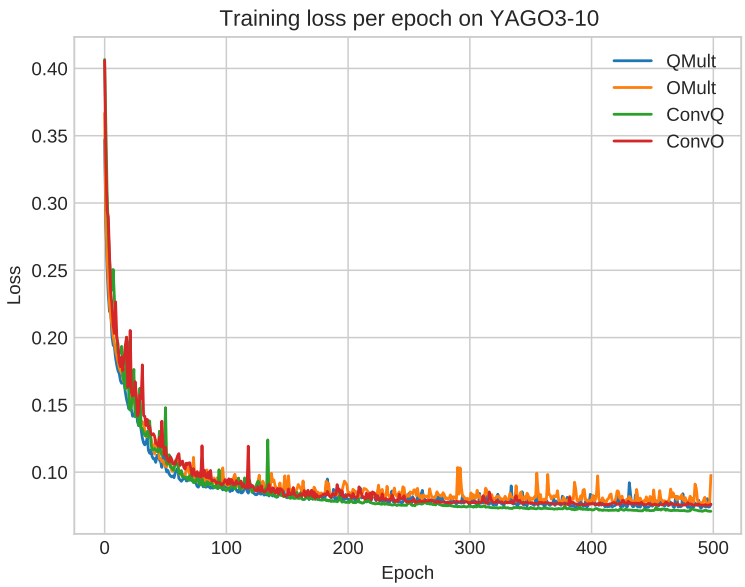


Figure 6.1: A plot of convergence during training on YAGO3-10.

Convergence on YAGO3-10

Figure 6.1 indicates that incurred binary cross entropy losses significantly decrease within the first 100 epochs. After the 300th iteration, ConvQ and ConvO appear to converge as losses do not fluctuate, whereas training losses of QMult and OMult continue fluctuating.

Link Prediction Results on Previous Benchmark Datasets

Table 6.10 reports results on WN18 and FB15K showing that our approaches ConvQ and ConvQ outperform state-of-the-art approaches in 6 out of 8 metrics on the datasets.

Semantic Constraint

we design a semantic constraint technique that incorporates the domain and range information of a relation at testing time to further boost link prediction performance. For a given $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}^{train}$, all entities $x \notin domain(r)$ are filtered out at computing the rank of missing head entity.

Table 6.10: Link prediction results on WN18 and FB15K.

	WN18					FB15K				
Model	Param.	MRR	Hit@10	Hit@3	Hit@1	Param.	MRR	Hit@10	Hit@3	Hit@1
TransE	-	0.495	0.943	0.888	0.113	-	0.463	0.749	0.578	0.297
TransR	-	0.605	0.940	0.876	0.335	-	0.346	0.582	0.404	0.218
ER-MLP	-	0.712	0.863	0.775	0.626	-	0.288	0.501	0.317	0.173
RESCAL	-	0.890	0.928	0.904	0.842	-	0.354	0.587	0.409	0.235
HolE	-	0.938	0.949	0.945	0.930	-	0.524	0.739	0.613	0.402
SimplE	-	0.942	0.947	0.944	0.939	-	0.727	0.838	0.773	0.660
TorusE	-	0.947	0.954	0.950	0.943	-	0.733	0.832	0.771	0.674
RotatE	-	0.947	0.961	0.953	0.938	-	0.699	0.872	0.788	0.585
QuatE	-	0.949	0.960	0.954	0.941	-	0.770	0.878	0.821	0.700
QuatE^2	-	0.950	0.962	0.954	0.944	-	**0.833**	0.900	0.859	**0.800**
QMult	16.39M	.975	.980	.976	.972	7.05M	0.755	0.896	0.819	0.668
OMult	16.39M	.975	**0.981**	.976	.972	7.05M	0.748	0.889	0.813	0.660
ConvQ	21.51M	**0.976**	.980	**0.977**	**0.973**	12.17M	.813	**0.923**	**0.868**	0.743
ConvO	21.51M	**0.976**	0.980	**0.977**	**0.973**	12.17M	0.810	**0.923**	0.865	0.739

All entities $x \notin range(r)$ are filtered out at computing the rank of missing tail entity. Table 6.11 reports link prediction results with semantic constraint technique.

6.2.2 Discussion

Our approaches often outperform many state-of-the-art approaches on all benchmark datasets. QMult and OMult outperform many state-of-the-art approaches including DistMult and ComplEx. These results indicate that scoring functions based on hypercomplex multiplications are more effective than scoring functions based on real and complex multiplications. This observation corroborates findings of Zhang et al. (2019). ConvO often perform slightly better than ConvQ on all datasets. Additionally, QMult and OMult perform particularly well on YAGO3-10. Figure 1 indicates that ConvQ and ConvO reach lower training errors than QMult and OMult on YAGO3-10.

Overall, superior performance of our models stems from (1) hypercomplex embeddings and (2) the inclusion of convolution operations. Our models are allowed to degrade into ComplEx or DistMult if necessary. Inclusion of the convolution operation followed by an affine transformation permits finding a good ratio between expressiveness and the number of parameters.

Table 6.11: Link prediction results on WN18RR, F15K-237 and YAGO3-10. Results are obtained from corresponding papers. Bold entries denote best results. The dash (-) denotes values missing in the papers. † represents applying the semantic constraint at prediction time

	WN18RR				**FB15K-237**				**YAGO3-10**			
	MRR	@1	@3	@10	MRR	@1	@3	@10	MRR	@1	@3	@10
QMult	0.438	0.393	0.449	0.537	0.346	0.252	0.383	0.535	0.555	0.475	0.602	0.698
QMult †	0.473	0.427	0.491	0.566	0.382	0.285	0.421	0.576	0.576	0.490	0.631	0.728
OMult	0.449	0.406	0.467	0.539	0.347	0.253	0.383	0.534	0.543	0.461	0.592	0.692
OMult †	0.484	0.444	0.504	0.563	0.381	0.284	0.418	0.576	0.563	0.480	0.612	0.716
ConvQ	0.457	0.424	0.470	0.525	0.343	0.251	0.376	0.528	0.539	0.459	0.587	0.687
ConvQ †	0.474	0.442	0.486	0.535	0.375	0.280	0.409	0.568	0.497	0.403	0.553	0.672
ConvO	0.458	0.427	0.473	0.521	0.366	0.271	0.403	0.543	0.489	0.395	0.546	0.664
ConvO †	0.471	0.441	0.484	0.529	0.398	0.301	0.437	0.592	0.545	0.463	0.594	0.694
ConEx	0.481	0.448	0.493	0.550	0.366	0.271	0.403	0.555	0.552	0.474	0.601	0.696
ConEx †	0.491	0.457	0.504	0.561	0.398	0.301	0.437	0.595	0.565	0.484	0.613	0.709

Trade-offs & Possible Improvements: During our experiments we observe that QMult and OMult consistently require less runtime than ConvQ and ConvO. This corroborate our previous findings at Chapter 5, i.e., as the number of operations increases to compute a score for a single triple, the required number of epochs for the convergence increases. For instance, Figure 6.1 shows that QMult and OMult reaches a better training loss than ConvQ and ConvO around the first 50 epochs. Consequently, to benefits 2D convolutions and non-linearity in ConvQ and ConvO, the computation budget for the training should permit longer training.

7

Kronecker Decomposition for Knowledge Graph Embeddings

Preamble: This chapter is based on Demir et al. (2022b).

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Julian Lienen, and Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Julian Lienen, and Axel-Cyrille Ngonga Ngomo. The code is available at https://github.com/dice-group/dice-embeddings.

Research Questions: In this work, we are concerned with the following two research questions:

1. **RQ1.** Can we design a knowledge graph embedding model to predict missing triples on large knowledge graphs?
2. **RQ2.** Can we answer RQ1. while ensuring the parameter efficiency of our models?

7.1 Methodology

In this section, we introduce KronE. Previous works have shown that pre-trained overparameterized language models can be effectively decomposed into smaller weight matrices (Sanh et al., 2019; Jiao et al., 2019; Sun et al., 2020; Rashid et al., 2021). Particularly, the Kronecker Decomposition (KD) based on the Kronecker Product (KP) elucidated in Section 2.7 leads to a significant reduction in the number of parameters with at most a mild cost of predictive accuracy (Edalati et al., 2021; Tahaei et al., 2021). Analogous to the aforementioned two works, findings of Zhang et al. (2021) and Wu (2016) suggest that training a neural network via KD on weight matrices results in increased parameter efficiency. We are interested in learning compressed KGE by applying KD during training. We aim to design a generic technique that can be applied in existing knowledge graph embedding (KGE) models to reduce the number of explicitly stored parameters while retaining their expressiveness. Importantly, through KD on embedding matrices, we aim to capture interactions within an embedding vector without requiring additional parameters. This is expected to encourage parameter reuse and reduces redundancy in model's parameters (Huang et al., 2017). In Table 7.1, we summarize the symbols used in this chapter.

Table 7.1: An overview of our notation used in Chapter 7.

Notation	Description
$\mathcal{G}$	Knowledge graph
$\mathcal{E}$,$\mathcal{R}$	Entities and relations, respectively
Θ	Model parameters
$\phi_\Theta(\cdot,\cdot,\cdot)$	Parameterized scoring function
$\mathbf{E}, \mathbf{R}$	Entity and relation embedding matrices, respectively
$\circ$	Hadamard product
$\otimes$	Kronecker product
d	Embedding vector dimensions
$\sigma(\cdot)$	Logistic sigmoid function
$\ell(\cdot,\cdot)$	Binary cross-entropy loss function

Most KGE models are designed as a parameterized scoring function $\phi_\Theta : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \mapsto \mathbb{R}$ that maps an input triple $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$ to a scalar value that is mapped to

the unit interval using the sigmoid/logistic function. This normalized scalar value is interpreted to reflect the likelihood of (h, r, t) is true. Θ denotes the parameters of ϕ that consists of an entity embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d}$, a relation embedding matrix $\mathbf{R} \in \mathbb{R}^{|\mathcal{R}| \times d}$, and other trainable parameters, such as affine transformation over input embeddings, convolutions, batch normalization or instance normalization. Assume that $\phi =:$ DistMult and $\Theta := \{\mathbf{E}, \mathbf{R}\}$, for a given $(\mathsf{h}, \mathsf{r}, \mathsf{t}) \in \mathcal{G}$, a score is computed as

$$\text{DistMult}(\mathsf{h}, \mathsf{r}, \mathsf{t}) = \mathbf{e}_h \circ \mathbf{e}_r \cdot \mathbf{e}_t. \tag{7.1}$$

In Equation 7.1, a triple score is computed thorough element-wise interactions between 3 d-dimensional real-valued embedding vectors, e.g., given $d = 2$ and, $a, b, c, d, e, f \in \mathbb{R}$, a triple score is $[a\ b] \circ [c\ d] \cdot [e\ f] = ace + bdf$. During training, the gradient of the set loss function (e.g., the binary cross entropy) w.r.t. 3 d-dimensional embedding vectors $\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t$ is computed. The gradient of the loss w.r.t. the first item a in $\mathbf{e}_h$ is computed in two steps: the gradient of the loss. w.r.t. the prediction is computed and distributed over the addition operation and the element-wise multiplication via ce. Yet, the interaction between (a, b) as well as between (a, a) are ignored, although a and b constituted $\mathbf{e}_h$ together. This stems from computing a scalar value via element-wise operations. These ignored interactions can be captured through applying KD on embedding vectors as follows

$$\begin{aligned}[aa\ ab\ ab\ bb] \circ [cc\ cd\ cd\ dd] \cdot [ee\ ef\ ef\ ff] = aacce \\ + 2(abcdef) \\ + bbddff\ .\end{aligned} \tag{7.2}$$

Through the middle term in Equation 7.2, interactions between (a, b) and (a, a) are incorporated without requiring additional parameters. Hence, we argue that DistMult defined in Equation 7.1 is less expressive than its KD variation defined in Equation 7.2. Consequently, if the both models are trained properly, the latter model is expected to perform at least as well as the former model in the link prediction problem. Importantly, the number of possible interactions within an embedding vector grows by $(d + 1)^2$ when d grows by 1. Although these interactions are expected to encourage *feature reuse* and *reduce redundancy* in model's parameters, the computation of KP on embedding vectors may become a bottleneck due to high computationally complexity of KD. In this setting, $\mathbf{E}$ and $\mathbf{R}$ can be seen as compressed embeddings of entities and relations. Hence, given a triple (h, r, t), their compressed embeddings are retrieved and a triple

score is computed via their decompressed embeddings. Our technique can be readily applied on many multiplicative based KGE models. For the numerical stability, the batch normalization or layer normalization can be applied to reduce the dependence of gradients on the scale of embedding vectors as these normalization techniques have beneficial effect on the gradient flow through models Ioffe and Szegedy (2015); Ba et al. (2016); Xu et al. (2019).

7.1.1 Kronecker Decomposition for Relation Embeddings

KD on relation embeddings of DistMult can be applied as

$$\text{KD-Rel-DistMult}(\mathsf{h}, \mathsf{r}, \mathsf{t}) = \mathbf{e}_h \circ (\mathbf{e}_r \otimes \mathbf{e}_r) \cdot \mathbf{e}_t, \tag{7.3}$$

where $\mathbf{e}_h, \mathbf{e}_t \in \mathbf{E} : \mathbf{E} \in \mathbb{R}^d$, and $\mathbf{e}_r \in \mathbf{R} : \mathbf{R} \in \mathbb{R}^{\sqrt{d}}$. The parameter gain can be computed as $(d - \sqrt{d}) \times (|\mathcal{R}|)$. Note that scores of triples are still computed based on 3 d-dimensional embedding vectors. As the size of the input graph and the number of relations grow, the parameter gain becomes more tangible.

7.1.2 Kronecker Decomposition of Embeddings for 1vsAll

Most KGs contain more entities than relations (see Table 7.2). Hence, the potential parameter gain of applying KD on entity embeddings is expectedly larger than applying KD on relation embeddings. Yet, applying KD on tail entities embeddings in 1vsAll or KvsAll increase the runtime and memory requirements as these training strategies require considering all entities to compute an incurred loss for a single prediction. To alleviate this limitation, KD can be applied as

$$\text{KD-DistMult}(\mathsf{h}, \mathsf{r}, \mathsf{t}) = \sum_{i}^{d} R(\big(\mathbf{e}_h \otimes \mathbf{e}_h) \circ (\mathbf{e}_r \otimes \mathbf{e}_r)\big)_i \cdot \mathbf{e}_t, \tag{7.4}$$

where $R(\cdot)$ reshapes a d^2 dimensional vector into d by d matrix. The matrix vector product of the resulting vector and $\mathbf{e}_t$ is summed to obtain a score for an input triple (h, r, t).

In Section 7.1.1 and 7.1.2, we elucidated applying KD on embedding vectors. Yet, KD can be also applied to decompose linear transformation weight matrices in feed-forward and convolutional based embedding models during training.

Table 7.2: An overview of benchmark datasets.

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{G}^{\text{Train}}\rvert$	$\lvert\mathcal{G}^{\text{Val.}}\rvert$	$\lvert\mathcal{G}^{\text{Test}}\rvert$
UMLS	136	93	10,432	1304	1965
KINSHIP	105	51	17,088	2136	3210

7.2 Experiments & Results

7.2.1 Experiments

Experimental Setup

All experiments were carried out on a single core of a server running Ubuntu 18.04 with 126 GB RAM with 16 Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz processors.

Datasets

We used two of benchmark datasets (KINSHIP and UMLS). The Kinships knowledge graph describes the 26 different kinship relations of the Alyawarra tribe and the unified medical language system (UMLS) knowledge graph describes 135 medical entities via 49 relations describing (Trouillon and Nickel, 2017). Note that we omitted WN18RR, FB15K-237, and YAGO3-10 from our experiments for two reasons: First, we aim to conduct experiments to quantify the impact of Kronecker Decomposition in a fine-grained many unique configurations with very large number of epochs (1000). KINSHIP and UMLS datasets are considerably smaller datasets compared to WN18RR, FB15K-237, and YAGO3-10. Hence, training three models with 21 unique configurations for 1000 epochs on WN18RR, FB15K-237, and YAGO3-10 can take up to a month. Moreover, two recent works showed that these datasets involve entities on the validation and test data splits that do not occur in the train split (Broscheit et al., 2020; Demir and Ngonga Ngomo, 2021c).

Evaluation

We use the standard metrics *filtered* MRR and hits at N (H@N) for link prediction (Dettmers et al., 2018; Balažević et al., 2019c,b). We also evaluate link prediction performances of approaches with respect to number of parameters.

In 1vsAll or KvsAll, for each test triple $(\mathsf{h}, \mathsf{r}, \mathsf{t})$, the score of $(\mathsf{h}, \mathsf{r}, \mathsf{x})$ triples for all $x \in \mathcal{E}$ is computed. Based on these scores, the filtered ranking $rank_t$ of the triple having t is obtained. Then the MRR: $\frac{1}{|\mathcal{G}^{\text{test}}|} \sum_{(\mathsf{h},\mathsf{r},\mathsf{t}) \in \mathcal{G}^{\text{test}}} \frac{1}{rank_t}$ is computed. Next, Hi@1, H@3, and H@10 are computed in literature (Dettmers et al., 2018; Balažević et al., 2019c; Ruffinelli et al., 2019). The number of parameters consists of $|\mathbf{E}|$, $|\mathbf{R}|$, and all other trainable parameters. Moreover, we illustrate our methodology with DistMult for three reasons: (1) many recent KGE model can be seen as an effective extension of DistMult, i.e., the Hadamard product followed by an inner product of input embeddings (see Section 2.4.1). (2) Findings of Ruffinelli et al. (2019) indicate that DistMult perform competitive performance provided that it is properly trained. (3) DistMult requires less floating point operations than more sophisticated recent models. Hence, it can be trained in less time (Costabello et al., 2019; Valeriani, 2020).

Hyperparameter Optimization

We train approaches with the 1vsAll training strategy as commonly done in the literature (Lacroix et al., 2018; Ruffinelli et al., 2019). All models are trained with the ADAM optimizer (Kingma and Ba, 2014) and minimize the binary cross entropy loss function (see Equation (6.12)). We use the same loss function for all approaches on all datasets as Mohamed et al. (2019) previously showed that generalization performance of KGE models can be significantly influenced by the choice of the loss function. Moreover, we apply the batch normalization to facilitate numerical stability and accelerate convergence during training (Ioffe and Szegedy, 2015). We trained the model for 1000 epochs with a learning rate of .01 and a batch size of 1024. We further optimized the embedding vector sizes in $\{4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 289, 324, 361, 400\}$ using a grid search. For label smoothing and label relaxation, we consider $\alpha \in \{.1, .2\}$. This results in training models with soften target values, i.e., $0 < \mathbf{y}^{(i)} < 1$. Through large parameter sweep in the embedding vector size, we aim to obtain a fine-grained performance analysis. As elucidated in Section Section 9.1, we hypothesize that the impact of learning compressed embeddings becomes more tangible as the size of the embedding vector increases. Importantly, we also share test and train performance of all operations with all configurations. By doing so, we aim to quantify the impact of applying label smoothing and label relaxation during training and testing separately.

We do not apply an explicit regularization (e.g., L1, L2 regularization or the Dropout technique (Srivastava et al., 2014)) as regularization techniques may not allow to observe any regularization effect of label smoothing and label relaxation.

7.2.2 Results

Standard Link Prediction

Table 7.3 reports link prediction results on benchmark datasets. Overall results indicate that applying KD on embedding matrices reduces the number of required parameters, while yielding a competitive performance. KD-DistMult outperforms DistMult and KD-Rel-DistMult in 9 out of 10 metrics in Table 7.3 with surprisingly less parameters. Specifically, KD-DistMult outperforms DistMult and KD-DistMult, while requiring **18.2**× and **11.4**× fewer number of parameters than DistMult and KD-Rel-DistMult on UMLS, respectively. Similarly, KD-DistMult requires **14.6**× and **14.2**× fewer number of parameters than DistMult and KD-Rel-DistMult on KINSHIP, respectively. After observing these results, we delved into the details of training process to validate the existence of overfitting. To this end, we evaluated each model on the training datasets and added the MRR and Hit@N scores in Table 7.3.

Table 7.3: Link prediction results on UMLS and KINSHIP. $|\Theta|$ denotes the number of parameters. Bold entries denote best results.

	UMLS					**KINSHIP**				
	$\|\Theta\|$	MRR	@1	@3	@10	$\|\Theta\|$	MRR	@1	@3	@10
DistMult	67,915	0.517	0.441	0.536	0.659	46,818	0.568	0.500	0.593	0.693
on training set		0.995	0.992	1.00	1.00		0.919	0.876	0.952	0.991
KD-Rel-DistMult	42,619	0.531	0.432	0.584	**0.684**	45,420	0.562	0.493	0.588	0.694
on training set		0.996	0.993	0.999	1.00		0.914	0.867	0.953	0.992
KD-DistMult	**3,728**	**0.541**	**0.447**	**0.598**	**0.684**	**3,200**	**0.599**	**0.534**	**0.631**	**0.709**
on training set		0.814	0.704	0.904	0.989		0.705	0.572	0.804	0.950

These results suggest that (a) all models seem to suffer from overfitting and (b) increasing embedding vector size does not proportionally increase the expressiveness of the model.

DistMult with **59**% more number of parameters than KD-Rel-DistMult does not lead to a significant change in MRR and Hit@N scores on the training datasets. These results also highlight the importance of applying extensive extensive hyperparameter optimization and model calibration to increase generalization performances.

Link Prediction with Model Calibration

To observe the impact of model calibration, we retrain models with label smoothing and label relaxation. Table 7.5 shows that performances of models are improved with model calibration on UMLS, whereas performances are not increased and in some circumstances decreased. Importantly, model calibration seem to help more on those models that suffer greatly from the overfitting. For instance, models seem to perform better without model calibration on KINSHIP, where the overfitting is less severe.

Link Prediction under Noise

We were interested to observe the impact of adding noisy triples into the training dataset in the link prediction task, since many real-world KGs contains noisy triples. To this end, we add 10% noise in the training splits and evaluate models. Table 7.4 suggest that 10% noise in the input data decrease the standard DistMult model by absolute 7% in MRR on KINSHIP, whereas performance of KD-Rel-DistMult and KD-DistMult are more robust against the input noise. Surprisingly, KD-Rel-DistMult reaches the highest MRR, Hit@1 and Hit@3 scores throughout our experiments with 10% noisy data. Bishop (1995) showed that the addition of noise to the numerical input training data lead to significant improvements in generalization performance. Our results signal that adding additional noise in the structured data may have the similar effect.

Parameter Analysis

Table 7.6 and Table 7.7 report performances with a wide range of embedding vector sizes. Overall, our results corroborate our hypothesis, namely, as the size of embedding vectors decreases, benefits of applying KD becomes less tangible. More specifically, Table 7.6 suggests that as $|\Theta|$ grows KD-Rel-DistMult and KD-DistMult perform quite well compared to DistMult on the both benchmark datasets.

Table 7.4: Link prediction results on noisy UMLS and KINSHIP. $|\Theta|$ denotes the number of parameters. Bold entries denote best results.

	UMLS					KINSHIP				
	$\|\Theta\|$	MRR	@1	@3	@10	$\|\Theta\|$	MRR	@1	@3	@10
DistMult	23,500	0.448	0.319	0.484	0.741	16,200	0.523	0.452	0.538	0.665
	28,435	0.470	0.382	0.497	0.677	19,602	0.512	0.444	0.523	0.655
	33,840	0.423	0.335	0.411	0.621	23,328	0.508	0.439	0.517	0.658
	39,715	0.518	0.418	0.556	0.741	27,378	0.510	0.440	0.524	0.658
	46,060	0.489	0.355	0.586	0.738	31,752	0.512	0.438	0.529	0.668
	52,875	0.465	0.339	0.500	0.677	36,450	0.518	0.445	0.534	0.671
	60,160	0.422	0.340	0.413	0.591	41,472	0.520	0.447	0.536	0.679
	67,915	0.485	0.396	0.497	0.699	46,818	0.517	0.443	0.534	0.679
	76,140	0.532	0.436	0.562	**0.746**	52,488	0.523	0.447	0.547	0.674
	84,835	0.448	0.341	0.467	0.695	58,482	0.529	0.453	0.551	0.684
	94,000	0.466	0.343	0.541	0.662	64,800	0.529	0.454	0.556	0.685
KD-Rel-DistMult	15,130	0.521	0.428	0.534	0.730	11,610	0.540	0.477	0.554	0.674
	18,205	0.550	0.467	0.578	0.690	13,992	0.516	0.449	0.528	0.655
	21,564	0.456	0.349	0.499	0.676	16,596	0.507	0.442	0.514	0.644
	25,207	0.469	0.338	0.591	0.718	19,422	0.500	0.439	0.502	0.631
	29,134	0.515	0.422	0.561	0.701	22,470	0.503	0.437	0.511	0.645
	33,345	0.580	0.500	0.617	0.701	25,740	0.503	0.436	0.509	0.647
	37,840	0.526	0.432	0.544	0.718	29,232	0.506	0.438	0.515	0.644
	42,619	0.567	0.493	0.607	0.693	32,946	0.505	0.431	0.519	0.655
	47,682	0.439	0.322	0.482	0.678	36,882	0.511	0.440	0.529	0.661
	53,029	0.554	0.490	0.574	0.671	41,040	0.511	0.441	0.524	0.657
	58,660	**0.583**	**0.503**	**0.632**	0.728	45,420	0.514	0.442	0.531	0.666
KD-DistMult	2,330	0.461	0.339	0.527	0.651	1,600	0.562	0.487	0.592	0.699
	2,563	0.331	0.233	0.343	0.496	1,760	0.573	0.501	0.609	**0.703**
	2,796	0.376	0.252	0.448	0.622	1,920	0.567	0.497	0.599	0.696
	3,029	0.346	0.254	0.370	0.506	2,080	0.576	0.505	0.604	0.702
	3,262	0.374	0.274	0.391	0.541	2,240	0.567	0.498	0.589	**0.703**
	3,495	0.393	0.259	0.480	0.559	2,400	0.569	0.502	0.597	0.690
	3,728	0.452	0.329	0.529	0.626	2,560	0.572	0.503	0.603	0.697
	3,961	0.374	0.276	0.385	0.655	2,720	0.566	0.501	0.590	0.694
	4,427	0.358	0.247	0.376	0.589	2,880	0.578	0.509	0.607	**0.703**
	4,194	0.545	0.438	0.596	0.744	3,040	0.584	0.520	0.609	0.698
	4,660	0.366	0.258	0.406	0.541	3,200	**0.586**	**0.521**	**0.612**	0.612
avg. DistMult	55,225	0.470	0.364	0.501	0.691	38,070	0.518	0.446	0.535	0.670
avg. KD-Rel-DistMult	34,765	**0.524**	**0.431**	**0.565**	**0.700**	26,850	0.510	0.443	0.521	0.653
avg. KD-DistMult	**3,495**	0.398	0.287	0.441	0.594	**2,400**	**0.573**	**0.504**	**0.601**	**0.691**

7.2.3 Discussion

We conjecture that all KGE models may benefit from an extensive hyperparameter optimization. Yet, here, we were interested in relative link prediction performances under optimizing solely the embedding vector size. We aimed to observe possible benefits of learning compressed embeddings in link prediction task. Table 7.3 and Table 7.7 suggest that benefits of learning compressed embeddings becomes more beneficial as the embedding vector size grows. Increasing the size of embedding vectors in DistMult does not decrease training loss as well as MRR and Hit@N scores on training datasets. This may stem from the fact that increasing the size of embeddings increases redundancy in the parameters, hence does not improve training and test performance of DistMult. Yet, Table 7.5 shows that model calibration consistently improved generalization performances on UMLS, while performances on KINSHIP are not improved as such. In our experiments, performance of KD-Rel-DistMult and KD-DistMult suggest that learning compressed knowledge graph embeddings does not only lead to a competitive and sometimes superior performance but also decrease the the need of over-parameterization. We argue that a comprehensive study including many recent state-of-the-art knowledge graph embedding models on benchmark datasets from different domains is needed to further analyze the benefits of learning compressed embeddings via KD.

Trade-offs & Possible Improvements: During our experiments, we observe that using KronE consistently increased total runtimes. Consequently, the benefits of using KronE come with a cost of increased runtimes. The performance of KronE may be further improved by replacing the plain Kronecker product with the Kronecker attention operator (Gao et al., 2020).

Table 7.5: Link prediction results on UMLS and KINSHIP with model calibration. Rows with on training set report the performances on the training dataset. |Θ|, LS, LR denote the number of parameters, Label Smoothing and Label Relaxation, respectively. Bold entries denote best results.

	UMLS					**KINSHIP**				
	|Θ|	MRR	@1	@3	@10	|Θ|	MRR	@1	@3	@10
DistMult	67,915	0.517	0.441	0.536	0.658	46,818	0.568	0.499	0.593	0.693
on training set		0.995	0.992	1.000	1.000		0.919	0.876	0.952	0.991
with LS $\alpha = .1$		0.568	0.499	0.602	0.707		0.515	0.417	0.563	0.685
on training set		0.996	0.993	1.000	1.000		0.918	0.845	0.954	0.992
with LS $\alpha = .2$		0.548	0.475	0.582	0.690		0.502	0.398	0.555	0.667
on training set		0.995	0.992	0.999	1.000		0.916	0.871	0.952	0.992
with LR $\alpha = .1$		0.552	0.436	0.630	0.723		0.567	0.499	0.592	0.694
on training set		0.995	0.992	1.000	1.000		0.919	0.875	0.952	0.992
with LR $\alpha = .2$		**0.579**	**0.487**	**0.648**	0.725		0.567	0.499	0.592	0.695
on training set		0.995	0.992	1.000	1.000		0.917	0.873	0.952	0.992
KD-Rel-DistMult	42,619	0.531	0.432	0.584	0.684	32,946	0.556	0.487	0.580	0.689
on training set		0.996	0.993	0.999	1.000		0.913	0.865	0.951	0.992
with LS $\alpha = .1$		0.565	0.493	0.611	0.704		0.500	0.394	0.555	0.677
on training set		0.996	0.992	0.999	1.000		0.913	0.866	0.951	0.991
with LS $\alpha = .2$		0.592	0.531	0.617	0.704		0.504	0.403	0.556	0.671
on training set		0.994	0.990	0.999	1.000		0.907	0.855	0.949	0.991
with LR $\alpha = .1$		0.543	0.444	0.602	0.692		0.555	0.486	0.577	0.692
on training set		0.996	0.993	0.999	1.000		0.912	0.864	0.952	0.992
with LR $\alpha = .2$		0.562	0.476	0.600	0.718		0.555	0.487	0.576	0.689
on training set		0.993	0.999	1.000	1.000		0.912	0.863	0.952	0.992
KD-DistMult	3,728	0.541	0.447	0.598	0.684	3,200	**0.599**	**0.534**	**0.631**	0.709
on training set		0.814	0.704	0.904	0.989		0.665	0.519	0.774	0.936
with LS $\alpha = .1$		0.592	0.511	0.627	0.751		0.519	0.382	**0.621**	0.705
on training set		0.796	0.677	0.897	0.984		0.640	0.491	0.745	0.928
with LS $\alpha = .2$		0.572	0.475	0.623	**0.754**		0.486	0.326	0.605	0.704
on training set		0.781	0.655	0.888	0.980		0.629	0.476	0.730	0.930

Table 7.6: Link prediction results on UMLS and KINSHIP with the highest half of the the parameter sweep in the number of parameters $|\Theta|$. Bold entries denote best results.

	UMLS					**KINSHIP**				
	$\|\Theta\|$	MRR	@1	@3	@10	$\|\Theta\|$	MRR	@1	@3	@10
DistMult	28,435	0.430	0.346	0.430	0.650	19,602	0.556	0.487	0.585	0.687
	33,840	0.439	0.357	0.455	0.545	23,328	0.563	0.494	0.589	0.695
	39,715	0.425	0.344	0.435	0.596	27,378	0.562	0.493	0.587	0.695
	46,060	0.484	0.419	0.480	0.585	31,752	0.563	0.494	0.588	0.699
	52,875	0.507	0.439	0.525	0.646	36,450	0.565	0.498	0.589	0.693
	60,160	0.459	0.363	0.487	0.646	41,472	0.567	0.498	0.598	0.700
	67,915	0.517	0.441	0.536	0.659	46,818	0.568	0.500	0.593	0.693
	76,140	0.471	0.374	0.485	0.677	52,488	0.548	0.460	0.596	0.697
	84,835	0.454	0.354	0.496	0.632	58,482	0.567	0.498	0.598	0.697
	94,000	0.436	0.353	0.439	0.587	64,800	0.563	0.493	0.591	0.697
KD-Rel-DistMult	18,205	0.544	**0.475**	0.576	0.665	13,992	0.546	0.478	0.568	0.684
	21,564	0.532	0.426	0.576	**0.735**	16,596	0.544	0.462	0.580	0.691
	25,207	0.455	0.368	0.473	0.639	19,422	0.546	0.476	0.567	0.687
	29,134	0.477	0.373	0.508	0.661	22,470	0.544	0.472	0.571	0.688
	33,345	**0.525**	0.452	0.548	0.642	25,740	0.548	0.475	0.576	0.691
	37,840	0.516	0.441	0.549	0.669	29,232	0.553	0.484	0.578	0.689
	42,619	0.531	0.432	**0.584**	0.684	32,946	0.556	0.487	0.580	0.689
	47,682	0.483	0.397	0.519	0.640	36,882	0.390	0.155	0.578	0.694
	53,029	**0.525**	0.447	0.554	0.643	41,040	0.559	0.488	0.587	0.693
	58,660	0.435	0.348	0.438	0.598	45,420	0.562	0.493	0.588	0.694
KD-DistMult	2,563	0.357	0.275	0.374	0.453	1,760	0.556	0.487	0.581	0.686
	2,796	0.403	0.276	0.468	0.637	1,920	0.579	0.509	0.613	0.702
	3,029	0.354	0.271	0.375	0.492	2,080	0.567	0.493	0.606	0.698
	3,262	0.357	0.267	0.372	0.470	2,240	0.577	0.511	0.605	0.701
	3,495	0.361	0.252	0.382	0.618	2,400	0.572	0.501	0.607	0.706
	3,728	0.541	0.447	0.598	0.684	2,560	0.591	0.526	0.621	0.703
	3,961	0.407	0.282	0.474	0.645	2,720	0.587	0.521	0.619	0.701
	4,194	0.379	0.302	0.392	0.461	2,880	0.591	0.526	0.620	0.708
	4,427	0.382	0.285	0.390	0.605	3,040	**0.599**	**0.535**	0.628	0.707
	4,660	0.505	0.397	0.581	0.697	3,200	**0.599**	0.534	**0.631**	**0.709**
avg. DistMult	58,397	0.462	0.379	0.477	0.622	40,257	0.562	0.491	0.591	0.695
avg. KD-Rel-DistMult	36,728	**0.502**	**0.416**	**0.532**	**0.658**	28,374	0.535	0.447	0.577	0.690
avg. KD-DistMult	**3,611**	0.405	0.305	0.441	0.576	**2,480**	**0.582**	**0.514**	**0.613**	**0.702**

Table 7.7: Link prediction results on UMLS and KINSHIP with the lowest half of the the parameter sweep in the number of parameters $|\Theta|$. Bold entries denote best results.

	UMLS					KINSHIP				
	$\|\Theta\|$	MRR	@1	@3	@10	$\|\Theta\|$	MRR	@1	@3	@10
DistMult	940	0.475	0.378	0.501	0.669	648	0.450	0.291	0.559	0.686
	2,115	0.550	0.472	0.579	0.668	1,458	0.582	0.516	0.607	0.707
	3,760	0.493	0.336	0.633	0.757	2,592	0.607	0.554	0.634	0.712
	5,875	0.584	0.498	0.626	0.749	4,050	0.613	0.554	0.640	0.714
	8,460	0.488	0.363	0.567	0.681	5,832	0.612	0.555	0.631	0.709
	11,515	0.409	0.326	0.417	0.601	7,938	0.597	0.533	0.625	0.710
	15,040	0.473	0.378	0.502	0.679	10,368	0.581	0.519	0.605	0.699
	19,035	0.548	0.476	0.590	0.709	13,122	0.560	0.492	0.584	0.690
	23,500	0.489	0.409	0.518	0.598	16,200	0.549	0.480	0.570	0.683
KD-Rel-DistMult	754	0.385	0.220	0.550	0.683	546	0.508	0.428	0.532	0.679
	1,557	0.537	0.444	0.582	0.692	1,152	0.382	0.137	0.581	0.694
	2,644	0.531	0.453	0.559	0.646	1,980	0.579	0.508	0.609	0.710
	4,015	0.547	0.466	0.579	0.694	3,030	0.591	0.523	0.621	0.712
	5,670	0.457	0.367	0.487	0.635	4,302	0.599	0.536	0.620	0.709
	7,609	0.560	0.431	0.657	0.742	5,796	0.592	0.529	0.615	0.709
	9,832	0.555	0.477	0.576	0.716	7,512	0.581	0.517	0.607	0.703
	12,339	0.507	0.415	0.550	0.679	9,450	0.575	0.512	0.593	0.698
	15,130	0.565	0.503	0.586	0.687	11,610	0.560	0.493	0.581	0.693
KD-DistMult	466	0.354	0.266	0.384	0.539	320	0.358	0.358	0.400	0.528
	699	0.321	0.215	0.309	0.564	480	0.428	0.380	0.421	0.508
	932	0.347	0.252	0.357	0.643	640	0.492	0.422	0.505	0.643
	1,165	0.360	0.260	0.381	0.503	800	0.524	0.454	0.547	0.660
	1,398	0.335	0.225	0.347	0.620	960	0.527	0.457	0.553	0.655
	1,631	0.375	0.225	0.493	0.651	1,120	0.550	0.483	0.573	0.676
	1,864	0.357	0.248	0.362	0.686	1,280	0.575	0.503	0.607	0.708
	2,097	0.355	0.244	0.373	0.648	1,440	0.572	0.500	0.603	0.707
	2,330	0.533	0.433	0.588	0.737	1,600	0.557	0.488	0.587	0.689
avg. DistMult	10,026	0.501	0.404	0.548	0.679	6,912	**0.572**	**0.499**	**0.606**	**0.701**
avg. KD-Rel-DistMult	6,616	**0.516**	**0.420**	**0.569**	**0.686**	5,042	0.552	0.465	0.595	**0.701**
avg. KD-DistMult	**1,398**	0.371	0.263	0.399	0.621	**960**	0.509	0.449	0.534	0.642

8

Polyak Parameter Ensemble for Knowledge Graph Embeddings

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Axel-Cyrille Ngonga Ngomo. The code is available at https://github.com/dice-group/dice-embeddings.

Research Questions: In this work, we are concerned with the following two research questions:

1. **RQ1.** Can we design a knowledge graph embedding model to predict missing triples on large knowledge graphs?
2. **RQ2.** Can we answer RQ1. while ensuring the parameter efficiency of our models?

8.1 Methodology

In this section, we introduce PPE. Prediction averaging improves the generalization performance of various knowledge graph embedding models in link prediction. Yet, prediction averaging comes with two disadvantages: the computational overhead of training multiple models and increased latency and memory requirements at test time. PPE improves the generalization performance of KGE models in the link prediction problem *with virtually no additional computational cost.* To this end, PPE maintains a running weighted average of parameters at each epoch interval. By utilizing the noisy approximation of mini-batch gradients at each epoch interval, PPE constructs a high performing parameter ensemble model with an expense of training a single knowledge graph embedding model. In Table 8.1, we summarize the symbols used in this chapter.

Table 8.1: An overview of our notation used in Chapter 8.

Notation	Description
$\mathcal{G}$	Knowledge graph
$\mathcal{E}$,$\mathcal{R}$	Sets of entities and relations, respectively
$\mathbf{w}$	Model parameters
d	Embedding vector dimension
$\mathcal{D}, \mathcal{B}$	Training dataset and a subset of $\mathcal{D}$
$\ell(\cdot)$	Binary cross entropy loss function
$\mathcal{L}_{\mathcal{B}_t}(\mathbf{w}_t)$	A mini-batch loss based at time t

The mini-batch SGD update can be defined as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \frac{1}{|\mathcal{B}_t|} \sum_b \nabla_{\mathbf{w}} \ell_b(\mathbf{w}_t) = \mathbf{w}_t - \eta \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}_t}(\mathbf{w}_t), \tag{8.1}$$

where $\mathbf{w}_t \in \mathbb{R}^d, \eta_t > 0$ and $\mathcal{B}_t := \{(\mathbf{x}_b, y_b)\}_{b=1}^m$ denote the model parameters, the learning rate, a randomly sampled training data points from the training dataset $\mathcal{B} \subset \mathcal{D}$ at a time t, respectively. $\nabla_{\mathbf{w}} \ell_b(\mathbf{w}_t)$ denotes the gradient of the loss function on the bases

of a single $(\mathbf{x}_b, y_b)$ w.r.t. $\mathbf{w}_t$. Let $\mathbf{w}_t \approx \mathbf{w}^* = \arg\min_{\mathbf{w}} \mathcal{L}_{\mathcal{D}}(\mathbf{w})$ be given. Two consecutive mini-batch SGD updates can be defined as

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}_t}(\mathbf{w}_t) \tag{8.2}$$

$$\mathbf{w}_{t+2} = \mathbf{w}_{t+1} - \eta_{t+1} \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}_{t+1}}(\mathbf{w}_{t+1}). \tag{8.3}$$

A sequence of mini-batch updates leads $\mathbf{w}_t$ to hover/circle around $\mathbf{w}^*$ provided that $\nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}_t) \neq \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{D}}(\mathbf{w}_t)$ and $\eta_t \not\to 0$ as $t \to +\infty$. Since the variance of the fluctuations around $\mathbf{w}^*$ is proportional to η, decreasing η is necessary (LeCun et al., 2002). The former condition of the noisy gradient approximation often holds, since $\mathcal{D}$ does not consists copies of a single data point (LeCun et al., 2002). Using an optimizer (e.g., Adam) adapting η_t w.r.t. $\nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}_t)$ or using a learning rate scheduling technique can often alleviate the issue in practice. Yet, we conjecture that a parameter ensemble model can be constructed from a linear combination of all parameter vectors obtained at each epoch interval. By this, (1) this circling issue can be alleviated regardless of the optimizer and (2) a high performing parameter ensemble is constructed with virtually no additional computational cost. Hence, we propose PPE (Polyak Parameter Ensemble) technique defined as

$$\mathbf{w}_{\text{PPE}} = \text{PPE}(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \boldsymbol{\alpha}_i \odot \mathbf{w}_i, \tag{8.4}$$

where $\mathbf{w}_i \in \mathbb{R}^d$ and $\boldsymbol{\alpha}_i$ with $\sum_i^N \boldsymbol{\alpha}_i = 1$) denote a parameter vector of a model at the end of the i-th epoch, and a scalar weight/ensemble coefficient for the i-th epoch, respectively. $\odot$ multiplies every element of $\mathbf{w}_i$ with a scalar $\boldsymbol{\alpha}_i$. Therefore, $\mathbf{w}_{\text{PPE}}$ is a linear combination of all parameter vectors obtained at each epoch. At each epoch i, PPE updates the running weighted average of parameters by a scalar matrix multiplication. Setting all $\boldsymbol{\alpha}_{0:N-1} = 0$ and $\boldsymbol{\alpha}_{N-1:N} = 1$ results in obtaining a parameter ensemble model that is only influenced by a parameter vector obtained at the end of the final epoch. Using positive equal ensemble coefficients $\boldsymbol{\alpha}_i = \frac{1}{N}$ corresponds to applying the Polyak averaging technique at each epoch interval. Next, we show that using such $\boldsymbol{\alpha}$ mitigates the issue of hovering around $\mathbf{w}^*$. A parameter vector $\mathbf{w}_{i+1}$ at the end of the i-th epoch can derived as

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \sum_{t=1}^{T} \eta_{(i,t)} \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}_{(i,t)}}\left(\mathbf{w}_{(i,t)}\right), \tag{8.5}$$

where T denotes the number of mini-batches to iterative over the training dataset. $\eta_{(i,t)}$ and $\mathcal{L}_{\mathcal{B}_{(i,t)}}(\mathbf{w}_{(i,t)})$ denote the learning rate and the incurred mini-batch loss for the t-th mini-batch step in the i-th epoch, respectively. Assume that $T = 2$ and $N = 2$ with η, a parameter ensemble model is obtained as follows

$$\begin{aligned}\mathbf{w}_{\text{PPE}} \;=\; & \boldsymbol{\alpha}_1\big(\mathbf{w}_0 - (\eta_{(0,1)}\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}_{(0,1)}} + \eta\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}_{(0,2)}})\big) + \\ & \boldsymbol{\alpha}_2\big(\mathbf{w}_0 - (\eta_{(0,1)}\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}_{(0,1)}} + \eta_{(0,2)}\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}_{(0,2)}} + \\ & \eta_{(1,1)}\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}_{(1,1)}} + \eta_{(1,2)}\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}_{(1,2)}})\big).\end{aligned} \tag{8.6}$$

where $\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}_{(i,j)}}$ denotes the gradients of the loss on the bases of the random mini-batch w.r.t. parameter vectors at the i-th epoch and j-th parameter update. Using positive equal ensemble coefficients $\boldsymbol{\alpha}_1 = \boldsymbol{\alpha}_2 = 1/2$ results in the following parameter ensemble ensemble model

$$\begin{aligned}\mathbf{w}_{\text{PPE}} = \mathbf{w}_0 - \Big(\eta_{(0,1)}\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}_{(0,1)}} + \eta_{(0,2)}\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}_{(0,2)}} + \\ \frac{\eta_{(2,2)}\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}_{(1,1)}}}{2} + \frac{\eta_{(2,2)}\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}_{(2,2)}}}{2}\Big).\end{aligned}$$

More generally, using PPE with equal ensemble coefficients $\boldsymbol{\alpha}_{0:j} = \mathbf{0}$ and $\boldsymbol{\alpha}_{j+1:N} = \frac{\mathbf{1}}{\mathbf{N-j}}$ can be rewritten as

$$\mathbf{w}_{\text{PPE}} = \mathbf{w}_j - \Big(\sum_{i=j+1}^{N}\sum_{t}^{T}\frac{\eta_{(i,t)}\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}_{(i,t)}}}{i}\Big). \tag{8.7}$$

Therefore, using such ensemble coefficients (i.e., averaging parameters at each epoch interval) results in deriving such $\mathbf{w}_{\text{PPE}}$ that is more heavily influenced by the parameter vectors obtained at the early stage of the training phase starting from $\mathbf{w}_j$. Consequently, selecting such j-th epoch, after $\mathcal{L}_{\mathcal{D}}(\mathbf{w})$ stagnates, the circling issue around a minima can be alleviated. Yet, using positive equal $\boldsymbol{\alpha}$ arguably may not be optimal for all learning problems, since it is implicitly assumed that $\mathbf{w}_j \approx \mathbf{w}^*$. Given $\boldsymbol{\alpha}$, $\mathbf{w}_{\text{PPE}}$ can be iteratively constructed as shown in Algorithm 2. In Section 8.1.1, we propose two techniques to determine $\boldsymbol{\alpha}$.

Algorithm 2 Polyak Parameter Ensemble

Require: $\boldsymbol{\alpha}, \eta, N, T, \mathcal{L}$
$\mathbf{w}_{\text{PPE}} = \mathbf{0} \leftarrow$ Initialize
$\mathbf{w} \leftarrow$ Initialize
for i = 1, …, N **do**
  for t = 1, …, T **do**
    $\mathcal{B} \leftarrow$ Sample mini-batch
    $\mathcal{L}_{\mathcal{B}} \leftarrow$ Compute mini-batch loss
    $\nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}) \leftarrow$ Compute gradients
    $\mathbf{w} := \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}}(\mathbf{w}) \leftarrow$ Parameter Update
  end for
  $\mathbf{w}_{\text{PPE}}$ += $\boldsymbol{\alpha}_i \mathbf{w} \leftarrow$ Running Average Update
end for

return $\mathbf{w}_{\text{PPE}}$

8.1.1 Determining Ensemble Coefficients

Parameter ensemble coefficients $\boldsymbol{\alpha}$ can be determined in various fashion. For instance, $\boldsymbol{\alpha}$ can be dynamically determined by tracking the trajectory of the validation losses at each epoch. More specifically, initially $\boldsymbol{\alpha}$ are initialized with zeros. As the discrepancy between the validation and training loss exceeds a certain threshold at the end of the j-th epoch, positive equal ensemble coefficients can be used, i.e., $\boldsymbol{\alpha}_{0:j} = \mathbf{0}$ and $\boldsymbol{\alpha}_{j+1:N} = \frac{1}{\text{N}-\text{j}}$. Although this may alleviate possible overfitting, hence improve the generalization performance, assigning equal weights for $N - j$ epochs may not be ideal for all learning problems. Instead, remaining parameter ensemble coefficients $\boldsymbol{\alpha}$ can be determined in an exponentially increasing manner, i.e., $\boldsymbol{\alpha}_{j+1} = \lambda \boldsymbol{\alpha}_j$, where λ is a scalar value denoting the rate of increase. Figure 8.1 visualizes the ensemble coefficients with different growth rates. Using a growth rate of 1.0 implies that $\mathbf{w}_{\text{PPE}}$ is constructed with positive equal coefficients.

8.1.2 KvsSample Training Strategy

During our experiments, we observe that applying the KvsAll training strategy becomes a computational bottleneck as $|\mathcal{E}|$ grows. On the YAGO3-10 dataset, computing a single forward pass $\phi_{\mathbf{w}}(\cdot)$ with $|\mathcal{B}| = 1024$ and $d = 256$ requires $1024 \times 256 \times 123182 \times 32$-bit memory.

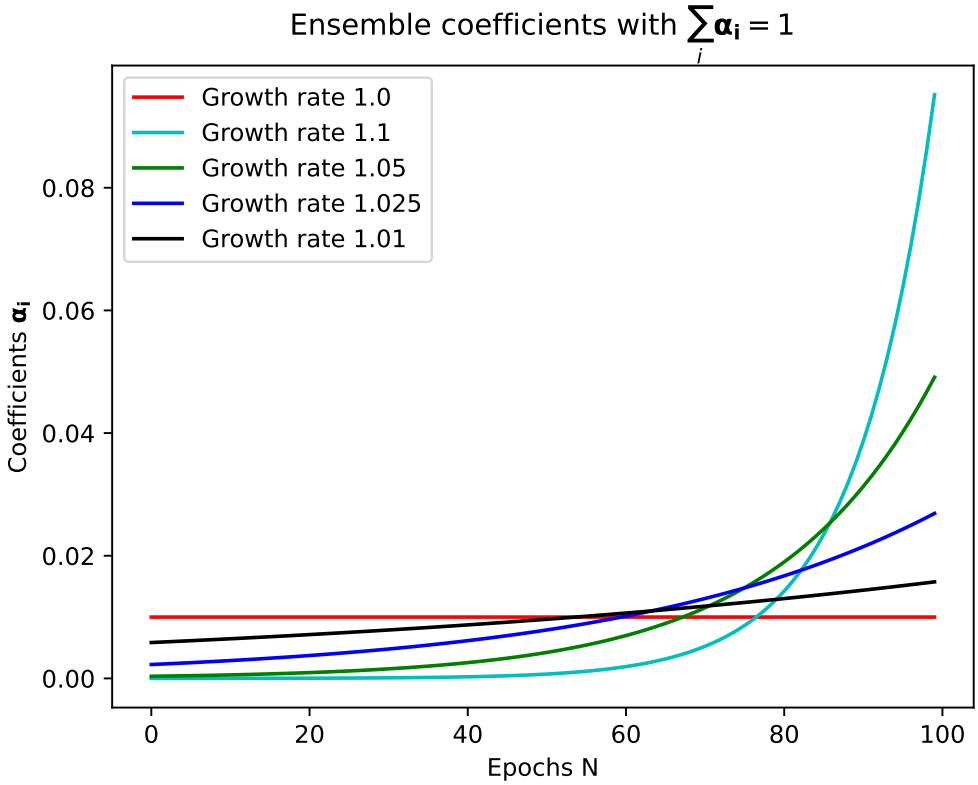


Figure 8.1: Ensemble coefficients with different growth rates.

Although the memory requirement of applying the negative sampling technique is lower than KvsAll, the size of the mini-batch increases with a rate of $k \times m$ and the regularization effect of KvsAll is not utilized Dettmers et al. (2018). Here, we design a new training strategy called the KvsSample training strategy that combines the negative sampling with the KvsAll training strategies. As in KvsAll (see Section 2.4.2), a training data point consists of a unique pair of a head entity and a relation ($\mathbf{x}$=(h, r)) and a binary vector $\mathbf{y} \in [0,1]^{|\mathcal{E}|}$, where $\mathbf{y}_i = 1$ iff $\mathsf{h}, \mathsf{r}, \mathsf{e}_i \in \mathcal{G}$, otherwise 0. Yet, KvsSample stochastically subsamples $\mathbf{y}$. As the negative sample ratio k increases, the size of the mini-batch in KvsSample remains unchanged, while the size of the mini-batch in the negative sampling technique increases with a rate of $k \times m$. Algorithm 3 illustrates the KvsSample training strategy.

Algorithm 3 KvsSample Training strategy

Require: $\mathcal{B}, k$
for i = 1, ..., $|\mathcal{B}|$ **do**
  $(\mathsf{h}, \mathsf{r}), \{e_l, \ldots, e_k\} ::= \mathcal{B}_i \leftarrow$ A data point
  if $|\mathbf{t}| \geq k$ **then**
    $\mathbf{t} := \text{sample}(\{e_l, \ldots, e_k\}, k)$
    $\mathbf{neg} := \text{sample}(\mathcal{E}, k)$
  else
    $\mathbf{t} := \{e_l, \ldots, e_k\}$
    $\mathbf{neg} := \text{sample}(\mathcal{E}, k + (k - |\mathbf{t}|)$
  end if
  $\mathbf{y} := 0^{2k}$
  $\mathbf{y}_{0:k} := 1^{|\mathbf{t}|}$
  $\mathbf{y}_{\text{index}} := [\mathbf{t}, \mathbf{neg}]$
  $\mathcal{B}_i := ((\mathsf{h}, \mathsf{r}), \mathbf{y}_{\text{index}}, \mathbf{y}) \leftarrow$ An updated data point
end for

8.2 Experiments & Results

8.2.1 Experiments

Experimental Setup

All experiments were carried out on a single core of a server running Ubuntu 18.04 with 126 GB RAM with 16 Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz processors.

Datasets

We used the standard benchmark datasets (UMLS, KINSHIP, NELL-995 h25, NELL-995 h50, NELL-995 h100, FB15K-237, YAGO3-10) for the link prediction problem. An overview of the datasets is provided in Table 8.2. KINSHIP describes the 26 different kinship relations of the Alyawarra tribe and the unified medical language system (UMLS) describes 135 medical entities via 49 relations describing (Trouillon and Nickel, 2017). FB15K-237 and YAGO3-10 are subsets of Freebase and YAGO (Dettmers et al., 2018), Never-Ending Language Learning datasets are designed to multi-hop reasoning capabilities released (NELL-995 h25, NELL-995 h50, NELL-995 h100) (Xiong et al., 2017).

We were also interested to evaluate PPE on other benchmark datasets. So, we include Mutagenesis and Carcinogenesis benchmark datasets that are often used in the description logic concept learning problem (Heindorf et al., 2022).

Table 8.2: An overview of datasets in terms of number of entities, number of relations, and node degrees in the train split along with the number of triples in each split of the dataset.

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	$\lvert\mathcal{G}^{\text{Train}}\rvert$	$\lvert\mathcal{G}^{\text{Validation}}\rvert$	$\lvert\mathcal{G}^{\text{Test}}\rvert$
Mutagenesis	14,250	8	55,023	-	-
Carcinogenesis	22,540	9	63,382	-	-
UMLS	135	46	5,216	652	661
KINSHIP	104	25	8,544	1,068	1,074
NELL-995 h100	22,411	43	50,314	3,763	3,746
NELL-995 h50	34,667	86	72,767	5,440	5,393
NELL-995 h25	70,145	172	122,618	9,194	9,187
FB15K-237	14,541	237	272,115	17,535	20,466
YAGO3-10	123,182	37	1,079,040	5,000	5,000

Evaluation

We evaluated the link prediction performance of DistMult, ComplEx, and QMult with and without PPE. To this end, we used the Hits@N and MRR benchmark metrics Ruffinelli et al. (2019); Demir et al. (2021a); Trouillon et al. (2016). We reported the Hits@N and MRR training, validation and test scores on each benchmark dataset for the link prediction problem. By reporting the training and validation scores, we aim to detect possible impacts on the training performance. Since the Mutagenesis and Carcinogenesis datasets do not contain the validation and test splits, we applied 10-fold cross validated results and reported the mean results.

Hyperparameter Optimization

We followed the experimental setup used in Ruffinelli et al. (2019); Demir et al. (2021a). More specifically, we trained DistMult, ComplEx, and QMult KGE models with the following hyperparameter configuration: the number of epochs $N \in \{200, 250\}$, Adam optimizer with $\eta = 0.1$, batch size 1024, layer normalization on embedding vectors and an embedding vector size $d \in \{256, 128, 64\}$, and $\lambda \in \{1.0, 1.1\}$.

Note that $d = 256$ corresponds to 256 real-valued embedding vector size, hence 128 and 64 complex- and quaternion-valued embedding vector sizes respectively. We ensure that all models have the same number of parameters, while exploring various d. Throughout our experiments, we used KvsAll training strategy, unless said otherwise. On all datasets, we used the same the j-th epoch to determine ensemble coefficients $\boldsymbol{\alpha}$ (see Equation (8.7)) We fixed the j-th epoch as 200. Hence, in our experiments, we did not dynamically determined $\boldsymbol{\alpha}$ by tracking the validation loss. By doing this, we ensure that PPE **does not benefit any additional information (e.g., the validation loss) during the training process**.

8.2.2 Results

Overall, our experiments suggest that using PPE consistently improves the link prediction performance of DistMult, ComplEx, and QMult on all datasets. Results of our parameter analysis suggest that as the embedding size d increases, the benefits of using PPE becomes more tangible. Throughout our experiments, we did not detect any runtime overhead of using PPE. Despite all models are trained with the Adam optimizer, PPE seems alleviate the circling behavior around a minima further. Tables 8.3, 8.4, 8.6 and 8.7 report the link prediction results on FB15K-237 and YAGO3-10, NELL-995 h25, NELL-995 h50, NELL-995 h100, UMLS, and KINSHIP benchmark datasets.

Table 8.5 reports the 10-fold cross validated link prediction results on bio-related benchmark datasets that do not have validation and test splits. Table 8.8 reports the link prediction results with different embedding dimensions and different growth rates for ensemble coefficients. Using PPE improves the results across datasets, three models and four metrics.

Overall, results suggest that PPE improves the generalization performance of DistMult, ComplEx, and QMult on the both datasets. As d grows, the benefits of PPE becomes more tangible. For instance, if $d \geq 32$ DistMult with PPE reaches **81 higher Hit@N or MRR out of 96 scores**, while DistMult with PPE performed slightly worse at only 6 out of 96 scores. Using the 1.1 growth rate leads to slight improvement over the 1.1 growth rate.

Tables 8.9 and 8.10 report link prediction performance comparisons between the KvsSample and the negative sampling techniques with different negative sample ratios. As k grows, the size of a single mini-batch increases by the rate of $k \times m$, where m

Table 8.3: Link prediction results on the train, validation and test splits of FB15K-237 and YAGO3-10. Bold results indicate the best results.

	FB15K-237				**YAGO3-10**			
	MRR	@1	@3	@10	MRR	@1	@3	@10
DistMult-train	0.991	0.985	**0.999**	**1.000**	0.980	0.962	**0.998**	**1.000**
With PPE	**0.994**	**0.990**	0.997	**1.000**	**0.981**	**0.963**	**0.998**	0.999
DistMult-val	0.124	0.074	0.132	0.222	0.400	0.337	0.433	0.520
With PPE	**0.138**	**0.082**	**0.149**	**0.249**	**0.446**	**0.384**	**0.481**	**0.558**
DistMult-test	0.122	0.071	0.129	0.223	0.393	0.330	0.425	0.512
With PPE	**0.134**	**0.080**	**0.145**	**0.243**	**0.441**	**0.377**	**0.481**	**0.558**
ComplEx-train	0.995	0.991	0.999	**1.000**	**0.984**	**0.969**	**1.000**	**1.000**
With PPE	**0.996**	**0.993**	**1.000**	**1.000**	**0.984**	0.969	**1.000**	**1.000**
ComplEx-val	0.128	0.075	0.138	0.233	0.408	0.344	0.439	0.530
With PPE	**0.153**	**0.095**	**0.169**	**0.270**	**0.444**	**0.378**	**0.484**	**0.562**
Complex-test	0.126	0.075	0.134	0.229	0.394	0.325	0.431	0.525
With PPE	**0.150**	**0.094**	**0.165**	**0.264**	**0.433**	**0.366**	**0.473**	**0.554**
QMult-train	0.989	0.981	0.997	0.999	0.821	0.790	0.841	0.877
With PPE	**0.995**	**0.990**	**1.000**	**1.000**	**0.828**	**0.792**	**0.852**	**0.881**
QMult-val	0.141	0.083	0.151	0.258	0.306	0.241	0.338	0.427
With PPE	**0.172**	**0.108**	**0.188**	**0.302**	**0.341**	**0.283**	**0.367**	**0.346**
QMult-test	0.138	0.082	0.146	0.253	0.300	0.232	0.334	0.424
With PPE	**0.167**	**0.102**	**0.183**	**0.298**	**0.339**	**0.282**	**0.365**	**0.444**

denotes the number of positive training data points (see Section 2.4.2), whereas the size of the mini-batch in KvsSample does remains unchanged. In KvsSample, k only influences the size of the sparse multi-label vectors. Important to note that, since the full computation runtimes are reported, the overhead of iterating over all triples to store unique head entity and relation pairs is visible at $k = 2$. Moreover, since there are few unique entity relation pairs occur more than $k = 2$ times on an input KG, setting k very low with KvsAll often leads to inferior performance on both benchmark dataset. Results suggest that as the negative sample ratio k increases, the rate of increase in the total computation time is larger for the negative sampling training technique than the KvsSample technique.

Table 8.4: Link prediction results on the train, validation and test splits of UMLS and KINSHIP benchmark datasets. Bold results indicate the best results.

	UMLS				**KINSHIP**			
	MRR	@1	@3	@10	MRR	@1	@3	@10
DistMult-train	0.992	0.987	0.997	**1.000**	0.847	0.781	0.893	0.977
With PPE	**0.999**	**0.998**	**0.999**	**1.000**	**0.865**	**0.806**	**0.906**	**0.981**
DistMult-val	0.458	0.325	0.500	0.753	0.399	0.256	0.432	0.741
With PPE	**0.499**	**0.376**	**0.528**	**0.778**	**0.426**	**0.288**	**0.455**	**0.760**
DistMult-test	0.450	0.321	0.491	0.755	0.404	0.260	0.442	0.755
With PPE	**0.493**	**0.372**	**0.526**	**0.778**	**0.433**	**0.290**	**0.470**	**0.782**
ComplEx-train	0.998	0.997	**1.000**	**1.000**	0.993	0.989	0.998	**1.000**
With PPE	**1.000**	**1.000**	**1.000**	**1.000**	**0.996**	**0.993**	**0.999**	**1.000**
ComplEx-val	0.442	0.285	0.521	0.766	0.521	0.378	0.595	0.829
With PPE	**0.491**	**0.350**	**0.550**	**0.783**	**0.599**	**0.463**	**0.677**	**0.861**
ComplEx-test	0.444	0.287	0.528	0.773	0.533	0.388	0.614	0.821
With PPE	**0.502**	**0.361**	**0.573**	**0.787**	**0.603**	**0.479**	**0.675**	**0.842**
QMult-train	0.998	0.998	0.999	0.999	0.990	0.983	0.996	**0.999**
With PPE	**1.000**	**1.000**	**1.000**	**1.000**	**0.995**	**0.992**	**0.998**	**0.999**
QMult-val	0.445	0.280	0.524	0.791	0.500	0.352	0.571	0.805
With PPE	**0.485**	**0.326**	**0.578**	**0.803**	**0.598**	**0.468**	**0.675**	**0.852**
QMult-test	0.426	0.272	0.498	0.757	0.502	0.355	0.580	0.801
With PPE	**0.480**	**0.334**	**0.555**	**0.786**	**0.591**	**0.467**	**0.668**	**0.838**

Discussion Our results indicate that constructing a parameter ensemble model by maintaining a weighted average of parameters obtained at each epoch interval improves the generalization performance across datasets and knowledge graph embedding models. We show that with our formulation, weights/parameter ensemble coefficients can be determined in various forms, e.g., dynamically by tracking the validation loss or choosing an exponential function over coefficients. Overall, results suggest that using exponentially increasing ensemble coefficients consistently improves the generalization results. This may suggest that although Adam dynamically adapts the learning rate w.r.t. the gradients, our weighted parameter averaging approach (PPE) accelerates the converge on a minima during training. Yet, our parameter analysis show that the benefits of applying PPE dissipates if the embedding vector is very low (e.g. $d < 16$).

Table 8.5: 10-fold cross validated link prediction results on Mutagenesis and Carcinogenesis datasets. Bold results indicate the best results.

	Mutagenesis				**Carcinogenesis**			
	MRR	@1	@3	@10	MRR	@1	@3	@10
DistMult	0.150	0.121	0.151	0.204	0.045	0.025	0.047	0.085
With PPE	**0.186**	**0.156**	**0.192**	**0.225**	**0.059**	**0.034**	**0.063**	**0.106**
ComplEx	0.143	0.109	0.148	0.200	0.054	**0.027**	0.056	0.110
With PPE	**0.203**	**0.158**	**0.220**	**0.286**	**0.056**	**0.027**	**0.059**	**0.117**
QMult	0.136	0.104	0.137	0.190	**0.033**	0.014	**0.029**	**0.065**
With PPE	**0.195**	**0.154**	**0.203**	**0.266**	**0.033**	**0.015**	**0.029**	**0.065**

Our results also suggest that the KvsSample training strategy finds a middle ground between the negative sampling technique and the KvsAll training strategy. As $|\mathcal{E}|$ grows, the KvsAll training strategy can be more suitable than using the negative sampling strategy in terms of runtimes as well as effectiveness in the link prediction problem.

Trade-offs & Possible Improvements: Tracking the validation loss and determining the ensemble coefficients accordingly may further improve the generalization performance in the link prediction problem.

Table 8.6: Link prediction results on the train, validation and test splits of NELL-995 h25 and NELL-995 h50 benchmark datasets. Bold results indicate the best results.

	h25				**h50**			
	MRR	@1	@3	@10	MRR	@1	@3	@10
DistMult-train	**0.995**	0.991	0.998	**1.000**	**0.955**	**0.934**	**0.974**	**0.990**
With PPE	**0.995**	**0.992**	**0.999**	**1.000**	0.895	0.863	0.921	0.951
DistMult-val	0.151	0.107	0.164	0.235	0.162	0.114	0.178	**0.258**
With PPE	**0.159**	**0.116**	**0.172**	**0.240**	**0.164**	**0.116**	**0.184**	0.257
DistMult-test	0.154	0.111	0.168	0.238	0.164	0.116	0.116	0.257
With PPE	**0.162**	**0.119**	**0.177**	**0.245**	**0.166**	**0.119**	**0.184**	**0.258**
ComplEx-train	**1.000**	**1.000**	**1.000**	**1.000**	0.991	0.986	0.995	0.996
With PPE	**1.000**	**1.000**	**1.000**	**1.000**	**0.995**	**0.991**	**0.999**	**1.000**
ComplEx-val	**0.105**	**0.069**	**0.110**	0.175	0.079	0.048	0.082	0.143
With PPE	**0.105**	**0.069**	**0.110**	**0.176**	**0.089**	**0.054**	**0.094**	**0.160**
Complex-test	**0.106**	**0.071**	**0.110**	0.175	0.080	0.049	0.085	0.141
With PPE	0.105	0.069	**0.110**	**0.178**	**0.093**	**0.058**	**0.097**	**0.160**
QMult-train	0.977	0.967	0.986	0.994	0.917	0.890	0.934	0.962
With PPE	**1.000**	**0.999**	**1.000**	**1.000**	**0.924**	**0.902**	**0.937**	**0.966**
QMult-val	0.084	0.055	0.090	0.140	0.102	0.058	0.115	0.191
With PPE	**0.090**	**0.059**	**0.096**	**0.150**	**0.114**	**0.068**	**0.127**	**0.206**
QMult-test	0.081	0.052	0.086	0.138	0.105	0.061	0.114	0.191
With PPE	**0.086**	**0.055**	**0.090**	**0.146**	**0.116**	**0.070**	**0.126**	**0.211**

Table 8.7: Link prediction results on the train, validation and test splits of NELL-995 h100 benchmark datasets. Bold results indicate the best results.

	h100			
	MRR	@1	@3	@10
DistMult-train	0.962	0.940	0.982	0.992
With PPE	**0.970**	**0.952**	**0.987**	**0.995**
DistMult-val	0.158	0.107	0.171	0.265
With PPE	**0.173**	**0.119**	**0.186**	**0.282**
DistMult-test	0.151	0.101	0.165	0.254
With PPE	**0.169**	**0.118**	**0.186**	**0.275**
ComplEx-train	0.943	**0.914**	**0.970**	**0.990**
With PPE	**0.916**	0.879	0.942	0.979
ComplEx-val	0.104	0.056	0.118	0.198
With PPE	**0.111**	**0.063**	**0.123**	**0.209**
Complex-test	0.097	0.052	0.105	0.191
With PPE	**0.107**	**0.060**	**0.116**	**0.107**
QMult-train	0.704	0.634	0.751	0.826
With PPE	**0.913**	**0.868**	**0.957**	**0.979**
QMult-val	0.087	0.050	0.096	0.159
With PPE	**0.104**	**0.063**	**0.117**	**0.189**
QMult-test	0.083	0.048	0.094	0.152
With PPE	**0.101**	**0.063**	**0.114**	**0.179**

Table 8.8: Link prediction results of DistMult with different embedding dimensions d on the train, validation and test splits of UMLS and KINSHIP benchmark datasets. PPE† denotes applying PPE with $\lambda = 1.1$. Bold results indicate the best results.

	d	**UMLS**				**KINSHIP**			
		MRR	@1	@3	@10	MRR	@1	@3	@10
DistMult-test	2	0.309	0.225	**0.321**	**0.470**	**0.061**	**0.010**	**0.039**	**0.123**
With PPE		**0.310**	**0.226**	0.318	0.469	0.054	**0.010**	0.033	0.106
With PPE†		0.309	0.225	0.316	0.469	0.055	**0.010**	0.031	0.107
DistMult-test	4	0.617	0.486	0.711	0.834	**0.105**	**0.038**	**0.083**	0.218
With PPE		**0.627**	**0.490**	**0.724**	0.837	0.099	0.028	**0.083**	**0.220**
With PPE†		0.626	0.489	0.723	**0.840**	0.102	0.033	0.082	**0.221**
DistMult-test	8	**0.775**	**0.663**	**0.869**	**0.939**	0.518	0.351	0.602	0.886
With PPE		0.763	0.643	0.865	0.935	**0.533**	**0.373**	**0.616**	**0.883**
With PPE†		0.764	0.644	0.865	0.935	**0.534**	**0.376**	0.614	**0.883**
DistMult-test	16	0.862	0.781	**0.936**	0.981	0.665	0.520	0.771	**0.938**
With PPE		0.863	0.785	0.932	0.982	0.669	0.523	0.779	0.932
With PPE†		**0.865**	**0.788**	0.934	**0.983**	**0.676**	**0.534**	**0.783**	0.934
DistMult-test	32	0.812	0.722	0.883	0.970	0.704	0.569	0.796	0.958
With PPE		**0.839**	**0.760**	0.896	**0.974**	0.713	0.584	0.801	0.957
With PPE†		0.837	0.759	**0.898**	**0.974**	**0.714**	**0.585**	**0.803**	**0.959**
DistMult-test	64	0.673	0.539	0.760	0.940	0.602	0.437	0.717	0.924
With PPE		0.680	0.551	0.759	0.939	**0.632**	**0.483**	**0.724**	**0.932**
With PPE†		**0.686**	**0.554**	**0.777**	**0.946**	0.629	0.475	**0.724**	**0.932**
DistMult-test	128	0.665	0.527	**0.767**	**0.934**	0.481	0.307	0.561	0.884
With PPE		0.667	**0.532**	0.766	0.933	0.507	0.338	0.591	0.889
With PPE†		**0.669**	0.531	0.766	0.933	**0.520**	**0.351**	**0.600**	**0.890**
DistMult-test	256	0.648	0.503	0.746	0.932	0.444	0.270	0.518	0.851
With PPE		0.651	0.506	0.746	0.933	0.464	0.291	0.539	0.861
With PPE†		**0.659**	**0.511**	**0.765**	**0.936**	**0.475**	**0.309**	**0.541**	**0.868**

Table 8.9: Link prediction performance comparison with the KvSample and negative sampling training strategies. k and RT denotes the number of negative example per triple and the total runtime in seconds, respectively. Bold results indicate the best results.

	UMLS					
	k	RT	MRR	@1	@3	@10
NegSample-test	2	**24.2**	0.797	0.698	0.873	0.976
KvsSample-test		30.8	**0.864**	**0.796**	**0.917**	**0.979**
NegSample-test	4	31.1	0.753	0.646	0.828	0.954
KvsSample-test		**29.1**	**0.857**	**0.798**	**0.892**	**0.973**
NegSample-test	8	52.1	0.805	0.718	0.866	0.970
KvsSample-test		**31.0**	0.811	0.734	0.852	0.964
NegSample-test	16	86.6	0.807	0.718	0.865	**0.974**
KvsSample-test		**37.3**	**0.833**	**0.762**	**0.880**	**0.974**
NegSample-test	32	267.8	0.789	0.696	0.856	0.961
KvsSample-test		**52.9**	**0.809**	**0.719**	**0.877**	**0.970**
NegSample-test	64	540.7	0.789	0.697	0.852	0.967
KvsSample-test		**110.8**	**0.823**	**0.740**	**0.879**	**0.976**
NegSample-test	128	976.3	**0.838**	**0.765**	0.897	0.976
KvsSample-test		**142.8**	**0.838**	**0.765**	**0.899**	**0.980**
NegSample-test	256	1857.9	0.843	0.772	0.894	0.979
KvsSample-test		**279.9**	**0.867**	**0.812**	**0.903**	**0.980**

Table 8.10: Link prediction performance comparison with the KvSample and negative sampling training strategies. Results are obtained by training DistMult with a selected training strategy. k and RT denotes the number of negative example per triple and the total runtime in seconds, respectively. Bold results indicate the best results.

	KINSHIP					
	k	RT	MRR	@1	@3	@10
NegSample-test	2	**36.4**	**0.559**	**0.397**	**0.647**	**0.904**
KvsSample-test		63.6	0.519	0.351	0.605	0.887
NegSample-test	4	**52.8**	0.505	0.341	0.585	0.877
KvsSample-test		62.7	**0.527**	**0.359**	**0.624**	**0.895**
NegSample-test	8	80.4	0.507	0.348	0.574	0.870
KvsSample-test		**58.5**	**0.555**	**0.385**	**0.649**	**0.915**
NegSample-test	16	83.6	0.502	0.342	0.572	0.877
KvsSample-test		**56.9**	**0.534**	**0.363**	**0.620**	**0.910**
NegSample-test	32	403.4	0.493	0.324	0.571	0.881
KvsSample-test		**96.9**	**0.514**	**0.350**	**0.593**	**0.889**
NegSample-test	64	788.1	0.481	0.312	0.556	0.861
KvsSample-test		**158.2**	**0.539**	**0.376**	**0.622**	**0.901**
NegSample-test	128	1558.0	0.485	0.323	0.553	0.865
KvsSample-test		**276.1**	**0.528**	**0.371**	**0.596**	**0.889**
NegSample-test	256	3059.5	0.507	0.343	0.583	**0.892**
KvsSample-test		**555.8**	**0.536**	**0.381**	**0.600**	0.889

9

Learning Permutation-Invariant Embeddings for Description Logic Concepts

Preamble: This chapter is based on Demir and Ngonga Ngomo (2023).

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and Axel-Cyrille Ngonga Ngomo revised it. The code is available at https://github.com/dice-group/Nero.

Research Questions: In this work, we are concerned with the following research question:

1. **RQ3.** Can we design a model to learn embeddings for expressive DLs?

9.1 Methodology

In this section, we introduce NeRo. The goal in the Concept Learning (CL) problem is to find a DL concept $\mathsf{H} \in C$ ideally maximizing Equation (2.67), given a set of positive examples E^+ and a set of negative examples E^-. Given E^+ and E^-, NeRo predicts F_1 scores of pre-selected DL concepts as shown in Figure 9.1.

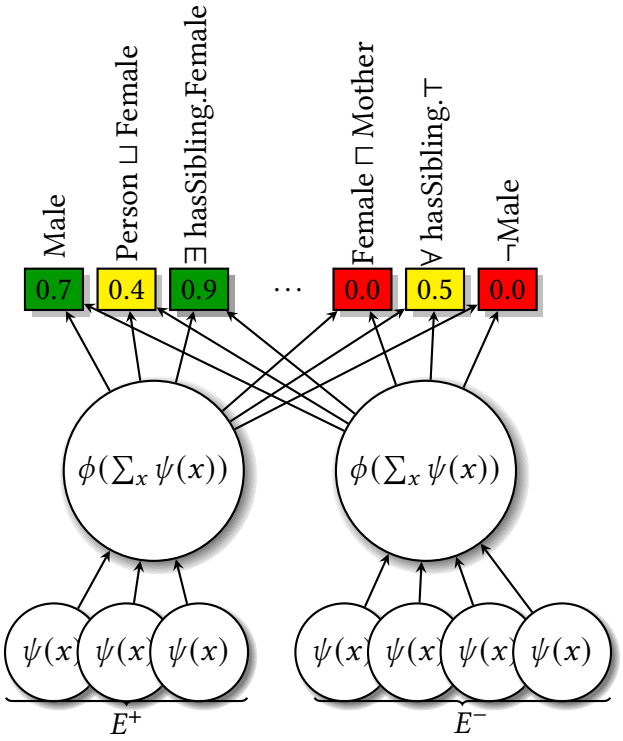


Figure 9.1: A visualization of NeRo. Boxes and values denote the pre-selected unique DL concepts and their predicted F_1 scores, respectively.

By ranking pre-selected DL concepts in descending order of predicted scores, a goal concept can be found by only exploring few top-ranked concepts. Importantly, top-ranked concepts can be used to initialize the standard search procedure of state-of-the-art models, if a goal concept is not found. By this, a state-of-the-art CL model is endowed with the capability of starting the search in more advantageous states, instead of starting it in the most general concept $\top$. Our experiments on 5 benchmark datasets with 770 learning problems indicate that NeRo **significantly** (p-value $< 1\%$) outperforms the state-of-the-art models in standard metrics such as F_1 score, the number

of explored concepts, and the total runtime. Importantly, equipping NeRo with a state-of-the-art model (e.g. CELOE) further improves F_1 scores on benchmark datasets with a low runtime cost. The results of Wilcoxon signed rank tests confirm that the superior performance of NeRo is significant. In Table 9.1, we summarized the symbols used in this chapter.

Table 9.1: An overview of our notation used in Chapter 9.

Notation	Description
$\mathcal{K}$	Knowledge base
$\mathcal{L}$	A description logic
C	All description logic concepts in $\mathcal{L}$
$\mathcal{R}(\cdot)$	Instance retrieval function
N_I	A set of individuals on $\mathcal{K}$
N_C	A set of concepts on $\mathcal{K}$
N_R	A set of roles on $\mathcal{K}$
E^+, E^-	Positive and negative example sets
$\sigma(\cdot)$	Logistic sigmoid function
$f(\cdot)$	A deep set neural network
$\pi(\cdot), \psi(\cdot)$	Continuous functions used in $f(\cdot)$
$\mathcal{T}$	Pre-selected target class expressions
$\ell(\cdot, \cdot)$	Binary cross entropy loss function
$\mathcal{D}$	Training dataset

Approach. Equation (2.67) indicates that $F_1(\cdot)$ is invariant to the order of individuals in E^+, E^-, and $\mathcal{R}(\cdot)$. Previously, Zaheer et al. (2017) have proven that all functions being invariant to the order in inputs can be decomposed into

$$f(\mathbf{x}) = \phi\Big(\sum_{x \in \mathbf{x}} \psi(x)\Big), \tag{9.1}$$

where $\mathbf{x} = \{x_1, \ldots, x_m\} \in 2^X$ and $\phi(\cdot)$ and $\psi(\cdot)$ denote a set of input and two parameterized continuous functions, respectively. A permutation-invariant neural network defined via Equation (9.1) still abides by the universal approximation theorem (Zaheer et al., 2017).

We conjecture that such neural network can learn permutation-invariant embeddings for sets of individuals (e.g., E^+ and E^-) tailored towards predicting F_1 scores of pre-selected concepts.

Through accurately predicting F_1 scores of pre-selected DL concepts, possible goal concepts from pre-selected concepts can be detected without using $F_1(\cdot)$ and $\mathcal{R}(\cdot)$. With these considerations, we define NeRo as follows

$$\text{NeRo}(E^+, E^-) = \sigma\Big(\phi\Big(\sum_{x \in E^+} \psi(x)\Big) - \phi\Big(\sum_{x \in E^-} \psi(x)\Big)\Big), \tag{9.2}$$

where $\psi(\cdot) : N_I \to \mathbb{R}^m$ and $\phi : \mathbb{R}^m \to [0,1]^{|\mathcal{T}|}$ denote an embedding look-up operation and an affine transformation, respectively. $\mathcal{T}$ represents the pre-selected DL concepts. The result of the translation operation denoted with $\mathbf{z} \in \mathbb{R}^m$ is normalized via the logistic sigmoid function $\sigma(\mathbf{z}) = \frac{1}{1+\exp(-\mathbf{z})}$. Hence, $\text{NeRo} : 2^{N_I} \times 2^{N_I} \mapsto [0,1]^{|\mathcal{T}|}$ can be seen as a mapping from two sets of individuals to $|\mathcal{T}|$ unit intervals. NeRo can be seen as a multi-task learning approach that leverages the similarity between multi-tasks, where a task in our case corresponds to accurately predicting the F_1 score of a pre-selected DL concept (Caruana, 1998).

Selecting Description Logic Concepts: The importance of learning representations tailored towards *related* tasks has been well investigated (Goller and Kuchler, 1996; Caruana, 1998). Motivated by this, we elucidate the process of selecting DL concepts in Algorithm 4. We select such concepts that their canonical interpretations do not fully overlap (see the 4.th line). As shown therein, NeRo can be trained on knowledge base defined over any DLs provided that $\mathcal{R}(\cdot)$ and $\rho(\cdot)$ are given.

Training Process: Let $\mathcal{D} = \{(E_i^+, E_i^-, \mathbf{y}_i)\}_{i=1}^N$ represent a training dataset, where a data point $(E^+, E^-, \mathbf{y})$ is obtained in four consecutive steps:

1. Sample $\mathtt{C}$ from $\mathcal{T}$ uniformly at random,
2. Sample k individuals $E^+ \subset \mathcal{R}(\mathtt{C})$ uniformly at random,
3. Sample k individuals $E^- \subset N_I \setminus E^+$ uniformly at random, and
4. Compute F_1 scores $\mathbf{y}$ via Equation (2.67) w.r.t. E^+, E^-, for $\mathcal{T}$.

Algorithm 4 Constructing target DL concepts

Input: $\mathcal{R}(\cdot)$, $\rho(\cdot)$, d, maxlength **Output**: $\mathcal{T}$

$\mathcal{T} := \{\mathsf{C} \mid \mathsf{C} \in \rho(\top) \wedge |\mathsf{C}| \leq \text{ maxlength } \wedge 0 < |\mathcal{R}(\mathsf{C})|\}$
for each $\mathsf{A} \in \mathcal{T}$ **do**
  for each $\mathsf{B} \in \mathcal{T}$ **do**
    if $\mathcal{R}(\mathsf{A}) \neq \mathcal{R}(\mathsf{B})$ **then**
      for each $\mathsf{X} \in \{\mathsf{A} \sqcap \mathsf{B}, \mathsf{A} \sqcup \mathsf{B}\}$ **do**
        if $|\mathcal{R}(\mathsf{X})| > 0 \wedge \mathcal{R}(\mathsf{X}) \notin \{\mathcal{R}(\mathsf{E}) \mid \mathsf{E} \in \mathcal{T}\}$ **then**
          Add X to $\mathcal{T}$.
        end if
        if $|\mathcal{T}| = d$ **then**
          return $\mathcal{T}$
        end if
      end for
    end if
  end for
end for
if $|\mathcal{T}| < d$ **then**
  Go to the step (2).
end if

For a given $(E^+, E^-, \mathbf{y})$ and predictions $\hat{\mathbf{y}} := \textsc{NeRo}(E^+, E^-)$, an incurred binary cross entropy loss. Important to note that after training process, permutation-invariant embeddings of any $\mathcal{ALC}$ DL concepts can be readily obtained omitting the translation operation in NeRo. In Figure 9.2, we visualize a 2D PCA visualization of few concepts on the Family KB.

9.2 Experiments & Results

9.2.1 Experiments

Experimental Setup

We based our experimental setup on Bühmann et al. (2016); Lehmann et al. (2011); Bühmann et al. (2018) and used learning problems provided therein. To perform extensive comparisons between models, we also generated additional learning problems by randomly sampling E^+ and E^-. We ensured that none of the learning problems used in our evaluation has been used in the unsupervised training phase. In our experiments, we evaluated all models in $\mathcal{ALC}$ for CEL on the same hardware.

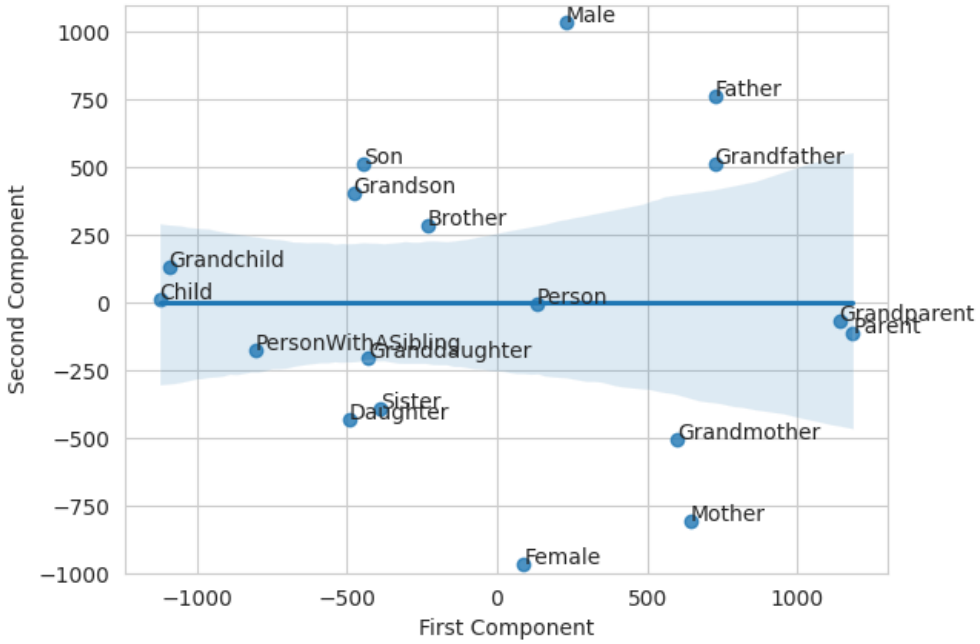


Figure 9.2: Regression on 2D PCA Visualization of all 1-length expressions on the Family benchmark knowledge base. Blue shaded are represents confidence interval.

All experiments were conducted with Ubuntu 18.04 with 16 GB RAM and an Intel(R) Core(TM) i5-7300U CPU v4 @ 2.60GHz.

Datasets

An overview of the benchmark datasets is provided in Table 12.3.

Evaluation

In our experiments, we evaluated all models in $\mathcal{ALC}$ for CL. We evaluated models via the F_1 score, the runtime and number of explored concepts. The F_1 score is used to measure the quality of the concepts found w.r.t. positive and negative examples, while the runtime and the number of explored concepts are to measure the efficiency. We measured the full computation time including the time spent prepossessing time of the input data and tackling the learning problem.

Table 9.2: An overview of class expression learning benchmark datasets.

Dataset	$\lvert N_I \rvert$	$\lvert N_C \rvert$	$\lvert N_R \rvert$
Family	202	18	4
Carcinogenesis	22372	142	21
Mutagenesis	14145	86	11
Biopax	323	28	49
Lymphography	148	49	1

We used two standard stopping criteria for state-of-the-art models. We set the maximum runtime to 10 seconds, although models often reach good solutions within 1.5 seconds (Lehmann and Hitzler, 2010). The models are configured to terminate as soon as they found a goal concept. We only considered top-100 ranked concepts to evaluate NeRo.

Hyperparameter Optimization

During training, we set $|\mathcal{T}| = 1000$, $N = 50$, $d = 100$, the batch size to 512, and used Adam optimizer with 0.01 learning rate.

9.2.2 Results

Results with Benchmark Learning Problems: Table 9.3 reports the concept learning results with benchmark learning problems. Table 9.3 suggests that equipping NeRo with the standard search procedure improves the state-of-the-art performance in terms of F_1 scores even further with a small cost of runtimes. CELOE and ELTL require at least **14.7**× more time than NeRo to find accurate concepts on Family. This stems from the fact that NeRo explores on average only 21 concepts, whereas CELOE explored 1429. On Mutagenesis and Carcinogenesis, NeRo finds more accurate concepts, while exploring less, hence, achieving better runtime performance. Runtime gains stem from the fact that NeRo explores at least **2.3**× fewer concepts.

Important to note we did not use parallelism in NeRo and we reload parameters of NeRo for each single learning problem. To conduct more extensive evaluation, we generated total 750 random learning problems on five benchmark datasets. Since

Table 9.3: Results on benchmark learning problems. F_1, T, and Exp. denote F_1 score, total runtime in seconds, and the number of explored concepts, respectively. NeRo† denotes equipping NeRo with CELOE. ELTL does not report the Exp.

Dataset	NeRo†			NeRo			CELOE			ELTL	
	F_1	T	Exp.	F_1	T	Exp.	F_1	T	Exp.	F_1	T
Family	**0.987**	0.83	26	0.984	**0.68**	**21**	0.980	4.65	1429	0.964	4.12
Mutagenesis	**0.714**	17.30	200	0.704	**13.18**	**100**	0.704	23.05	516	0.704	21.04
Carcinogenesis	**0.725**	32.23	200	0.720	**26.26**	**100**	0.714	37.18	230	0.719	36.29

Lymphography and Biopax datasets do not contain any learning problems, they are not included in Table 9.3.

Results with Random Learning Problems: Table 9.4 reports the concept learning results with random learning problems. Table 9.4 suggests that CELOE explores at least **3.19**× more concepts than NeRo. Importantly, NeRo finds on-par or more accurate concepts, while exploring less. Here, we load the parameters of NeRo only once per dataset and are used to tackle learning problems sequentially. This resulted in reducing the total computation time of NeRo by $\mathbf{3-6}\times$ on Family, Mutagenesis and Carcinogenesis benchmark datasets. Although NeRo can tackle learning problems in parallel (e.g., through multiprocessing), we did not use any parallelism, since CELOE and ELTL do not abide by parallelism Bühmann et al. (2018). Loading the learning problems in a standard mini-batch fashion and using multi-GPUs may further improve the runtimes of NeRo. These results suggest that NeRo can be more suitable than CELOE and ELTL on applications requiring low latency.

Results with Limited Exploration: Table 9.5 reports concept learning results with limited exploration on five benchmark datasets. Table 9.5 suggests that NeRo-10 often outperforms CELOE and ELTL (see Table 9.4) in all metrics even when exploring solely 10 top-ranked concepts.

Significance Testing: To validate the significance of our results, we performed Wilcoxon signed-rank tests (one and two-sided) on F_1 scores, runtimes and the number of explored concepts. Our null hypothesis was that the performances of NeRo and CELOE come from the same distribution. We were able to reject the null hypothesis

Table 9.4: Random learning problems with different sizes per benchmark dataset. Each row reports the mean and standard deviations attained in 50 learning problems. $|E|$ denotes $|E^+| + |E^-|$.

Dataset	$\|E\|$	NeRo			CELOE			ELTL	
		F_1	T	Exp.	F_1	T	Exp.	F_1	T
Family	10	**0.913 ± 0.06**	**0.16 ± 0.51**	**74 ± 43**	0.903 ± 0.06	11.61 ± 3.58	5581 ± 2375	0.718 ± 0.01	4.45 ± 2.84
	20	**0.807 ± 0.04**	**0.16 ± 0.49**	**100 ± 00**	0.795 ± 0.05	13.28 ± 1.47	7586 ± 645	0.678 ± 0.02	3.59 ± 1.27
	30	**0.775 ± 0.03**	**0.15 ± 0.41**	**100 ± 00**	0.760 ± 0.03	13.24 ± 1.42	7671 ± 575	0.672 ± 0.01	3.46 ± 1.59
Lymphography	10	**0.968 ± 0.07**	**0.12 ± 0.43**	**75 ± 41**	0.968 ± 0.07	6.63 ± 4.29	5546 ± 5169	0.733 ± 0.09	3.07 ± 0.30
	20	**0.828 ± 0.04**	**0.13 ± 0.40**	**100 ± 00**	0.826 ± 0.05	13.01 ± 1.23	11910 ± 1813	0.678 ± 0.02	3.08 ± 0.50
	30	**0.780 ± 0.04**	**0.13 ± 0.01**	**100 ± 00**	0.780 ± 0.04	13.02 ± 1.69	13138 ± 2601	0.672 ± 0.01	3.09 ± 0.72
Biopax	10	**0.859 ± 0.08**	**0.19 ± 0.71**	**86 ± 34**	0.806 ± 0.07	13.26 ± 1.94	4752 ± 2153	0.685 ± 0.06	3.71 ± 0.10
	20	**0.793 ± 0.05**	**0.19 ± 0.52**	**100 ± 00**	0.746 ± 0.04	13.63 ± 0.10	4151 ± 748	0.668 ± 0.06	3.72 ± 0.10
	30	**0.749 ± 0.03**	**0.18 ± 0.52**	**100 ± 00**	0.718 ± 0.02	13.91 ± 0.44	3843 ± 963	0.668 ± 0.06	3.90 ± 0.22
Mutagenesis	10	**777 ± 0.05**	**3.47 ± 1.61**	**100 ± 00**	0.753 ± 0.06	20.27 ± 1.39	546 ± 613	0.670 ± 0.02	10.29 ± 0.40
	20	**0.746 ± 0.05**	**3.09 ± 1.75**	**100 ± 00**	0.712 ± 0.02	20.38 ± 1.30	430 ± 28	0.667 ± 0.00	10.73 ± 1.10
	30	**0.721 ± 0.03**	**2.89 ± 1.60**	**100 ± 00**	0.700 ± 0.02	20.39 ± 1.06	429 ± 38	0.667 ± 0.00	11.74 ± 0.97
Carcinogenesis	10	**0.768 ± 0.06**	**5.39 ± 2.98**	**98 ± 14**	0.764 ± 0.06	29.90 ± 1.02	401 ± 125	0.673 ± 0.05	19.99 ± 0.67
	20	**0.722 ± 0.03**	**5.40 ± 1.87**	**100 ± 00**	0.713 ± 0.02	30.30 ± 0.19	318 ± 152	0.667 ± 0.00	20.00 ± 1.11
	30	**0.704 ± 0.05**	**4.70 ± 2.78**	**100 ± 00**	0.697 ± 0.02	29.99 ± 0.58	319 ± 43	0.667 ± 0.00	20.38 ± 0.85

Table 9.5: Performance comparison with different number of explored concepts. Each row reports the mean and standard deviations attained in 50 learning problems.

Dataset	$\|E\|$	NeRo-1		NeRo-10		NeRo-1000	
		F_1	T	F_1	T	F_1	T
Family	10	0.906 ± 0.07	**0.08 ± 0.06**	0.910 ± 0.05	0.09 ± 0.06	**0.916 ± 0.06**	0.81 ± 0.50
	20	0.793 ± 0.05	**0.08 ± 0.05**	0.806 ± 0.04	0.09 ± 0.05	**0.807 ± 0.04**	1.17 ± 0.50
	30	0.742 ± 0.05	**0.08 ± 0.05**	0.773 ± 0.03	0.09 ± 0.05	**0.775 ± 0.03**	1.15 ± 0.50
Lymphography	10	0.882 ± 0.07	**0.08 ± 0.06**	0.905 ± 0.05	0.08 ± 0.06	**0.916 ± 0.06**	0.77 ± 0.50
	20	0.793 ± 0.05	**0.07 ± 0.05**	0.827 ± 0.04	0.08 ± 0.05	**0.828 ± 0.04**	1.03 ± 0.50
	30	0.738 ± 0.05	**0.07 ± 0.06**	0.777 ± 0.04	0.08 ± 0.05	**0.780 ± 0.03**	1.00 ± 0.60
Biopax	10	0.853 ± 0.08	**0.09 ± 0.06**	0.856 ± 0.05	0.97 ± 0.59	**0.868 ± 0.08**	1.31 ± 0.80
	20	0.779 ± 0.05	**0.09 ± 0.06**	0.791 ± 0.04	0.10 ± 0.62	**0.793 ± 0.04**	1.35 ± 0.60
	30	0.708 ± 0.07	**0.09 ± 0.06**	0.742 ± 0.04	0.10 ± 0.63	**0.749 ± 0.03**	1.39 ± 0.60
Mutagenesis	10	0.733 ± 0.07	**0.32 ± 2.03**	0.785 ± 0.06	0.57 ± 1.81	**0.803 ± 0.06**	36.98 ± 5.81
	20	0.689 ± 0.08	**0.31 ± 1.99**	0.734 ± 0.05	0.52 ± 1.82	**0.751 ± 0.04**	34.29 ± 5.91
	30	0.673 ± 0.08	**0.31 ± 2.03**	0.712 ± 0.04	0.49 ± 1.77	**0.728 ± 0.03**	32.88 ± 6.13
Carcinogenesis	10	0.717 ± 0.09	**0.41 ± 2.53**	0.740 ± 0.09	0.89 ± 2.89	**0.783 ± 0.02**	56.941 ± 9.55
	20	0.680 ± 0.06	**0.40 ± 2.49**	0.707 ± 0.05	0.82 ± 2.45	**0.731 ± 0.02**	57.205 ± 5.08
	30	0.610 ± 0.11	**0.41 ± 2.83**	0.671 ± 0.06	0.77 ± 2.66	**0.716 ± 0.02**	52.872 ± 7.58

with a p-value $< 1\%$ across all the datasets, hence, the superior performance of NeRo is statistically significant.

9.2.3 Discussion

Our results uphold our hypothesis: F_1 scores of DL concepts can be accurately predicted by means of learning permutation-invariant embeddings for sets of individuals.

Through considering top-ranked DL concepts at first, the need of excessive number of retrieval operations to find a goal concept can be mitigated. Throughout our experiments, NeRo consistently outperforms state-of-the-art models w.r.t. the F_1 score, the number of explored concepts and the total computational time. Importantly, starting the standard search procedure on these top-ranked concepts further improves the results. Hence, NeRo can be applied within state-of-the-art models to decrease their runtimes. However, it is important to note that Lehmann et al. Lehmann et al. (2011) have previously proved the completeness of CELOE in the CL problem, i.e., for a given learning problem, CELOE finds a goal expression if it exists provided that there are no upper-bounds on the time and memory requirements. Although these requirements are simply not practical, equipping NeRo with the search procedure of CELOE is necessary to achieve the completeness in CL.

Trade-offs & Possible Improvements: Replacing the deep set neural network with the set transformer model may improve the performance further (Lee et al., 2019). Moreover, designing a heuristic function based on embeddings of DL concepts may improve the search procedure of symbolic models.

Part III

Software Frameworks and Use cases

10
DICE Embeddings Framework

Preamble: This chapter is based on Demir and Ngomo (2022).

Declaration of Authorship: The original research contributions were developed by Caglar Demir and discussed with Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the framework, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and revised it with Axel-Cyrille Ngonga Ngomo. The code is available at https://github.com/dice-group/dice-embeddings and https://pypi.org/project/dicee/.

10.1 Large-scale Knowledge Graph Embeddings

Here, we introduce our open-source software framework that facilitates large-scale applications for knowledge graph embedding models. Recently developed KGE frameworks can be effectively applied in research related applications. Yet, these frameworks do not fulfill many requirements of real-world applications. For instance, finding a suitable hyperparameter setting w.r.t. time and computational budgets are left to practitioners. In addition, the continual learning aspect in KGE frameworks is often ignored, although continual learning plays an important role in many real-world (deep) learning-driven applications (Lopez-Paz and Ranzato, 2017; Lesort et al., 2020).

Most of publicly available frameworks require domain knowledge to move computation from a commodity computer to a cluster of computers We develop a framework based on the frameworks Pytorch, Pytorch Lightning, Hugging Face, Pandas, and Polars software libraries to compute embeddings for large-scale knowledge graphs in a hardware-agnostic manner, which can address real-world challenges pertaining to the scale of real applications.

Setup: Let $\mathcal{E}$ and $\mathcal{R}$ represent the sets of entities and relations, respectively. A KG is often formalized as a set of triples $\mathcal{G} = \{(\mathsf{h}, \mathsf{r}, \mathsf{t})\} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ where each triple contains two entities $\mathsf{h}, \mathsf{t} \in \mathcal{E}$ and a relation $\mathsf{r} \in \mathcal{R}$ (Dettmers et al., 2018). Most KGE models are defined as a parameterized scoring function $\phi_\Theta : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \mapsto \mathbb{R}$ such that $\phi_\Theta(\mathsf{h}, \mathsf{r}, \mathsf{t})$ ideally signals the likelihood of $(\mathsf{h}, \mathsf{r}, \mathsf{t})$ is true (Dettmers et al., 2018). In a simple setting, Θ contains an entity embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d}$ and a relation embedding matrix $\mathbf{R} \in \mathbb{R}^{|\mathcal{R}| \times d}$, where d stands for the embedding vector size. Given the triples (`Barack`, `Married`, `Michelle`) and (`Michelle`, `HasChild`, `Malia`) $\in \mathcal{G}$, a good scoring function is expected to return high scores for (`Barack`, `HasChild`, `Malia`) and (`Michelle`, `Married`, `Barack`), while returning a considerably lower score for (`Malia`, `HasChild`, `Barack`).

10.1.1 Challenges

As $|\mathcal{G}|$ increases, the total training time required to learn Θ accurately for a given task also increases. Consequently, This magnifies the importance of effectively utilizing the available computational resources. Most available frameworks including KGE frameworks (e.g. PyKEEN (Ali et al., 2021) and Libkge (Broscheit et al., 2020)) solely rely on a single CPU to load and prepossess the data into the main memory. Consequently, even loading a large KG can take few hours with these frameworks. The aforementioned frameworks facilitate multi-CPU and multi-GPU training as they rely on Pytorch and/or Pytorch-lightning. Yet, they do not support the multi-node training without modification of the code. For instance, to use the multi-node training module of Pytorch, the torchrun module must be included in PyKEEN. Yet, this module is not available. Consequently, without distributed computing software skills, the aforementioned frameworks can only be successfully used provided that data fits into a single computer/node.

Within the publicly available frameworks, downstream applications of pre-trained KGE models are often neglected. To alleviate these limitations among few others, we develop the Dice Embedding Framework.

10.2 DICE Embeddings: Hardware-agnostic Framework for Large-scale Knowledge Graph Embeddings

10.2.1 Hardware-agnostic Computation

The goal of our framework is to facilitate learning large-scale knowledge graph embeddings in an hardware-agnostic manner. Hence, practitioners can use our framework to learn embeddings of KG on commodity hardware as well as a cluster of computers without changing a single line of code. We based our framework on Pytorch, Pytorch Lightning (Falcon et al., 2019), Pandas (pandas development team, 2020), Polars software frameworks (Vink et al., 2023). We use Pandas or Polars to read and preprocess the data in a parallel fashion by using multi-threading and/or multi-CPUs. We observe that most embedding frameworks rely on a single core in reading and preprocessing the input KG. As the size of KG grows, this design decision becomes a hindrance to the scalability and increases the total runtimes. Moreover, the process of the reading, preprocessing, and indexing an input KG is often intransparent to practitioners. That means that as the size of KG grows, practitioners are not informed about the current stage of the total computation. Pytorch and Pytorch Lightning allow our framework to use multi-CPUs,-GPU and even -TPUs in an hardware-agnostic manner.

10.2.2 Finding Suitable Configurations

Our framework dynamically suggests a suitable configuration setting for a given dataset and available computational resources. This includes many features, e.g., finding most memory efficient integer data type for indexing as well as finding a batch size w.r.t. the computational resources. We are actively working on decreasing the domain expert knowledge to facilitate the usage of framework. By this, we aim to share our expert knowledge with practitioners to that their computational and time budgets can be effectively utilized. Computational and time budgets of practitioners play an important role in real-world successful ML applications (Bottou and Bousquet, 2007).

10.2.3 Continual Learning

Our framework assists practitioners to deploy a pre-trained model or continuously train a pre-trained model without writing a single line of code. In many ML applications, the input data set evolves with the time. Hence, continual learning plays an important role in successful applications of ML models Diethe et al. (2019). Although the validity of many assertions is confined withing a time interval (e.g. (`BarackObama`, `president`, `USA`)), most KGE frameworks do not provide means for conform the confined validity of assertions. Hence, pre-trained KGE models are not continuously improved by newly emerging assertions.

10.3 Downstream Applications

In our framework, a pre-trained KGE model can be easily used in many applications including link prediction, triple classification, relation prediction, and conjunctive query answering. Moreover, currently we are working on description logic concept learning.

Finding Missing Triples in a Knowledge Graph

A pre-trained model can be used to detect missing triples as shown in Figure 10.1. Given a path of a pre-trained model, a set of missing triples can be easily computed. For a given entity and a relation, triples whose predicted likelihoods are greater or equal to 0.95, can be found.

```
from dicee import KGE
model = KGE(path='...')
model.find_missing_triples(confidence=0.1,
                           entities=['...'],
                           relations=['...'])
```

Figure 10.1: Finding missing triples with a pre-trained KGE model.

Answering Conjunctive Logic Queries

A pre-trained model can be used to answer conjunctive logic queries as shown in Figure 10.1. Given an entity, spouses of the given entities siblings can be computed.

```
from dicee import KGE
model = KGE(path='')
model.predict_conjunctive_query(entity='...',
                                relations=['...'],
                                topk=1)
```

Figure 10.2: Conjunctive query answering with a pre-trained KGE model.

Description Logic Concept Learning

Currently, we are working on learning description logic concepts by using a pre-trained KGE model as a reasoner. As shown in Figure 10.3, a pre-trained KGE model will be used as a description logic concept learner model.

```
from dicee import KGE
model = KGE(path='..')
model.learn_concepts(pos={''},neg={''},topk=1)
```

Figure 10.3: Learning description logic concept via a pre-trained KGE model.

10.3.1 Deployment

A pre-trained model can be deployed in a web-application without writing a single line of code as shown in Figure 10.4.

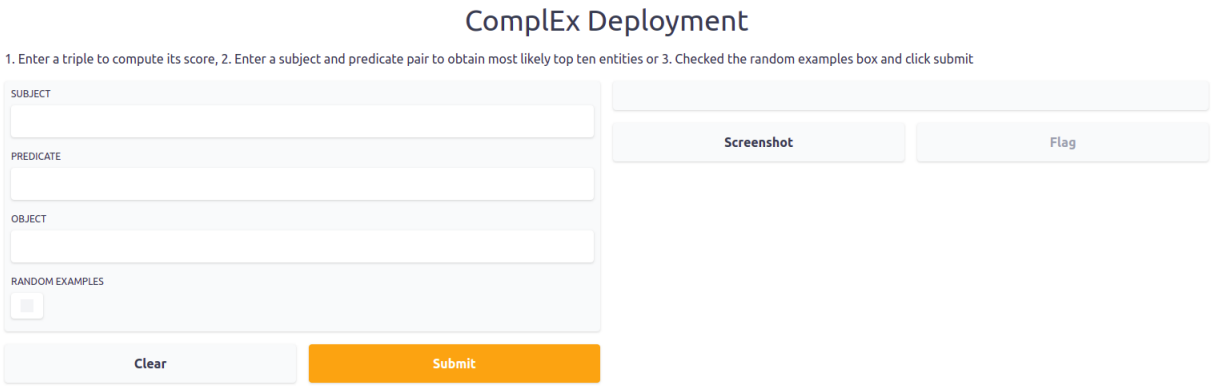


Figure 10.4: A web-application for the deployment without writing a single line of code.

```
class ComplEx(BaseKGE):
    def __init__(self, args):
        super().__init__(args)
        self.name = 'ComplEx'
    def forward_triples(self, x: torch.LongTensor) -> torch.
                                        FloatTensor:
        # (1) Retrieve Embedding Vectors
        head_ent_emb, rel_ent_emb, tail_ent_emb = self.
                                            get_triple_representation(
                                            x)
        # (2) Split (1) into real and imaginary parts.
        emb_head_real, emb_head_imag = torch.hsplit(head_ent_emb, 2)
        emb_rel_real, emb_rel_imag = torch.hsplit(rel_ent_emb, 2)
        emb_tail_real, emb_tail_imag = torch.hsplit(tail_ent_emb, 2)
        # (3) Compute Hermitian inner product.
        real_real_real = (emb_head_real * emb_rel_real *
                                            emb_tail_real).sum(dim=1)
        real_imag_imag = (emb_head_real * emb_rel_imag *
                                            emb_tail_imag).sum(dim=1)
        imag_real_imag = (emb_head_imag * emb_rel_real *
                                            emb_tail_imag).sum(dim=1)
        imag_imag_real = (emb_head_imag * emb_rel_imag *
                                            emb_tail_real).sum(dim=1)
        return real_real_real + real_imag_imag + imag_real_imag -
                                            imag_imag_real
```

Figure 10.5: An example of including state-of-the-art embedding model.

10.3.2 Extendability

The software design of our framework allows practitioners to solely focus on their novel ideas, instead of engineering. For instance, by creating a subclass of the BaseKGE class, a new embedding model can be included into our framework without an effort as shown in Figure 10.5. The structure of BaseKGE class and along with few existing knowledge graph embedding models is visualized in Figure 10.6.

10.3.3 Discussion

PyKEEN uses a single thread on a single CPU core to load a dataset, whereas, our framework uses multi-threads on multi-CPUs. Moreover, our framework can automatically find a good batch size w.r.t. the available CPU memory, whereas a batch size needs to be predetermined in PyKEEN. By automatically finding a large possible batch size, we aim to decrease domain expert knowledge to successfully train KGE models.

Figure 10.6: The structure of BaseKGE class along with few embedding model classes.

Trade-offs & Possible Improvements: Compared to PyKEEN, our framework offers fewer KGE models and less detailed software documentation. Although we are actively improving these matters, our software framework is not as mature as PyKEEN.

11
Ontolearn Framework

Declaration of Authorship: Caglar Demir implemented the framework, conducted the initial experiments, and analyzed their results. Thereafter, the framework has been continuously developed by DICE Group members. The code is available at https://github.com/dice-group/Ontolearn and https://pypi.org/project/ontolearn/.

11.1 OWL Class Expressions in Python

Here, we introduce Ontolearn—an open-source a machine learning framework for class expression learning in Python programming language. Ontolearn includes wide range of class expression learning (CEL) models. Therein, efficient Python implementations of state-of-the-art symbolic models (CELOE, OCEL, ELTL, and DL-Foil), hybrid models (CLIP, EvoLearner), and a neuro-symbolic model (e.g., NCES, Drill, NeRo) are available (Fanizzi et al., 2008; Lehmann et al., 2011; Kouagou et al., 2021, 2022b; Heindorf et al., 2022). Importantly, we implement the following features to facilitate CEL applications: A binding to easily use the DL-learner framework within Ontolearn, automatic learning problem generation, enriching input RDF knowledge base by learned description logic axioms, and an web-service to deploy class expression learning algorithms.

Ontolearn allows practitioners to developed applications by using CEL models with an ease. For instance, as shown in Figure 11.1, a CEL model can be used with few lines of

Python code. The documents of Ontolearn is provided in the project page. [1] [2] Ontolearn is used in the BMWi-funded project RAKI (01MD19012D) and the BMBF-funded project DAIKIRI (01IS19085B). The two industrial use case based on our scientific and software contributions are elucidated in Chapter 13.

```
from ontolearn import *
from owlapy import *
NS = Namespaces('ex', 'http://example.com/father#')
# (1) Learning problem
positive_examples = {OWLNamedIndividual(IRI.create(NS, 'stefan')),
                     OWLNamedIndividual(IRI.create(NS, 'markus')),
                     OWLNamedIndividual(IRI.create(NS, 'martin'))}
negative_examples = {OWLNamedIndividual(IRI.create(NS, 'heinz')),
                     OWLNamedIndividual(IRI.create(NS, 'anna')),
                     OWLNamedIndividual(IRI.create(NS, 'michelle'))}
# (2) Class Expression Learner
model = ModelAdapter(learner_type=CELOE,
                     reasoner_factory=ClosedWorld_ReasonerFactory,
                     path="KGs/father.owl")
# (3) Learning starts
model.fit(pos=positive_examples,neg=negative_examples)
# (4) Display best prediction
dlsr = DLSyntaxObjectRenderer()
for desc in model.best_hypotheses(1):
    print('The result:', dlsr.render(desc.concept), 'has quality',
                                      desc.quality)
```

Figure 11.1: An example of using CELOE model in Ontolearn.

Figure 11.2 illustrates the software design of Ontolearn.

Trade-offs & Possible Improvements: Reading a large input data is a challenge as most open-source libraries read knowledge graphs with a single thread on a single CPU core. Moreover, the standard search procedure for the class expression learning problem becomes a computational bottleneck. Findings of our current work being under review show that tackling the class expression learning problem via our triple store (Tentris (Bigerl et al., 2020)) is an efficient means to scale on large data.

[1] https://github.com/dice-group/Ontolearn
[2] https://ontolearn-docs-dice-group.netlify.app

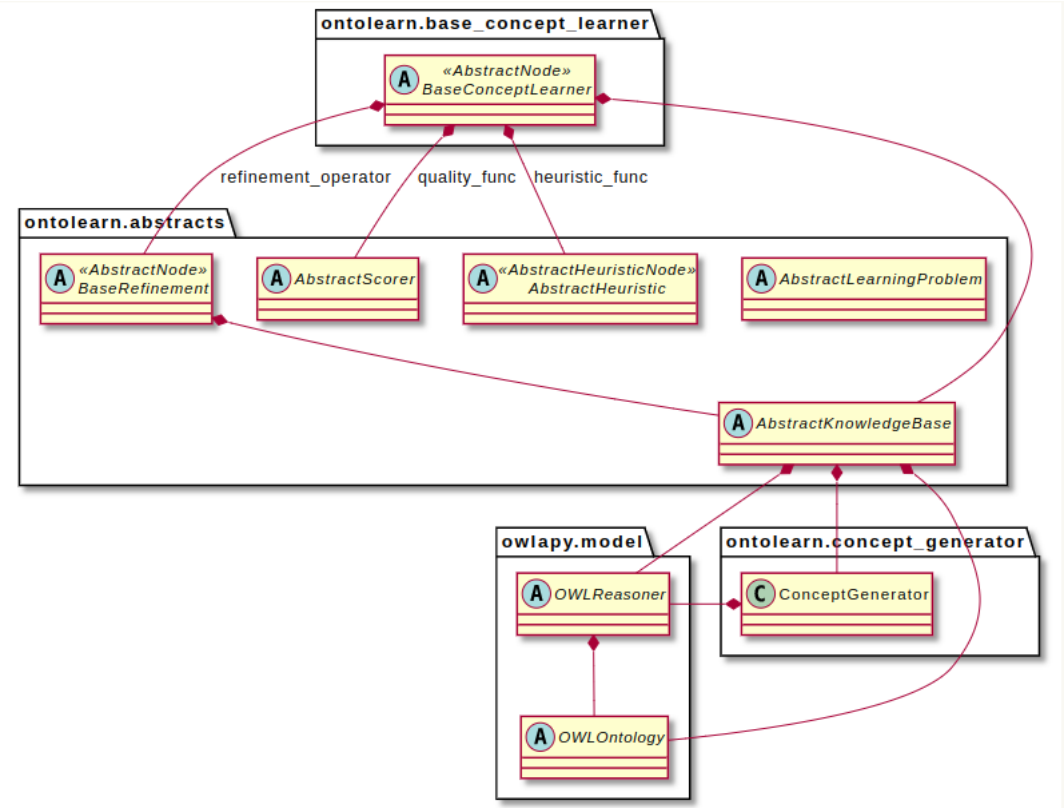


Figure 11.2: A visualization of the Ontolearn framework.

12

Use Case: Deep Reinforcement Learning for Class Expression Learning

Preamble: This chapter is based on Demir and Ngonga Ngomo (2021b).

Declaration of Authorship: The original research idea was introduced by Axel-Cyrille Ngonga Ngomo. The original research contributions were developed by Caglar Demir and discussed with Axel-Cyrille Ngonga Ngomo. Caglar Demir implemented the algorithm, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and Axel-Cyrille Ngonga Ngomo revised it. The code is available at https://github.com/dice-group/DRILL. Chapter 13 presents an industrial application of Drill with our DICE Embedding framework.

12.1 Methodology

Here, we introduce a scientific use case for using a pre-trained knowledge graph embedding model (KGE) to tackle the class expression learning (CEL) problem. We represent a description logic concept/class expression in an embedding vector space by using a pre-trained KGE model. By this, we can represent quasi-ordered description logic concept space as a quasi-ordered reinforcement learning (RL) environment and train a RL agent to tackle the CEL problem more efficiently.

Our model idea plays a foundational role in the BMWi-funded project RAKI (01MD19012D). An industrial use case of this idea is introduced in Chapter 13.

Motivation: Devising a suitable heuristic function is crucial in CEL (Lehmann et al., 2011) as also explained in Section 2.5.3. The search of a $H \in C$ satisfying Equation (2.64) and ideally maximizing Equation (2.67) is steered by a heuristic $\phi : \mathcal{S} \times \mathcal{S} \mapsto \mathbb{R}$ that signals how well refining a quasi-ordered state and transition into one of its refinement state assists to find a H. Equation (2.66) indicates that $\phi(\mathsf{A}, \mathsf{B})$ of state-of-the-art approaches compute heuristic values without incorporating any information pertaining to $\rho(\mathsf{B})$. This is analogous to setting $\gamma = 0$ in Equation (2.55), i.e., to setting the present value of future rewards to 0. This implies that heuristic functions of state-of-the-art CEL models correspond to myopic RL agents, whose only concern is to maximize immediate rewards (Sutton and Barto, 2018). To address this drawback, we leverage the deep Q-learning framework. In Table 12.1, we summarized the symbols used in this chapter.

Table 12.1: An overview of our notation used in Chapter 12.

Notation	Description
$\mathcal{K}$	Knowledge Base
N_C, N_R, N_I	Set of named concepts, roles, and individuals, respectively
C	Set of all valid description logic concepts
$\mathcal{S}$	Quasi-ordered C/search space
$\mathcal{R}(\cdot)$	Instance retrieval function
E^+, E^-	Positive and negative example sets for a CEL problem, respectively
ϕ_{CELOE}	CELOE's heuristic function
Drill_Θ	Drill's parameterized heuristic function
$\mathbf{W}, \mathbf{H}$	Two linear transformations
$\mathcal{E}(\cdot)$	Embedding lookup function for $\mathcal{ALC}$ concepts
d	Embedding vector dimensions
$\rho(\cdot)$	Refinement operator
$\mathbf{R}(\cdot, \cdot)$	Reward function
Ψ	Concatenation function
$*$	2D convolution operation
ω	Filters/kernels in the convolution operation

Quasi-ordered RL Environment: A RL environment is constructed via continuous vector representations of quasi-ordered states. To obtain these representations, we use pre-trained knowledge graph embedding models provided in (Demir and Ngonga Ngomo, 2021a). An embedding of $\mathcal{ALC}$ class expression $\mathtt{A}$ corresponds to an embedding of a set of individuals obtained via the retrieval function $\mathcal{R}(\mathtt{A})$. This embedding lookup operation is denoted with $\mathcal{E}(\mathcal{R}(\cdot))$, where $\mathcal{E} : 2^{N_I} \rightarrow \mathbb{R}^{|2^{N_I}| \times d}$. A RL state is represented via $\mathcal{R}$ with a pre-trained embedding model (e.g., ConEx). For instance, the initial state of this RL environment is represented with embeddings of $\top$, i.e., embedding of all individuals $\mathcal{E}(\mathcal{R}(\top))$. In this environment, we consider a RL action as an action of refining a quasi-ordered state and transitioning to another quasi-ordered state.

Drill: The standard deep Q-loss function introduced in Equation (2.58) cannot be directly applied in this quasi-ordered RL environment since the set of possible actions is not fixed as in (Mnih et al., 2015). To mitigate this issue, (Edwards et al., 2020) designed the state-state Q function that leads RL agents to naturally *avoid redundant states*. As the number of redundant states increases, using the state-state Q function often leads in more favorable results than using the state-action Q function. The ability of avoiding redundant states without requiring any additional computation is particularly important in our purpose, as many state-of-the-art symbolic models (e.g., CELOE, OCEL, and ELTL) apply redundancy elimination technique to remove redundant states from their search (Lehmann, 2010; Lehmann et al., 2011). Therefore, minimizing the state-state Q function permits *incorporating consideration for future states in immediate decisions* and *pruning redundant states without additional computation*. With these considerations in mind, we designed the following loss function

$$\ell(\Theta_i) = \mathbb{E}_{(\mathbf{s},\mathbf{s}',r,\mathbf{e}_+,\mathbf{e}_-)\sim U(\mathcal{D})}\left[\left(r + \gamma \max_{x \leq s'} Q(\mathbf{s}', \mathbf{x}, \mathbf{e}_+, \mathbf{e}_-; \Theta_i^-) - Q(\mathbf{s}, \mathbf{s}', \mathbf{e}_+, \mathbf{e}_-; \Theta_i)\right)^2\right]. \quad (12.1)$$

To minimize Equation (12.1), we propose a convolutional deep Q-Network parameterized with $\Theta = [\omega, \mathbf{W}, \mathbf{H}]$ as follows

$$\textsc{Drill}_\Theta([\mathbf{s}, \mathbf{s}', \mathbf{e}_+, \mathbf{e}_-]) = \mathrm{ReLU}\Big(\mathrm{vec}(\mathrm{ReLU}\big[\Psi([\mathbf{s}, \mathbf{s}', \mathbf{e}_+, \mathbf{e}_-]) * \omega)\big] \cdot \mathbf{W}\Big) \cdot \mathbf{H}, \quad (12.2)$$

where $\mathbf{s} \in \mathbb{R}^{|\mathcal{R}(\mathsf{S})| \times d}$ and $\mathbf{s}' \in \mathbb{R}^{|\mathcal{R}(\mathsf{S}')| \times d}$ denote two RL states that are represented with embeddings of two $\mathcal{ALC}$ expressions S and S'. Embeddings of S and S' are obtained through applying $\mathcal{R}(\cdot)$ and $\mathcal{E}(\cdot)$ consecutively. Similarly, $\mathbf{e}_+ \in \mathbb{R}^{|E^+| \times d}$ and $\mathbf{e}_- \in \mathbb{R}^{|E^-| \times d}$ are obtained via $\mathcal{E}(\cdot)$. Note that $\mathcal{E}(\cdot)$ returns a set of embedding vectors of size d. To obtain permutation-invariant representations, we apply $\Psi(\cdot)$ to convert a input into $\mathbb{R}^{4 \times d}$ by averaging the embeddings of its input. By doing so, the output of ϕ_{Drill} is invariant to the order of items in $\mathbf{s}, \mathbf{s}', \mathbf{e}_+$, and $\mathbf{e}_-$. Permutation invariance is required as items in a set do not have an ordering. Moreover, $\text{ReLU}(\cdot)$, $\text{vec}(\cdot)$, $*$ and ω denote the rectified linear unit function, a flattening operation, the convolution operation, and kernels in the convolution operation. $\mathbf{W}$ and $\mathbf{H}$ denote two linear transformations, respectively. We design Drill via the convolution operation as we aim benefit its parameter efficiency (Mnih et al., 2013, 2015).

Construction of Rewards: We base our reward function on the heuristic function of CELOE:

$$\mathbf{R}(\mathbf{s}, \mathbf{s}') = \begin{cases} maxreward, & \text{if F1-score(S')=1;} \\ \phi_{\text{CELOE}}(S, S'), & \text{if F1-score(S')<1.} \end{cases} \tag{12.3}$$

A reward of transition from RL state $\mathbf{s}$ to $\mathbf{s}'$ is based on the heuristic value of refining a quasi-ordered class expression S and transitioning to S' provided that S' is not a class expression satisfying Equation (2.64). Through minimizing Equation (12.1) on data points with such rewards, Drill is expected to steer the search towards shorter class expressions without computing lengths of class expressions. This stems from the fact that Drill is trained on rewards that involves the length information provided within ϕ_{CELOE} (see Equation (2.66)). Drill mitigates the myopic heuristic nature of ϕ_{CELOE} through learning discounted cumulative future rewards, i.e., discounted cumulative future CELOE heuristic values. Through minimizing Equation (12.1) on $\mathcal{D}$, Drill is expected to steer the search towards shorter class expressions without computing lengths of class expressions.

12.1.1 Unsupervised Training

To generate learning problems, $\rho(\cdot)$ is iteratively applied in a randomized depth-first search manner starting from $\top$. During this randomized process, each state s satisfying the length constraint $1 \leq |S| \leq maxlen$ and $|\mathcal{R}(S)| > 0$ is stored.

Algorithm 5 Drill with deep Q-learning training procedure

Require: $E^+, E^-, \rho, \mathbf{R}, \mathcal{R}, \mathcal{E}, \Theta, \top, M, T, U$

for m := 1, M **do**
  $\mathbf{s}_0, s_0 ::= \mathcal{E}(\mathcal{R}(\top)), \top$
  for $t := 0, T$ **do**
    $\mathbf{z} := \{s' \in \rho(s_t) | \mathcal{E}(\mathcal{R}(s'))\}$
    if $\epsilon > .1$ **then**
      Select random $\mathbf{s}' \in \mathbf{z}$
    else
      Select $\mathbf{s}' := argmax_{\mathbf{s}' \in \mathbf{z}} \phi_{\text{Drill}}([\mathbf{s}_t, \mathbf{s}', \mathbf{e}_+, \mathbf{e}_-]); \Theta)$
    end if
    Compute reward $r_t := \mathbf{R}(\mathbf{s}_t, \mathbf{s}')$
    Append $[\mathbf{s}_t, \mathbf{s}, \mathbf{e}_+, \mathbf{e}_-, r_t]$ to $\mathcal{D}$
    Set $\mathbf{s}_{t+1} := \mathbf{s}_t$
  end for
  Reduce ϵ with a constant
  if m % U == 0 **then**
    Sample random mini-batches from $\mathcal{D}$
    Compute loss of mini-batches w.r.t. Equation (12.1)
    Update Θ accordingly
  end if
end for
return Θ

In our experiments, we set $maxlen = 5$. To ensure the heterogeneity of the set of learning problems, we perform this task m times. For each stored state, we compute all positive $E^+ = \mathcal{R}(S)$ and negative examples $E^- = N_I \setminus \mathcal{R}(S)$, respectively. This operation often results in creating imbalanced E^+ and E^-, with $|E^-| >> |E^+|$ being common. To alleviate imbalanced examples, we randomly undersample the largest set of examples so that $|E^+| = |E^-|$. The procedure is applied several times to generate various learning problems.

12.1.2 Refinement Operator

So far, we have assumed that a refinement operator ρ is given. In the following, we present the refinement operator on $\mathcal{ALC}$ designed for Drill. The operator is designed to make no use of subsumption information but rather to consider concept length as ordering. This in contrast to most refinement operators found the in the literature, which order concepts w.r.t. the subsumption relation. We chose to design such an operator to measure the influence of subsumption semantics on Drill.

Let N_C be a finite set of named concepts and let N_R be a finite set of roles. We write $\top$ to denote the top concept and $\bot$ to denote the bottom concept. We set $N_C^+ = N_C \cup \{\top, \bot\}$. The set S of all $\mathcal{ALC}$ class expressions built upon N_C and N_R is defined as follows: First, $N_C^+ \subset S$. Now, let $r \in N_R$. If C and D are elements of S, then so are $(C \sqcap D)$, $(C \sqcup D)$, $\exists r.C$, $\forall r.C$, and $\neg C$. The semantics of $\mathcal{ALC}$ are given in Table 12.2.

Table 12.2: $\mathcal{ALC}$ syntax and semantics. $\mathcal{I}$ stands for an interpretation, $\Delta^{\mathcal{I}}$ for its domain.

Construct	Syntax	Semantics
Atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Role	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Top concept	$\top$	$\Delta^{\mathcal{I}}$
Bottom concept	$\bot$	$\emptyset$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Existential restriction	$\exists\, r.C$	$\{x \mid \exists\, y.(x,y) \in r^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$
Universal restriction	$\forall\, r.C$	$\{x \mid \forall\, y.(x,y) \in r^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}$

Let $X, Y \in S$ and $r \in N_R$. We define the length of a concept $C \in S$ (denoted $|C|$) as follows:

1. If $C \in N_C^+$, then $|C| = 1$;
2. If $C = X \sqcup Y$ or $C = X \sqcap Y$, then $|C| = |X| + |Y| + 1$;
3. If $C = \exists r.X$ or $C = \forall r.X$, then $|C| = |X| + 2$;
4. If $C = \neg X$, then $|C| = |X| + 1$.

We use the length of concepts to define an ordering over the set S as follows: $\forall C, D \in S : C \preceq D$ iff $|C| \leq |D|$. We define the operator ρ over $(S, \preceq)$ as follows:

$$\rho(C) = \begin{cases} \{\exists r.C, \forall r.C, C \sqcap \top, C \sqcup \top, \neg C, C\} & \text{for any } C \\ \{\exists r.\rho(X)\} & \text{if } C = \exists r.X \\ \{\forall r.\rho(X)\} & \text{if } C = \forall r.X \\ \{\neg \rho(X)\} & \text{if } C = \neg X \\ \{\rho(X) \sqcup \rho(Y)\} & \text{if } C = X \sqcup Y \\ \{\rho(X) \sqcap \rho(Y)\} & \text{if } C = X \sqcap Y \\ N_C^+ & \text{if } C = \top \end{cases} \tag{12.4}$$

We use notation such as $\exists r.\rho(X)$ and $\forall r.\rho(X)$ as a shorthand for $\exists r.Y | Y \in \rho(X)$ and $\forall r.Y | Y \in \rho(X)$ respectively. We write the operator in this fashion for the sake of legibility. Note that several of the rules can apply for a given C. For example, if $C = \exists r.D$, then the first and second rule apply. In this case, the operator returns the union of the outputs of the rules which apply. In the following, we will assume that ρ begins the refinement process from $\top$.

Theorem 1 *ρ is an upward refinement operator over $(S, \preceq)$.*

Proof 1 *We need to prove that $|\rho(C)| \geq |C|$. This is a direct consequence of the construction of C. For each possible refinement computed ρ and by virtue of the definition of $|\cdot|$, ρ either preserves the length of a concept (e.g., if $C \in N_C^+$) or leads to longer concepts (e.g., for some refinements in the first line of the specification of ρ).*

It is obvious that ρ is *redundant*, i.e., there can be more than one sequence of refinements from $\top$ to a concept C.

The operator is *finite* as $|\rho(C)| < \infty$ for any $\mathcal{ALC}$ concepts C built upon a finite set of named concepts N_C and a finite set of roles. We exploited these two characteristics during the implementation of our refinement operator. In particular, we implemented the operator to only explore duplicate concepts once refraining from backtracking to deal with infinite refinements.

12.2 Experiments & Results

12.2.1 Experiments

Experimental Setup

We based our experimental setup on (Bühmann et al., 2016) and used learning problems provided therein. To perform an extensive comparison between approaches, we also generated learning problems automatically. We ensured that none of the learning problems used in our evaluation has been used in the unsupervised training phase. In our experiments, we evaluated all models in $\mathcal{ALC}$ for CEL on the same hardware. All experiments were conducted with Ubuntu 18.04 with 16 GB RAM and an Intel(R) Core(TM) i5-7300U CPU v4 @ 2.60GHz.

Datasets

We used four benchmark datasets (Family, Carcinogenesis, Mutagenesis and Biopax) to evaluate Drill (Bin et al., 2016; Fanizzi et al., 2008, 2019; Lehmann and Hitzler, 2010). An overview of the datasets is provided in Table 12.3.

Table 12.3: An overview of class expression learning benchmark datasets.

Dataset	$\lvert N_I \rvert$	$\lvert N_C \rvert$	$\lvert N_R \rvert$
Family	202	18	4
Carcinogenesis	22372	142	21
Mutagenesis	14145	86	11
Biopax	323	28	49

Evaluation

We compared approaches via the F1-score, accuracy, the runtime and number of tested class expressions in a manner akin to Lehmann and Hitzler (2010).

The F1-score and accuracy were used to measure the quality of the class expressions found w.r.t. positive and negative examples, while the runtime and the number of tested class expressions were used to measure the efficiency. We used two standard stopping criteria for all approaches. We set the maximum runtime to 5 seconds as models often reach good solutions within 1.5 seconds (Lehmann and Hitzler, 2010). Approaches were configured to terminate as soon as they found a goal state (i.e., a state with F1-score = 1.0).

Hyperparameter Optimization

We generated 10 learning problems for each benchmark datasets by following the unsupervised training procedure. For each learning problem, Drill is trained according to Algorithm 5. In our experiments, we used the pretrained embeddings of input knowledge graphs provided by (Demir and Ngonga Ngomo, 2021a). The parameter selection was carried out based on (Mnih et al., 2015) by using the following configuration: ADAM optimizer with learning rate of .01, mini-batches of size 512, number of episodes set to 100, an epsilon decay of .01, a discounting factor γ of .99, 32 input channels, and (3×3) kernels. The parameter selection was carried out based on (Mnih et al., 2015). The offline training on all benchmark datasets took 18 minutes on Family, 143 minutes on Mutagenesis, 119 on Carcinogenesis, and 21 minutes on Biopax.

12.2.2 Results

Comparison with the state of the art Table 12.4 reports results on standard learning problems provided within the DL-Learner framework (Bühmann et al., 2016). These results suggest that approaches yield similar performances in terms of F1-score and accuracy on benchmark datasets. Yet, Drill outperforms all other approaches on all datasets w.r.t. its runtime. On all benchmark datasets, Drill requires at most **4 seconds** to yield competitive performance, while CELOE and ELTL require at most **21 seconds** and **18 seconds**, respectively. Overall, Drill is at least **2.7 times** more time-efficient than CELOE, OCEL and ELTL on all standard learning problems. Moreover, during our experiments, we observed that (1) the number of tested class expressions in ELTL and

(2) the F1-score of the best found class expression in OCEL are not reported. Results also indicate that CELOE, OCEL, and ELTL do not terminate within the set maximum runtime.

We delved into their implementations in the DL-Learner framework and observed that the maximum runtime criterion is not checked until refinements of a given class expression are obtained. Table 12.4 indicates that CELOE explores fewer states than other approaches. This is probably due to **the redundancy elimination** and **expression simplification** implemented in the DL Learner (see section 6.1 in (Lehmann and Hitzler, 2007)). The redundancy elimination means that CELOE exploits the proven redundancy of its refinement operator by querying whether an expression already exists in the search tree. If yes, then this expression is not added into the search tree. The expression simplification reduces long expressions into shorter ones, e.g., $\top \sqcup \texttt{Person}$ and $\forall r.\top$ into `Person` and $\top$, respectively. These modifications often lead to explore less number of states. However, they introduce extra computation, which are clearly reflected in the longer runtimes of CELOE w.r.t. Drill (see Table 12.4). Note that we also evaluated DL-FOIL in our initial experiments. However, we observed that DL-FOIL often failed to terminate within 5 minutes. This may stem from the fact:

1. DL-FOIL does not use the elapsed time as a stopping criterion (see Section 4 (Fanizzi et al., 2008)).

2. DL-FOIL requires to find a class expression H with $\forall n \in E^- \; \mathcal{K} \not\models H(n)$ to terminate (see Figure 1 in (Fanizzi et al., 2008)).

Consequently, we could not include DL-FOIL in our final set of experiments. Moreover, we could not include SParCEL into our experiments as the publicly available code cannot be executed with some dependencies that are not provided (Tran et al., 2017).

Table 12.5 shows the details of our evaluation on 18 learning problems on the Family dataset. These details suggest that Drill and CELOE often converge towards shorter concepts compared to OCEL and ELTL. This may stem from the fact that Drill indeed learned to assign low values for longer concepts as Drill is trained on rewards based on the heuristic function of CELOE. Drill finds a goal state within a second in 12 out of 18 learning problems.

We also observed that there are only few learning problems per dataset. To perform more extensive comparisons, we automatically and randomly generated more learning problems.

Table 12.4: Results on benchmark learning problems. F1, Acc, T and Exp denote the F1-score, the accuracy, runtime in seconds, and number of class expression tested respectively. † stands for no solution found and by the respective approach. ∗ indicates that respective value is not reported in DL-Learner. Bold entries denote best results.

Dataset	Drill				CELOE				OCEL				ELTL			
	F1	Acc	T	Exp	F1	Acc	T	Exp	F1	Acc	T	Exp	F1	Acc	T	Exp
Family	0.96	0.95	**1.2**	2052	**0.97**	**0.97**	3.6	646	*	0.94	6.1	2563	0.96	0.95	3.4	*
Carcinogenesis	**0.71**	0.56	**3.3**	305	**0.71**	0.56	21.1	230	†	†	23.5	802	**0.71**	**0.57**	22.1	*
Mutagenesis	**0.70**	**0.54**	**3.0**	3941	**0.70**	**0.54**	13.9	135	†	†	13.2	4023	**0.70**	**0.54**	13.2	*

Table 12.6 reports results on 370 learning problems generated automatically on benchmark datasets. These results confirm that Drill finds a goal state faster than all other approaches. Importantly, Drill was always able to find goal concepts in all 370 learning problems.

Statistical Hypothesis Testing: We carried out a Wilcoxon signed-rank test to check whether our results are significant. Our null hypothesis was that the performances of Drill and CELOE come from the same distribution provided that a goal state is found. The alternative hypothesis was that these results come from different distributions. To perform the Wilcoxon signed-rank test (one-and two-sided), we used the runtimes of Drill and CELOE on benchmark datasets provided both approaches found a goal state. We were able to reject the null hypothesis with a p-value $< 1\%$. Ergo, the superior runtime performance of Drill is statistically significant.

Parameter Analysis and Optimization: Drill achieves state-of-the-art performance on all datasets without over-parameterization and extensive parameter optimization. Throughout our experiments, Drill is trained with a fixed configuration: 32 input channels, (3x3) kernel.

12.2.3 Discussion

Our results on all benchmark datasets suggest that Drill achieves state-of-the-art performance w.r.t. the quality of the expressions found during the search, while outperforming the state of the art significantly w.r.t. its runtime.

The superior performance of Drill is due to the following:

Table 12.5: Results on individual learning problems of the Family benchmark dataset. † denotes no solution found by the respective approach. L, F1, Acc and T denote the length of predicted class expression, F1-score, accuracy and runtime in seconds, respectively. Bold entries denote best results.

Expression	Drill				CELOE				OCEL				ELTL			
	L	F1	Acc	T	L	F1	Acc	T	L	F1	Acc	T	L	F1	Acc	T
Aunt	6	**0.83**	0.79	**3.3**	6	**0.83**	0.79	5.7	16	*	**1.00**	5.8	1	0.800	0.76	2.8
Brother	1	**1.00**	**1.00**	**0.2**	1	**1.00**	**1.00**	2.9	1	*	**1.00**	5.8	5	**1.00**	**1.00**	3.8
Cousin	4	0.73	0.65	**2.9**	5	**0.79**	0.74	5.9	21	*	**1.00**	6.2	1	0.66	0.50	3.0
Daughter	1	**1.00**	**1.00**	**0.2**	1	**1.00**	**1.00**	2.9	1	*	**1.00**	5.9	3	**1.00**	**1.00**	2.9
Father	1	**1.00**	**1.00**	**0.2**	1	**1.00**	**1.00**	3.0	1	*	**1.00**	6.0	3	**1.00**	**1.00**	3.0
Granddaughter	1	**1.00**	**1.00**	**0.2**	1	**1.00**	**1.00**	3.1	1	*	**1.00**	5.3	1	**1.00**	**1.00**	2.9
Grandfather	1	**1.00**	**1.00**	**0.2**	1	**1.00**	**1.00**	2.9	1	*	**1.00**	5.7	1	**1.00**	**1.00**	3.0
Grandgranddaughter	1	**1.00**	**1.00**	**0.2**	1	**1.00**	**1.00**	2.9	1	*	**1.00**	5.9	7	**1.00**	**1.00**	3.0
Grandgrandfather	1	0.94	0.94	**3.2**	5	**1.00**	**1.00**	3.0	5	*	**1.00**	5.8	7	**1.00**	**1.00**	3.7
Grandgrandmother	9	0.94	0.94	**3.3**	5	**1.00**	**1.00**	3.1	5	*	**1.00**	5.9	7	**1.00**	**1.00**	3.7
Grandgrandson	1	0.92	0.92	**3.5**	5	**1.00**	**1.00**	5.7	5	*	**1.00**	6.6	7	**1.00**	**1.00**	3.1
Grandmother	1	**1.00**	**1.00**	**0.2**	1	**1.00**	**1.00**	2.8	1	*	**1.00**	5.9	1	**1.00**	**1.00**	3.1
Grandson	1	**1.00**	**1.00**	**0.2**	1	**1.00**	**1.00**	2.8	1	*	**1.00**	6.0	1	**1.00**	**1.00**	3.0
Mother	1	**1.00**	**1.00**	**0.2**	1	**1.00**	**1.00**	2.9	1	*	**1.00**	5.9	5	**1.00**	**1.00**	3.1
PersonWithASibling	1	**1.00**	**1.00**	**0.2**	1	**1.00**	**1.00**	2.8	1	*	**1.00**	7.0	1	**1.00**	**1.00**	3.1
Sister	1	**1.00**	**1.00**	**0.2**	1	**1.00**	**1.00**	2.8	1	*	**1.00**	5.8	5	**1.00**	**1.00**	3.0
Son	1	**1.00**	**1.00**	**0.2**	1	**1.00**	**1.00**	3.0	1	*	**1.00**	5.7	3	**1.00**	**1.00**	2.9
Uncle	6	**0.90**	**0.89**	**2.9**	6	**0.90**	**0.89**	5.9	†	†	†	5.8	1	0.88	0.87	2.9

Table 12.6: Results on automatically generated learning problems. #LP denotes the number learning problems. Bold entries denote best results.

Dataset	#LP	Drill				CELOE				OCEL				ELTL			
		F1	Acc	T	Exp	F1	Acc	T	Exp	F1	Acc	T	Exp	F1	Acc	T	Exp
Family	74	**1.00**	**1.00**	**1.1**	32	**1.00**	**1.00**	3.6	14	*	**1.00**	6.2	2403	**1.00**	**1.00**	3.5	*
Carcinogenesis	100	**1.00**	**1.00**	**2.2**	47	**1.00**	**1.00**	17.3	16	*	**1.00**	20.6	5876	**1.00**	**1.00**	19.1	*
Mutagenesis	100	**1.00**	**1.00**	**1.4**	268	**1.00**	**1.00**	10.0	148	*	0.98	12.9	3867	0.97	0.97	10.2	*
Biopax	96	**1.00**	**1.00**	**1.1**	40	0.99	0.99	3.7	55	*	**1.00**	6.74	5691	0.99	0.98	3.7	*

1. deep Q-learning with the state-state Q loss function,
2. the length-based refinement operator and
3. the efficient computation of heuristic values.

Deep Q-learning endows Drill with the ability of considering future rewards, while selecting the next state for the refinement. State-of-the-art approaches lack this ability. Through minimizing the state-state Q-loss function, Drill can detect redundant states without any additional computation. To detect redundant states, CELOE, OCEL, and ELTL require additional computations. Moreover, Drill learns to converge to simple expressions without computing lengths of expressions, whereas other approaches yet again perform addition computations (see Equation (2.66)). These additional computations inherently increase their runtimes. Moreover, the Q-network used in Drill can assign scores for the refinements in a batch manner, while the sequential nature of state-of-the-art models precludes batch computation.

To exclude that our results were due to the approach used to generate learning problems being akin to that used to train Drill, we carried out experiments with learning problems whose positive and negative examples were selected randomly. Although CEL problems are rarely constructed at random, we aimed to check the hypothesis that our runtime improvement was due to our approach to problem generation. Our results clearly indicate that this is not the case.

We also wanted to know how the performance of the different approaches changes as the input size of random learning problems increases. Table 12.7 suggests that OCEL does not find any adequate solution as the size of the random inputs increase. ELTL performs poorly compared to CELOE and Drill. Moreover, Drill and CELOE often return expressions that have similar F1-scores. However, Drill often finds expressions having higher accuracy in a significantly better time. Hence, we can conclude that it is improbable that the better runtime of Drill is due to the experimental setting. Finally, we were also interested in comparing the impact of length-based and CELOE's refinement operator in Drill'performance. Table 12.8 shows that Drill with length-based performance performs better than using CELOE's refinement operator. These results suggest that our length-based refinement operator allows Drill to steer the search towards more accurate expressions efficiently.

Table 12.7: Results on random learning problems. Each row reports average results of 10 learning problems. E^+ and E^- are constructed via drawing N each random individuals N_I without replacement. † and #LP stand for no solution found by the approach and for the number learning problems, respectively. $*$ indicates that respective value is not reported in DL-Learner. Bold entries denote best results.

Dataset	N	Drill				CELOE				OCEL				ELTL			
		F1	Acc	T	Exp	F1	Acc	T	Exp	F1	Acc	T	Exp	F1	Acc	T	Exp
Family	1	**0.97**	0.95	**0.4**	563	**0.97**	0.95	3.6	250	*	**1.0**	6.1	4083	**0.97**	0.95	3.6	*
	10	0.77	**0.73**	**1.3**	4968	**0.78**	0.72	6.3	3218	†	†	†	†	0.67	0.51	3.3	*
Carcinogenesis	1	**0.93**	0.90	**1.2**	123	**0.93**	0.90	19.3	466	*	0.80	20.1	9253	**0.93**	0.90	19.3	*
	10	**0.71**	**0.60**	**2.2**	305	**0.71**	0.58	22.7	208	†	†	†	†	0.67	0.50	19.6	*
Mutagenesis	1	0.90	0.85	**2.8**	1192	**0.97**	0.95	11.4	345	*	0.90	14.0	4880	**0.93**	0.90	11.6	*
	10	**0.71**	**0.61**	**3.2**	3909	0.70	0.58	13.9	208	†	†	†	†	0.67	0.50	10.9	*
Biopax	1	**0.93**	0.90	**1.0**	1708	**0.93**	0.90	4.9	369	*	0.80	7.3	4943	**0.93**	0.90	4.7	*
	10	**0.73**	**0.65**	**3.7**	7033	0.72	0.60	7.1	1514	†	†	†	†	0.67	0.50	4.4	*

Trade-offs & Possible Improvements: Compared to symbolic models, Drill requires two separate training phases: training a knowledge graph embedding model phase and a reinforcement learning offline training phase. Performance of Drill can be improved by replacing the mean operation by a Deep Set or Set Transformer models to obtain invariant representations Zaheer et al. (2017); Lee et al. (2019). Moreover, designing a technique to measure the difficulty of learning problem in the context of class expression learning may also improve Drill by ignoring trivial learning problem during the offline training.

Table 12.8: Results on automatically generated learning problems. N, F1, Acc, T and Exp. denote the number of learning problems, the F1-score, accuracy, runtime in seconds, and the number of class expression tested respectively. Results are obtained by using our length-based refinement operator and CELOE's refinement operator. Bold entries denote best results.

Dataset	N	Length-Based Ref.				CELOE's Ref.			
		F1	Acc	T	Exp	F1	Acc	T	Exp
Family	74	**1.00**	**1.00**	**1.1**	**32**	0.97	0.97	1.25	1075
Carcinogenesis	100	**1.00**	**1.00**	**2.2**	**47**	0.93	0.93	12.8	475
Biopax	96	**1.00**	**1.00**	**1.1**	**40**	0.99	0.99	1.68	13

13

Use Case: Rapid Explainability For Skill Description Learning

Preamble: This chapter is based on Demir et al. (2022a).

Declaration of Authorship: The original research idea was introduced by Axel-Cyrille Ngonga Ngomo in the BMWi-funded project RAKI (01MD19012D). Caglar Demir implemented the framework containing the inductive logic programming and the deep reinforcement learning modules, conducted the experiments, and analyzed their results. Caglar Demir wrote the manuscript and all authors revised it.

Use case: Within the Industry 4.0, smart factories and cyber-physical systems are tied to the promise of increased flexibility, adaptability, and transparency in production, thus increasing the autonomy of machines. One such manufacturing process is *skill matching*, where operations in a production process are assigned to machines. A prerequisite for this technology are skill descriptions of the machines and skill requirements of the operations (Himmelhuber et al., 2020). In some cases, skill descriptions are only partially available or not available at all. Defining and digitizing skill descriptions of a production module are typically done manually by domain experts in a time- and resource-intensive process. Here, we report on the initial results of the RAKI project jointly carried out by Paderborn University, Leipzig University, and Siemens AG. We designed a framework

to facilitate Description Logic (DL) concept learning on large industrial RDF knowledge bases. Our framework learns Description Logics (DL) concepts significantly faster than state-of-the-art models. Through verbalization means, our framework obtains interpretable results even for non-domain experts. We open-sourced our framework to foster large-scale applications[1]. This use-case has been supported by the BMWi-funded project RAKI (01MD19012D).

Objectives: Our goal is to efficiently tackle the skill description learning problem. Given that skills can be encoded as OWL class expressions, addressing this challenge can be mapped to a OWL CEL problem (Lehmann et al., 2011). This form of ante-hoc explainable AI (XAI) is well suited for skill learning, as class expressions are interpretable—e.g., through verbalization techniques (Moussallem et al., 2020)—and, a lack of transparency and explainability in AI would reduce the acceptance of the skill descriptions learned automatically (Holzinger et al., 2019). However, successful large-scale industrial applications of CEL models have not yet taken place. Arguably, this stems from the impractical runtimes of CEL models due to their reliance of fixed myopic heuristic functions and not utilizing parallelism. We hence rely on a framework that reduces the impractical runtimes of classical CEL approaches so that CEL can be carried out on large RDF knowledge bases. The framework includes a verbalization module to decrease the amount of AI expertise necessary to interpret results of CEL problems. Hence, learned concepts are interpretable even for novice practitioners. The proposed CEL framework is generic and has the potential of easing the use of XAI in real-life applications along with contributing to the corresponding societal advantages tied to explainability (Burnett, 2020).

Solution: We design the RAKI framework that is based on inductive logic programming, deep reinforcement learning and verbalization modules. We design a model (Drill) based on deep Q-Network model to significantly decrease the impractical runtimes. This model is elucidated in Chapter 12. This is achieved by replacing a fixed myopic heuristic function with a learned Q-function that incorporates future considerations in immediate actions.

[1] https://github.com/dice-group/DRILL

In the RAKI framework, knowledge graph embedding model can be easily learned with our Dice framework (elucidated in chapter 10) to train Drill. The verbalization module translates a learned class expression into natural language sentences.

Results: The RAKI framework was evaluated on learning problems from the skill description use case. The results show that our framework yields the best performance with respect to accuracy and F1-score, and that it can also learn class expressions that describe the positive and negative examples in the learning problem more precisely than state-of-the-art baselines. In particular, the integration of domain knowledge by excluding previously defined concepts leads to non-trivial and thus more useful class expressions. Especially the high scalability of the framework allowed the calculation of results in a short time, while the second-fastest baseline needed approximately 8 times as long on this use case. Moreover, the verbalization of OWL class expressions to natural language made it easier for the plant operators to understand the skill descriptions, facilitating the subsequent skill matching step.

Business value: Automatizing the skill description learning problem reduces personnel expenses since a skill description of a production module can be learned without a domain expert. Moreover, the verbalization module saves time of domain experts, as learned descriptions can be interpreted easily. Importantly, the ability of tackling this problem efficiently allows us to fully utilize large RDF knowledge bases. Thanks to the RAKI project, we have developed Drill introduced in Chapter 12 that combines for Inductive Logic Programming (ILP) with RL to learn DL concepts for the skill learning problem, where a skill corresponds to a DL concept.

Part IV

Conclusion and Future Work

14
Conclusion and Discussion

In this thesis, we focused on learning continuous vector representations for knowledge graphs. Designing effective and efficient knowledge graph embedding models constituted the core of the thesis. In this chapter, we provide a summary of our contributions, and elucidate our findings.

- In Chapters 3 to 6, we proposed seven knowledge graph embedding models: Pyke, Shallom, ConEx, QMult, ConvO, OMult, and ConvO. Our models reached new state-of-the-art performances in the type prediction, relation prediction and link prediction problems. Our experiments suggest that ensemble learning consistently improved link prediction performance across models on benchmark datasets. Our experiments also show that exploiting semantics (e.g., excluding an entity not being an element the range of a relation from a set of possible predictions) can be effectively used to improve link prediction performance without addition computation.
- In Chapter 7, we proposed a novel technique (KronE) based on the Kronecker decomposition that improve the parameter efficiency of knowledge graph embedding models, while preserving their effectiveness in the link prediction problem. Our experiments show that applying KronE on a knowledge graph embedding model improve the robustness against noisy triples/randomly generated triples in the link prediction problem.

- In Chapter 8, we proposed a parameter ensemble technique (PPE) to improve the generalization performance of a knowledge graph embedding model in the link prediction problem with virtually no additional computational cost. Our extensive experiments on benchmark datasets with different knowledge graph embedding models suggest that constructing a parameter ensemble model by maintaining a weighted average of parameters during training improves the performance across models and datasets.

- In Chapter 9, we proposed a knowledge graph embedding model (NeRo) to learn embeddings for description logic concepts tailored towards concept learning. Therein, we formulated the learning problem as a multi-label classification problem from a set of description logic individuals/entities to pre-selected description logic concepts. Our experiments on benchmark datasets with 770 learning problems firmly suggest that NeRo significantly outperforms the state-of-the-art models in terms of F_1 scores, the number of explored concepts, and the total runtime.

- In Chapters 10 and 11, we presented three software frameworks developed within this thesis. In Chapter 10, we developed the DICE embedding open-source software framework to learn embeddings for knowledge graphs. Therein, we showed that the DICE Embedding framework can be easily used on large knowledge graphs. Importantly, the DICE Embedding framework allows practitioners to deploy a pre-trained knowledge graph embedding model without writhing a single line of code. In Chapter 11, we developed the Ontolearn open-source software framework to tackle the class expression learning problem. The Ontolearn open-source software framework includes many class expression learning models.

- In Chapter 12, we introduced a scientific use case of leveraging a knowledge graph embedding model in concept learning. Therein, we presented a scientific use case of using pre-trained a knowledge graph embedding model in the context of concept learning. Therein, we reformulated the concept learning problem as a reinforcement learning problem. A quasi-ordered concept space is represented with a pre-trained a knowledge graph embedding model. By this, our novel deep Q agent (Drill) effectively tackled the concept learning problem.

- In Chapter 13, we introduced an industrial use cases based on our scientific contributions and software frameworks. Therein, we introduced an industrial use case, where our industrial partners in the BMWi-funded RAKI project (01MD19012D) effectively used Drill, ConEx, and the DICE Embedding framework in the skill learning problem. Their results suggest that our contributions can be used to tackle class expression learning problem on tabular data.

Here, we provide a summary of our findings.

- Chapter 3: Incorporating domain knowledge through a similarity function in the learning process and designing it within a physical system governed by the Hook's Law accelerates the embedding learning process, i.e., it reduces the runtimes. Selecting such similarity function for Pyke tailored towards a particular downstream task improves the performance across datasets.

- Chapter 4: Using a deep neural network is not necessary to reach the new state-of-the-art relation prediction performance on benchmark knowledge graphs. A shallow neural network (Shallom) can accurately predict relations between two entities. This is an important finding due to the following reasons: (1) The total energy consumption of the training, hyperparameter optimization and deployment phases for a deep neural network is substantially larger than for a shallow neural network. (2) Since deep neural networks are more prone to suffer from overfitting, elaborately designed hyperparameter optimization setup is often required to generalize well on unseen data. (3) Impact of selected hyperparameters (e.g. applied parameter initialization and activation normalization techniques and dropout rates for each layer) is magnified. In the knowledge graph embedding research, these three points were often left unstudied. Considering these three aspects, Shallom can be a good choice to reach competitive performance for a large-scale applications. Reported results in two industrial applications based on large knowledge graphs corroborate our findings [1].

- Chapter 5: Combining two state-of-the-art models into a single composite model with additive or multiplicative connections lead to new state-of-the-art performance in the link prediction problem across datasets.

[1]The BMWi-funded project RAKI (01MD19012D) and BMBF-funded project DAIKIRI (01IS19085).

A particular aspect of our proposed combination allows the learning procedure to decide the importance of each models in the propose composite model (ConEx). This is an important findings as most knowledge graph embedding models do not come with such flexibility, i.e., their structure is often fixed. Finally, applying 2D convolution operation on complex numbers $\mathbb{C}$ in the context of knowledge graph embedding models were not extensively studied. These finding influenced our later works.

- Chapter 6: Applying 2D convolution operations on Quaternions $\mathbb{H}$ and Octonion $\mathbb{O}$ leads to better link prediction than applying it on $\mathbb{R}$ across datasets. Yet, the slight improvement in the effectiveness often comes with a cost of increased runtimes.

- Chapter 7: Entity and relation embedding matrices can be considered as compressed matrices and during the training, compressed embedding vectors of entities and relations can be decompressed to compute the training loss. Via our technique (KronE), embeddings of entities and relations are not plainly retrieved but reconstructed on the fly. Applying the Kronecker product ensures that elementwise interactions between three embedding vectors are extended with interactions within each embedding vector. This implicitly reduces redundancy in embedding vectors and encourages feature reuse. Hence, learning compressed knowledge graph embeddings reduces the memory requirements and makes the model more robust against noise in the input data. Yet, using KronE to learn compressed embeddings via Kronecker product comes with the cost of increased runtimes.

- Chapter 8: Although prediction averaging (a simple form of ensemble learning) improves the link prediction performance across models and datasets, it comes with three disadvantages: the computational overhead of training multiple models and increased latency and memory requirements at test time. Our findings suggest that these disadvantages can be alleviated by averaging model parameters instead of averaging predictions of models. During training, our approach (PPE) maintains a running weighted average of the model parameters at each epoch interval. By this, the drawbacks of prediction averaging are can be alleviated and the performance of a single model can be improved.

This is an important finding as PPE can be applied on many knowledge graph embedding models and improve the model performance with virtually no additional cost.

- Chapter 9: Learning embeddings for concepts in expressive description logics substantially improves the performance (particularly runtimes) in the concept learning problem. To best of our knowledge, learning embeddings of $\mathcal{ALC}$ description logic has not yet been studied. Although our approach (NERO) can be find accurate concepts without exploration (hence reduced runtimes), it is not complete in the concept learning problem. Our findings suggest that performing the search for a goal concept in a continuous vector space, instead of a quasi-ordered symbolic space can further accelerate the learning process. By this, concept learning problem can be tackled in large-scale applications.

- Chapters 12 and 13: A quasi-ordered infinite description logic concept space can be represented by a pre-trained knowledge graph embedding model. To best of our knowledge, representing $\mathcal{ALC}$ concept space in a continuous domain has not yet been studied. Through transforming the symbolic concept space to a continuous space, the search of a goal concept can be steered by a non-myopic guide implemented by a deep Q-network.

15
Future Works

In this chapter, we briefly discuss our future works. Until the end of the nineteenth century, the word *geometry* was understood as being equivalent to the Euclidean geometry. Yet, this monopoly ceased with the construction of the non-Euclidean geometries such as Hyperbolic geometry and Elliptic geometry. Felix Klein then unified the study of geometry by studying invariants (Klein, 1893). By focusing on the invariant properties of a geometry under some class of transformations, the relationships between different geometries became apparent. This approach is known as the *Erlangen Programme*. Bronstein et al. (2021) applied the Erlangen Programme mindset to unify "a veritable zoo of deep neural network architectures". Therein, various neural network architectures (e.g., convolutional, recurrent, graph neural networks as well as transformers) are unified under the Geometric Deep Learning framework. By this unification, the relationship between different architectures can be elucidated, hence a possible reinvention or rebranding of the same concepts can be avoided. Arguably, such a unification is also needed to unify "a veritable zoo of knowledge graph embedding models".

Most knowledge graph embedding models, including the models elucidated in this thesis are designed to learn embeddings for entities and relations in a pre-determined normed division algebra, e.g., $\mathbb{R}$, $\mathbb{C}$, $\mathbb{H}$, and $\mathbb{O}$. Most of these models can be unified under *a feature composition operator* followed by *an approximation operator* in a respective selected algebra. For instance, DistMult, ComplEx, QMult and OMult can be unified under a category, where an element-wise multiplication operation is used as a feature

composition operator and an inner product is used as an approximation operator in $\mathbb{R}$, $\mathbb{C}$, $\mathbb{H}$, and $\mathbb{O}$, respectively. It can be shown that this unification holds for many other knowledge graph embedding model, including TransE, RotatE, and MuRE (Bordes et al., 2013; Sun et al., 2019; Balažević et al., 2019a). In future, we plan to follow the Erlangen Programme mindset to establish a principle to unify knowledge graph embedding models. Specifically, we will investigate Clifford algebras $Cl_{p,q}$ and design a knowledge graph embedding model that parameterizes the algebra within which it operates. By this, a knowledge graph embedding model will be endowed with the capability of selecting a particular algebra (e.r. $\mathbb{R}, \mathbb{C}, \mathbb{H}$ or $\mathbb{H} \oplus \mathbb{H}$) within which it learns embeddings for knowledge graphs.

This approach to modeling knowledge graph embeddings gives rise to new means of adding supplementary features–in particular time–to embeddings. The veracity of an assertion can be modeled as a time interval. For instance, the veracity of the assertion *the chancellor of Germany is Angela Merkel* is confined to the closed time interval 2005–2021, whereas the veracity of the assertion *the birth place of Stephen Hawking is Oxford, United Kingdom* can be regarded as a half-infinite interval, whose starting point is the birthday of Stephen Hawking. Although the time line plays an important role in the underlying assumption of knowledge graphs–hence knowledge graph embeddings–, this aspect have not yet been extensively studied in the knowledge graph embedding research domain. Recently, there has been a growing interesting in learning embeddings for knowledge graphs containing temporal information about assertions. Yet, at the time being, it is unclear how to update a pre-trained knowledge graph embedding model only on those assertions, whose veracities either begin to be true or cease to be true. Achieving this non-trivial goal is important to decrease the energy consumption of training models periodically. In future, we also plan to follow the Erlangen Programme mindset to establish a principle for the unification of knowledge graph embedding models in learning the temporal aspect of assertions. We conjecture that incorporating the temporal aspect of assertions as an additional subspace into the direct sum of all subspaces of $Cl_{p,q}$. Finally, our unified approach to knowledge graph embeddings may also serve as a catalyst for learning embeddings on dynamic knowledge graphs and their possible alignments. At the time being, it is unclear how to update embeddings of entities based on assertions stored in different knowledge graphs.

In future, we plan to investigate techniques to establish a principle for the unification of knowledge graph embedding models trained on different knowledge graphs.

List of Figures

List of Tables

References

Ali, M., M. Berrendorf, C. T. Hoyt, L. Vermue, S. Sharifzadeh, V. Tresp, and J. Lehmann. PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Embeddings. *Journal of Machine Learning Research*, 22(82):1–6, 2021.

Allen, C., I. Balazevic, and T. Hospedales. Interpreting Knowledge Graph Relation Representation from Word Embeddings. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=gLWj29369lW.

Arakelyan, E., D. Daza, P. Minervini, and M. Cochez. Complex Query Answering with Neural Link Predictors. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=Mos9F9kDwkz.

Ba, J. L., J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Baader, F., D. Calvanese, D. McGuinness, P. Patel-Schneider, D. Nardi, et al. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

Baez, J. The octonions. *Bulletin of the American Mathematical Society*, 2002.

Balazevic, I., C. Allen, and T. 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 5185–5194, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1522. URL https://aclanthology.org/D19-1522.

Balažević, I., C. Allen, and T. Hospedales. Multi-relational Poincaré graph embeddings. In *Advances in Neural Information Processing Systems*, pages 4465–4475, 2019a.

Balažević, I., C. Allen, and T. M. Hospedales. Hypernetwork knowledge graph embeddings. In *International Conference on Artificial Neural Networks*, pages 553–565. Springer, 2019b.

Balažević, I., C. Allen, and T. M. Hospedales. Tucker: Tensor factorization for knowledge graph completion. *arXiv preprint arXiv:1901.09590*, 2019c.

Balog, K. and T. Kenter. Personal knowledge graphs: A research agenda. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*, pages 217–220, 2019.

Bansal, T., D.-C. Juan, S. Ravi, and A. McCallum. A2n: Attending to neighbors for knowledge graph inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4387–4392, 2019.

Bengio, Y., R. Ducharme, and P. Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.

Bengio, Y., A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8): 1798–1828, 2013.

Bigerl, A., F. Conrads, C. Behning, M. A. Sherif, M. Saleem, and A.-C. Ngonga Ngomo. Tentris – A Tensor-Based Triple Store. In *The Semantic Web – ISWC 2020*, pages 56–73. Springer International Publishing, 2020. ISBN 978-3-030-62419-4. URL https://papers.dice-research.org/2020/ISWC_Tentris/iswc2020_tentris_public.pdf.

Bin, S., L. Bühmann, J. Lehmann, and A.-C. Ngonga Ngomo. Towards SPARQL-based induction for large-scale RDF data sets. In *ECAI*, 2016.

Bin, S., P. Westphal, J. Lehmann, and A. Ngonga. Implementing scalable structured machine learning for big data in the SAKE project. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1400–1407. IEEE, 2017.

Bishop, C. M. Training with noise is equivalent to Tikhonov regularization. *Neural computation*, 7(1):108–116, 1995.

Bishop, C. M. and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Bordes, A., N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

Bottou, L. and O. Bousquet. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20, 2007.

Bronstein, M. M., J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Broscheit, S., D. Ruffinelli, A. Kochsiek, P. Betz, and R. Gemulla. LibKGE - A Knowledge Graph Embedding Library for Reproducible Research. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 165–174, 2020.

Brown, T., B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Bühmann, L., J. Lehmann, and P. Westphal. DL-Learner—A framework for inductive learning on the Semantic Web. *Journal of Web Semantics*, 39:15–24, 2016.

Bühmann, L., J. Lehmann, P. Westphal, and S. Bin. DL-Learner Structured Machine Learning on Semantic Web Data. In *WWW*, 2018.

Burnett, M. Explaining AI: fairly? well? In *Proceedings of the 25th International Conference on Intelligent User Interfaces*, pages 1–2, 2020.

Cai, H., V. W. Zheng, and K. C.-C. Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

Campello, R. J., D. Moulavi, and J. Sander. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 2013.

Cao, Z., Q. Xu, Z. Yang, X. Cao, and Q. Huang. Dual quaternion knowledge graph embeddings. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6894–6902, 2021.

Caruana, R. *Multitask learning*. Springer, 1998.

Chami, I., A. Wolf, D.-C. Juan, F. Sala, S. Ravi, and C. Ré. Low-dimensional hyperbolic knowledge graph embeddings. *arXiv preprint arXiv:2005.00545*, 2020.

Costabello, L., S. Pai, C. L. Van, R. McGrath, N. McCarthy, and P. Tabacof. AmpliGraph: a Library for Representation Learning on Knowledge Graphs, March 2019. URL https://doi.org/10.5281/zenodo.2595043.

Das, R., S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, and A. McCallum. Go for a Walk and Arrive at the Answer: Reasoning Over Paths in Knowledge Bases using Reinforcement Learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Demir, C. and A.-C. N. Ngomo. Hardware-agnostic computation for large-scale knowledge graph embeddings. *Software Impacts*, 13:100377, 2022. ISSN 2665-9638. doi: https://doi.org/10.1016/j.simpa.2022.100377. URL https://www.sciencedirect.com/science/article/pii/S2665963822000811.

Demir, C. and A.-C. Ngonga Ngomo. A physical embedding model for knowledge graphs. In Wang, X., F. A. Lisi, G. Xiao, and E. Botoeva, editors, *Joint International Semantic Technology Conference*, pages 192–209, Cham, 2020. Springer, Springer International Publishing. ISBN 978-3-030-41407-8.

Demir, C. and A.-C. Ngonga Ngomo. Convolutional Complex Knowledge Graph Embeddings. In Verborgh, R., K. Hose, H. Paulheim, P.-A. Champin, M. Maleshkova, O. Corcho, P. Ristoski, and M. Alam, editors, *The Semantic Web*, pages 409–424, Cham, 2021a. Springer International Publishing. ISBN 978-3-030-77385-4.

Demir, C. and A.-C. Ngonga Ngomo. DRILL–Deep Reinforcement Learning for Refinement Operators in ALC. *arXiv preprint arXiv:2106.15373*, 2021b.

Demir, C. and A.-C. Ngonga Ngomo. Out-of-Vocabulary Entities in Link Prediction. *arXiv preprint arXiv:2105.12524*, 2021c.

Demir, C. and A.-C. Ngonga Ngomo. Learning Permutation-Invariant Embeddings for Description Logic Concepts. *arXiv preprint arXiv:2303.01844*, 2023.

Demir, C., D. Moussallem, S. Heindorf, and A.-C. Ngonga Ngomo. Convolutional Hypercomplex Embeddings for Link Prediction. In Balasubramanian, V. N. and I. Tsang, editors, *Proceedings of The 13th Asian Conference on Machine Learning*, volume 157 of *Proceedings of Machine Learning Research*, pages 656–671. PMLR, 17–19 Nov 2021a. URL https://proceedings.mlr.press/v157/demir21a.html.

Demir, C., D. Moussallem, and A.-C. Ngonga Ngomo. A shallow neural model for relation prediction. In *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, pages 179–182, Los Alamitos, CA, USA, jan 2021b. IEEE Computer Society. doi: 10.1109/ICSC50631.2021.00038. URL https://doi.ieeecomputersociety.org/10.1109/ICSC50631.2021.00038.

Demir, C., A. Himmelhuber, Y. Liu, A. Bigerl, D. Moussallem, and A.-C. Ngonga Ngomo. Rapid Explainability for Skill Description Learning. In *International Semantic Web Conference*. Springer, 2022a. URL https://ceur-ws.org/Vol-3254/.

Demir, C., J. Lienen, and A.-C. Ngonga Ngomo. Kronecker Decomposition for Knowledge Graph Embeddings. In *Proceedings of the 33rd ACM Conference on Hypertext and Social Media*, HT '22, page 1–10, New York, NY, USA, 2022b.

Association for Computing Machinery. ISBN 9781450392334. doi: 10.1145/3511095.3531276. URL https://doi.org/10.1145/3511095.3531276.

Dettmers, T., P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Diethe, T., T. Borchert, E. Thereska, B. Balle, and N. Lawrence. Continual learning in practice. *arXiv preprint arXiv:1903.05202*, 2019.

d'Amato, C. Machine learning for the semantic web: Lessons learnt and next research directions. *Semantic Web*, 11(1):195–203, 2020.

Edalati, A., M. Tahaei, A. Rashid, V. P. Nia, J. J. Clark, and M. Rezagholizadeh. Kronecker Decomposition for GPT Compression. *arXiv preprint arXiv:2110.08152*, 2021.

Eder, J. S. Knowledge graph based search system, June 21 2012. US Patent App. 13/404,109.

Edwards, A. D., H. Sahni, R. Liu, J. Hung, A. Jain, R. Wang, A. Ecoffet, T. Miconi, C. Isbell, and J. Yosinski. Estimating Q(s,s') with Deep Deterministic Dynamics Gradients. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 2825–2835. PMLR, 2020.

Falcon, W. et al. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3:6, 2019.

Fanizzi, N., C. d'Amato, and F. Esposito. DL-FOIL concept learning in description logics. In *International Conference on Inductive Logic Programming*, pages 107–121. Springer, 2008.

Fanizzi, N., G. Rizzo, and C. d'Amato. Boosting DL concept learners. In *ESWC*, 2019.

Fawzi, A., M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R Ruiz, J. Schrittwieser, G. Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

Firmansyah, A. F., D. Moussallem, and A.-C. Ngonga Ngomo. GATES: Using Graph Attention Networks for Entity Summarization. In *Proceedings of the 11th on Knowledge Capture Conference*, pages 73–80, 2021.

Gao, H., Z. Wang, and S. Ji. Kronecker attention networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 229–237, 2020.

Goller, C. and A. Kuchler. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 347–352. IEEE, 1996.

Goodfellow, I., Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

Graham, A. *Kronecker products and matrix calculus with applications*. Courier Dover Publications, 2018.

Guo, Y., Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):158–182, 2005.

Hamilton, W., P. Bajaj, M. Zitnik, D. Jurafsky, and J. Leskovec. Embedding logical queries on knowledge graphs. *Advances in neural information processing systems*, 31, 2018.

Hamilton, W. R. LXXVIII. On quaternions; or on a new system of imaginaries in Algebra: To the editors of the Philosophical Magazine and Journal. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1844.

Hastie, T., R. Tibshirani, J. H. Friedman, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

He, K., X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Heindorf, S., L. Blubaum, N. Dusterhus, T. Werner, V. Golani Nandkumar, C. Demir, and A.-C. Ngonga Ngomo. EvoLearner: Learning Description Logics with Evolutionary Algorithms. In *WWW*, pages 818–828. ACM, 2022.

Himmelhuber, A., S. Grimm, T. Runkler, and S. Zillner. Ontology-Based Skill Description Learning for Flexible Production Systems. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 975–981. IEEE, 2020.

Hitzler, P., F. Bianchi, M. Ebrahimi, and M. K. Sarker. Neural-symbolic integration and the Semantic Web. *Semantic Web*, 2020.

Hogan, A., E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, J. E. L. Gayo, S. Kirrane, S. Neumaier, A. Polleres, et al. Knowledge graphs. *arXiv preprint arXiv:2003.02320*, 2020.

Holzinger, A., G. Langs, H. Denk, K. Zatloukal, and H. Müller. Causability and explainability of artificial intelligence in medicine. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(4):e1312, 2019.

Huang, G., Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Huang, Z., M.-F. Chiang, and W.-C. Lee. LinE: Logical Query Reasoning over Hierarchical Knowledge Graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 615–625, 2022.

Ioffe, S. and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Ji, S., S. Pan, E. Cambria, P. Marttinen, and P. S. Yu. A Survey on Knowledge Graphs: Representation, Acquisition and Applications. *arXiv preprint arXiv:2002.00388*, 2020.

Jiao, X., Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

Joulin, A., E. Grave, P. Bojanowski, M. Nickel, and T. Mikolov. Fast linear model for knowledge graph embeddings. *arXiv preprint arXiv:1710.10881*, 2017.

Jumper, J., R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

Kazemi, S. M. and D. Poole. Simple embedding for link prediction in knowledge graphs. In *Advances in neural information processing systems*, pages 4284–4295, 2018.

Keskar, N. S., D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Kingma, D. P. and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Klein, F. Vergleichende Betrachtungen über neuere geometrische Forschungen. *Mathematische Annalen*, 43(1):63–100, 1893.

Kouagou, N., S. Heindorf, C. Demir, and A.-C. N. Ngomo. Neural class expression synthesis. *arXiv preprint arXiv:2111.08486*, 2021.

Kouagou, N., S. Heindorf, C. Demir, and A.-C. Ngonga Ngomo. Learning Concept Lengths Accelerates Concept Learning in ALC. In *Nineteenth Extended Semantic Web Conference - Research Track*. Springer, 2022a.

Kouagou, N., S. Heindorf, C. Demir, and A.-C. Ngonga Ngomo. Learning Concept Lengths Accelerates Concept Learning in ALC. In *European Semantic Web Conference*, pages 236–252. Springer, 2022b.

Krizhevsky, A., I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

Krizhevsky, A., I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Kulmanov, M., M. A. Khan, and R. Hoehndorf. DeepGO: predicting protein functions from sequence and interactions using a deep ontology-aware classifier. *Bioinformatics*, 34(4):660–668, 2018.

Lacroix, T., N. Usunier, and G. Obozinski. Canonical tensor decomposition for knowledge base completion. In *International Conference on Machine Learning*, pages 2863–2872. PMLR, 2018.

Le, Q. and T. Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

LeCun, Y., L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.

LeCun, Y., Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Lee, J., Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019.

Lehmann, J. *Learning OWL class expressions*, volume 22. IOS Press, 2010.

Lehmann, J. and P. Hitzler. A Refinement Operator Based Learning Algorithm for the ALC Description Logic. In *International Conference on Inductive Logic Programming*, pages 147–160. Springer, 2007.

Lehmann, J. and P. Hitzler. Concept learning in description logics using refinement operators. *Machine Learning*, 2010.

Lehmann, J., S. Auer, L. Bühmann, and S. Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, 2011.

Lesort, T., V. Lomonaco, A. Stoian, D. Maltoni, D. Filliat, and N. Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information fusion*, 58:52–68, 2020.

Lin, X. V., R. Socher, and C. Xiong. Multi-Hop Knowledge Graph Reasoning with Reward Shaping. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3243–3253, Brussels, Belgium, 2018. Association for Computational Linguistics.

Lopez-Paz, D. and M. Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

Malyshev, S., M. Krötzsch, L. González, J. Gonsior, and A. Bielefeldt. Getting the most out of Wikidata: semantic technology usage in Wikipedia's knowledge graph. In *International Semantic Web Conference*, pages 376–394. Springer, 2018.

Manning, C., P. Raghavan, and H. Schütze. Introduction to information retrieval. *Natural Language Engineering*, 2010.

Melo, A., H. Paulheim, and J. Völker. Type prediction in RDF knowledge bases using hierarchical multilabel classification. In *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics*, page 14. ACM, 2016.

Mikolov, T., K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013b.

Minervini, P., M. Bošnjak, T. Rocktäschel, S. Riedel, and E. Grefenstette. Differentiable reasoning on large knowledge bases and natural language. In *Proceedings of the AAAI conference on artificial intelligence*, pages 5182–5190, 2020.

Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Mohamed, S. K., V. Novácek, P.-Y. Vandenbussche, and E. Muñoz. Loss Functions in Knowledge Graph Embedding Models. *DL4KG@ ESWC*, 2377:1–10, 2019.

Mondal, S., S. Bhatia, and R. Mutharaju. EmEL++: Embeddings for εL++ description logic. In *AAAI Spring Symposium on Combining Machine Learning and Knowledge Engineering*. CEUR-WS, 2021.

Moussallem, D., M. Wauer, and A.-C. Ngonga Ngomo. Machine translation using semantic web technologies: A survey. *Journal of Web Semantics*, 51:1–19, 2018.

Moussallem, D., A.-C. Ngonga Ngomo, P. Buitelaar, and M. Arcan. Utilizing knowledge graphs for neural machine translation augmentation. In *Proceedings of the 10th International Conference on Knowledge Capture*, pages 139–146, 2019.

Moussallem, D., D. Gnaneshwar, T. C. Ferreira, and A.-C. Ngonga Ngomo. NABU–Multilingual Graph-Based Neural RDF Verbalizer. In *International Semantic Web Conference*, pages 420–437. Springer, 2020.

Murphy, K. P. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Nguyen, D. Q., T. D. Nguyen, D. Q. Nguyen, and D. 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 2018a. Association for Computational Linguistics. doi: 10.18653/v1/N18-2053. URL https://aclanthology.org/N18-2053.

Nguyen, Q., M. C. Mukkamala, and M. Hein. Neural networks should be wide enough to learn disconnected decision regions. *arXiv preprint arXiv:1803.00094*, 2018b.

Nickel, M., V. Tresp, and H.-P. Kriegel. A three-way model for collective learning on multi-relational data. In *Icml*, 2011.

Nickel, M., V. Tresp, and H.-P. Kriegel. Factorizing yago: scalable machine learning for linked data. In *Proceedings of the 21st international conference on World Wide Web*, pages 271–280, 2012.

Nickel, M., K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

Nickel, M., L. Rosasco, and T. Poggio. Holographic embeddings of knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Noy, N., Y. Gao, A. Jain, A. Narayanan, A. Patterson, and J. Taylor. Industry-scale Knowledge Graphs: Lessons and Challenges: Five diverse technology companies show how it's done. *Queue*, 17(2):48–75, 2019.

Onuki, Y., T. Murata, S. Nukui, S. Inagi, X. Qiu, M. Watanabe, and H. Okamoto. Predicting relations of embedded RDF entities by Deep Neural Network. In *International Semantic Web Conference (Posters, Demos & Industry Tracks)*, 2017.

Onuki, Y., T. Murata, S. Nukui, S. Inagi, X. Qiu, M. Watanabe, and H. Okamoto. Relation prediction in knowledge graph by Multi-Label Deep Neural Network. *Applied Network Science*, 4(1):20, 2019.

pandas development team, T. pandas-dev/pandas: Pandas, February 2020. URL https://doi.org/10.5281/zenodo.3509134.

Pennington, J., R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Prechelt, L. Early stopping—but when? *Neural networks: tricks of the trade: second edition*, pages 53–67, 2012.

Rashid, A., V. Lioutas, and M. Rezagholizadeh. MATE-KD: Masked Adversarial TExt, a Companion to Knowledge Distillation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, pages 1062–1071. Association for Computational Linguistics, 2021.

Ren, H. and J. Leskovec. Beta Embeddings for Multi-Hop Logical Reasoning in Knowledge Graphs. In *Neural Information Processing Systems*, 2020.

Ren, H., W. Hu, and J. Leskovec. Query2box: Reasoning over Knowledge Graphs in Vector Space using Box Embeddings. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJgr4kSFDS.

Ren, H., H. Dai, B. Dai, X. Chen, D. Zhou, J. Leskovec, and D. Schuurmans. Smore: Knowledge graph completion and multi-hop reasoning in massive knowledge graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1472–1482, 2022.

Riedmiller, M. Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In *European conference on machine learning*, pages 317–328. Springer, 2005.

Ristoski, P. and H. Paulheim. Rdf2vec: Rdf graph embeddings for data mining. In *International Semantic Web Conference*, pages 498–514. Springer, 2016.

Robbins, H. and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Rudolph, S. Foundations of description logics. In *Reasoning Web International Summer School*, pages 76–136. Springer, 2011.

Ruffinelli, D., S. Broscheit, and R. Gemulla. You CAN teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*, 2019.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Saleem, M., M. R. Kamdar, A. Iqbal, S. Sampath, H. F. Deus, and A.-C. Ngonga Ngomo. Big linked cancer data: Integrating linked tcga and pubmed. *Journal of web semantics*, 27:34–41, 2014.

Saltelli, A., P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola. Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index. *Computer Physics Communications*, 181(2):259–270, 2010.

Sanh, V., L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Schmidt-Schauß, M. and G. Smolka. Attributive concept descriptions with complements. *Artificial intelligence*, 48(1):1–26, 1991.

Schockaert, S., Y. Ibanez-Garcia, and V. Gutierrez-Basulto. A Description Logic for Analogical Reasoning. In Zhou, Z.-H., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2040–2046. International Joint Conferences on Artificial Intelligence Organization, 8 2021a. doi: 10.24963/ijcai.2021/281. URL https://doi.org/10.24963/ijcai.2021/281. Main Track.

Schockaert, S., Y. Ibanez-Garcia, and V. Gutierrez-Basulto. A Description Logic for Analogical Reasoning. In Zhou, Z.-H., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2040–2046. International Joint Conferences on Artificial Intelligence Organization, 8 2021b. doi: 10.24963/ijcai.2021/281. URL https://doi.org/10.24963/ijcai.2021/281. Main Track.

Shen, W., J. Wang, and J. Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27 (2):443–460, 2014.

Shi, B. and T. Weninger. Proje: Embedding projection for knowledge graph completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Silva, A. A. M. d., M. Röder, and A.-C. Ngonga Ngomo. Using Compositional Embeddings for Fact Checking. In *International Semantic Web Conference*, pages 270–286. Springer, 2021.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Strubell, E., A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in NLP. *arXiv preprint arXiv:1906.02243*, 2019.

Sun, Z., Z.-H. Deng, J.-Y. Nie, and J. Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2019.

Sun, Z., H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*, 2020.

Sutskever, I., J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

Sutskever, I., O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

Sutton, R. S. and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Tahaei, M. S., E. Charlaix, V. P. Nia, A. Ghodsi, and M. Rezagholizadeh. KroneckerBERT: Learning Kronecker Decomposition for Pre-trained Language Models via Knowledge Distillation. *arXiv preprint arXiv:2109.06243*, 2021.

Thoma, S., A. Rettinger, and F. Both. Towards holistic concept representations: Embedding relational knowledge, visual attributes, and distributional word semantics. In *International Semantic Web Conference*. Springer, 2017.

Tieleman, T. and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4 (2):26–31, 2012.

Tran, A. C., J. Dietrich, H. W. Guesgen, and S. Marsland. Parallel Symmetric Class Expression Learning. *J. Mach. Learn. Res.*, 18:64:1–64:34, 2017.

Trouillon, T. and M. Nickel. Complex and holographic embeddings of knowledge graphs: a comparison. *arXiv preprint arXiv:1707.01475*, 2017.

Trouillon, T., J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR, 2016.

Trouillon, T., C. R. Dance, É. Gaussier, J. Welbl, S. Riedel, and G. Bouchard. Knowledge graph completion via complex tensor factorization. *The Journal of Machine Learning Research*, 18(1):4735–4772, 2017.

Valeriani, A. S. Runtime Performances Benchmark for Knowledge Graph Embedding Methods. *arXiv preprint arXiv:2011.04275*, 2020.

Van Loan, C. F. The ubiquitous Kronecker product. *Journal of computational and applied mathematics*, 123(1-2):85–100, 2000.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vaswani, S., A. Mishkin, I. Laradji, M. Schmidt, G. Gidel, and S. Lacoste-Julien. Painless stochastic gradient: Interpolation, line-search, and convergence rates. *Advances in neural information processing systems*, 32, 2019.

Vink, R., S. de Gooijer, A. Beedie, J. van Zundert, G. Hulselmans, C. Grinstead, M. E. Gorelli, M. Santamaria, D. Heres, ibENPC, J. Leitao, M. van Heerden, C. Jermain, R. Russell, C. Pryer, A. G. Castellanos, J. Goh, M. Wilksch, illumination k, M. Conradt, L. Brannigan, Y. R. Tan, elbaro, J. Peek, N. Stalder, S. H. Welling, A. Gregory, paq, and J. Keller. pola-rs/polars: Python Polars 0.16.11, March 2023. URL https://doi.org/10.5281/zenodo.7699984.

Wang, Q., Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

Westphal, P., S. Vahdati, and J. Lehmann. A Simulated Annealing Meta-heuristic for Concept Learning in Description Logics. In *International Conference on Inductive Logic Programming*, pages 266–281. Springer, 2021.

Wu, J.-N. Compression of fully-connected layer in neural network by kronecker product. In *2016 Eighth International Conference on Advanced Computational Intelligence (ICACI)*, pages 173–179. IEEE, 2016.

Xian, Y., Z. Fu, S. Muthukrishnan, G. de Melo, and Y. Zhang. Reinforcement Knowledge Graph Reasoning for Explainable Recommendation. In Piwowarski, B., M. Chevalier, É. Gaussier, Y. Maarek, J. Nie, and F. Scholer, editors, *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, pages 285–294. ACM, 2019.

Xiao, H., M. Huang, L. Meng, and X. Zhu. SSP: Semantic Space Projection for Knowledge Graph Embedding with Text Descriptions. In *AAAI*, volume 17, pages 3104–3110, 2017.

Xie, R., Z. Liu, J. Jia, H. Luan, and M. Sun. Representation learning of knowledge graphs with entity descriptions. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016a.

Xie, R., Z. Liu, and M. Sun. Representation Learning of Knowledge Graphs with Hierarchical Types. In *IJCAI*, pages 2965–2971, 2016b.

Xiong, W., T. Hoang, and W. Y. Wang. DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 564–573, Copenhagen, Denmark, 2017. Association for Computational Linguistics.

Xu, J., X. Sun, Z. Zhang, G. Zhao, and J. Lin. Understanding and improving layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Yang, B., W. Yih, X. He, J. Gao, and L. Deng. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In Bengio, Y. and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6575.

Yao, L., C. Mao, and Y. Luo. KG-BERT: BERT for knowledge graph completion. *arXiv preprint arXiv:1909.03193*, 2019.

Zaheer, M., S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Zhang, A., Y. Tay, S. Zhang, A. Chan, A. T. Luu, S. C. Hui, and J. Fu. Beyond Fully-Connected Layers with Quaternions: Parameterization of Hypercomplex Multiplications with $1/n$ Parameters. In *International Conference on Learning Representations*, 2021.

Zhang, F., N. J. Yuan, D. Lian, X. Xie, and W.-Y. Ma. Collaborative knowledge base embedding for recommender systems. In *KDD*, pages 353–362. ACM, 2016.

Zhang, S., Y. Tay, L. Yao, and Q. Liu. Quaternion knowledge graph embeddings. In *Advances in Neural Information Processing Systems*, pages 2731–2741, 2019.

Zhang, W., J. Chen, J. Li, Z. Xu, J. Z. Pan, and H. Chen. Knowledge Graph Reasoning with Logics and Embeddings: Survey and Perspective. *arXiv preprint arXiv:2202.07412*, 2022.

Zheng, G., F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li. DRN: A Deep Reinforcement Learning Framework for News Recommendation. In Champin, P., F. L. Gandon, M. Lalmas, and P. G. Ipeirotis, editors, *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 167–176. ACM, 2018.

Zhou, J., G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Zhu, Z., Z. Zhang, L.-P. Xhonneux, and J. Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems*, 34:29476–29490, 2021.