THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

École doctorale : MSTII - Mathématiques, Sciences et technologies de l'information, Informatique
Spécialité : Mathématiques et Informatique
Unité de recherche : Centre de recherche Inria de l'Université Grenoble Alpes

Intégration neuro-symbolique de l'extraction de connaissances et du raisonnement sur les graphs.

Neural-Symbolic Integration of knowledge extraction and reasoning on graph-structured data

Présentée par :

Luisa WERNER

Direction de thèse :

Nabil LAYAIDA DIRECTEUR DE RECHERCHE, INRIA	Directeur de thèse
Pierre GENEVES Directeur de recherche, CNRS Délégation Alpes	Co-directeur de thèse

Rapporteurs :

Fatiha SAïS
PROFESSEURE DES UNIVERSITES, Université Paris Saclay
Farouk TOUMANI
PROFESSEUR DES UNIVERSITES, Université Blaise Pascal- Clermont-Ferrand

Thèse soutenue publiquement le **11 décembre 2024**, devant le jury composé de :

Nabil LAYAIDA, DIRECTEUR DE RECHERCHE, Centre INRIA de l'Université Grenoble Alpes	Directeur de thèse
Pierre GENEVèS, DIRECTEUR DE RECHERCHE, CNRS Délégation Alpes	Co-directeur de thèse
Fatiha SAïS, PROFESSEURE DES UNIVERSITES, Université Paris Saclay	Rapporteure
Farouk TOUMANI, PROFESSEUR DES UNIVERSITES, Université Blaise Pascal-Clermont-Ferrand	Rapporteur
Massih-Reza AMINI, PROFESSEUR DES UNIVERSITES, Université Grenoble Alpes	Examinateur
Stefania Gabriela DUMBRAVA, MAITRESSE DE CONFERENCES, ENSIIE - École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise & Institut Polytechnique de Paris	Examinatrice
Axel-Cyrille NGONGA NGOMO, FULL PROFESSOR, Paderborn University	Examinateur

To Line van den Berg

Abstract

Graph-structured data has gained significant attention in recent years due to its ability to encode relationships between entities, making it a rich data structure capable of representing complex patterns, long-chain dependencies, and cyclical structures. However, graph-structured data, such as knowledge graphs, presents significant challenges that must be addressed to fully unlock its potential. Since graphs are often large, partly unstructured, and incomplete, algorithms designed for graphs need to efficiently process sparse data.

In parallel, AI research has seen a surge in the development of deep learning methods, while symbolic methods have reached a level of stability with fewer noticeable breakthroughs. Nonetheless, the explainability of symbolic methods, which are based on logic and prior knowledge, has the potential to complement the strengths of sub-symbolic methods in pattern matching, robustness, and scalability. As a result, research on neuro-symbolic methods has gained attention, aiming to unify symbolic and sub-symbolic AI approaches.

This thesis explores how neuro-symbolic methods can be applied to graph-structured data to solve reasoning tasks, such as knowledge graph completion, more efficiently and reliably. The primary focus is on how prior knowledge, such as ontologies, can be leveraged to enhance the performance of purely sub-symbolic methods. First, this thesis investigates how prior knowledge can be integrated into a graph neural network through differentiable neural layers based on fuzzy logic. Specifically, it examines the scalability and applicability of this technique across different types of graphs. Second, a neuro-symbolic method is proposed that injects knowledge into knowledge graph embeddings by integrating a semantic reasoning engine.

Résumé

Les données structurées sous forme de graphes, telles que les graphes de connaissances, ont attiré l'attention ces dernières années en raison de leur capacité à encoder les relations entre les entités, ce qui en fait une structure de données riche capable de représenter des modèles complexes, des dépendances à longue chaîne et des structures cycliques. Cependant, les données structurées par des graphes posent quelques difficultés majeures qu'il faut surmonter pour en exploiter le potentiel. Les graphes étant souvent de grande taille, non structurés et incomplets, les algorithmes appliqués aux graphes doivent être capables de traiter efficacement des données éparses et non structurées en grille.

Parallèlement, la recherche en IA a connu ces dernières années une explosion du développement des méthodes d'apprentissage profond, tandis que les méthodes symboliques ont été reléguées à l'arrière-plan. Néanmoins, la capacité d'explication des méthodes symboliques basées sur la logique et les connaissances préalables peut compléter la force des méthodes sub-symboliques en matière de correspondance des formes, de robustesse et d'évolutivité. C'est pourquoi la recherche sur les méthodes neuro-symboliques a attiré l'attention, dans le but d'unifier les méthodes symboliques et sub-symboliques de l'IA.

Cette thèse étudie comment les méthodes neuro-symboliques peuvent être employées sur des données structurées en graphe afin que des tâches de raisonnement telles que la complétion de graphes de connaissances puissent être résolues de manière plus efficace et plus fiable. L'accent est mis en particulier sur la question de savoir comment les connaissances préalables, par exemple sous la forme d'ontologies, peuvent être exploitées pour améliorer le comportement des méthodes purement sub-symboliques. On étudie comment les connaissances préalables peuvent être intégrées dans un réseau neuronal par le biais de couches neuronales différentiables basées sur la logique floue. L'extensibilité et l'applicabilité de cette technique à différents types de graphes sont notamment examinées. En outre, une méthode neuro-symbolique est proposée pour injecter des connaissances dans les graphes de connaissances en intégrant un moteur de raisonnement sémantique.

Acknowledgements

This PhD would not have been possible without all the help and guidance I have received over the past few years. First of all, I would like to thank my supervisors, Nabil and Pierre, for their support throughout my thesis. Thank you for trusting me and giving me space for my ideas, while always having an open ear for my questions. I would also like to thank the entire Tyrex team. Thank you for making INRIA such a nice place to work. I always felt welcome in the team and enjoyed our conversations during coffee and lunch breaks. Special thanks to Amela and Sarah for their support and advice, and to my office partner Laurent (without whose support in the form of Madeleines I would not have got through the occasional afternoon slump). Lucia, thank you for your advice on this thesis and, more generally, on finding a path in research. I would also like to thank Damien, my CSI expert, for his valuable feedback during my thesis. Thanks also to Jérôme Euzenat for working with me on my first paper and giving me the courage that a reproducibility paper can make it to an A* conference. Thanks to the members of the jury for their evaluation and feedback. I would also like to thank the University of Grenoble Alpes and the MIAI (ANR-19-P3IA-0003) for funding my research.

My time in Grenoble would not have been the same without the company of so many wonderful people. Those who woke up with me at 5am to ski before work on untouched powder and freshly groomed slopes. Those who have hiked, bivouacked and bikepacked every pass and peak in the region: A big shout out to Aurélien, Annet, Denis, Charlotte, Ronan, Jonas, Caro, Yann, Josh, Benoit, Elias, Pauline, ... and simply everyone who considers themselves part of the *Grelous* crew! Johanna, thank you for proving that camping at -15°C is possible and for your patience with me climbing! Special thanks to my roommate Joseba, who not only shared a flat with me, but also countless hours on the bike saddle and on the skis. Carla, starting bikepacking together was an adventure in itself, and our yearly trip has been the constant in my bikepacking life ever since - thank you for that! To Caro Meyer, even though you've been far away, you've always been there for me. Thank you for all your personal advice and for being a constant source of support.

My heartfelt thanks to my flatmate Line. I miss you and wish I could have spent more time with you. You were not only the one who encouraged me to start a PhD. It's also your voice I hear when I need someone to tell me I can do it.

A big thank you to my family, especially my sister Clara and my mum and dad for making it all possible. You are the reason I have been able to pursue any degree I wanted. You have taught me to never give up and to always support the decisions that have brought me here. Thank you for being not only my parents, but also my friends. I am so glad that

you were convinced by the charm of France and visited me regularly, and even more so that you cycled the epic "grands cols" with me.

Finally, a big thank you to my partner Fabian. Thank you for being by my side, supporting me at every step of this journey and teaching me countless things I'm proud to have learned, such as cross-country skiing technique, bike mechanics, ski touring conversions, setting up our own server, how to pronounce "Charmant Som", making cappucchino with latte art... and I'm sure a complete list wouldn't fit on a single page. I'm so grateful for all the moments and adventures we have shared together.

Publications

Parts of this research have been disseminated through publications and presentations at the following academic conferences [206, 207, 208]:

- Reproduce, Replicate, Reevaluate.The Long but Safe Way to Extend Machine Learning Methods. Luisa Werner, Nabil Layaïda, Pierre Genevès, Jérôme Euzenat, Damien Graux. AAAI 2024 - The 38th Annual AAAI Conference on Artificial Intelligence, Feb 2024, Vancouver, Canada

- Knowledge Enhanced Graph Neural Networks for Graph Completion. Luisa Werner, Nabil Layaïda, Pierre Genevès, Sarah Chlyah. The 10th IEEE International Conference on Data Science and Advanced Analytics, Oct 2023, Thessaloniki, Greece

  (This work has also been presented at the 1st International Workshop on Knowledge-Based Compositional Generalization (KBCG). The 32nd International Joint Conference On Artificial Intelligence. August 2023. Macao. S.A.R.)

- Neuro-Symbolic Integration for Reasoning and Learning on Knowledge Graphs. Luisa Werner. AAAI 2024 - The 38th Annual AAAI Conference on Artificial Intelligence, Feb 2024, Vancouver, Canada. Doctoral Consortium.

Software

- Reproduce Replicate Reevaluate. The Long but Safe Way to Extend Machine Learning Methods.
  `https://gitlab.inria.fr/tyrex-public/reproducibility-aaai24`
- Knowledge Enhanced Graph Neural Networks (KeGNN).
  `https://gitlab.inria.fr/tyrex-public/kegnn`
- Knowledge Enhanced Neural Networks on Large-scale Graphs.
  `https://gitlab.inria.fr/tyrex-public/scalable_ke`
- RuleKGE. Learning Rule-Injected Knowledge Graph Embeddings on Incomplete Knowledge Graphs.
  `https://gitlab.inria.fr/tyrex-public/rulekge`

Contents

List of Figures

List of Tables

Part I.

Introduction

Introduction

Graph-structured data is ubiquitous in various real-world applications such as e-commerce, natural sciences and web search engines. In essence, graphs connect nodes through edges, thus expressing relational information between entities. Graphs can be seen as a generalisation of other data formats, such as images and language. In contrast, they do not have any structural regularities. This makes them a powerful and versatile data structure for capturing dependencies in an extensible format.

In particular, knowledge graphs encode information as a set of facts of the form *(head, relation, tail)* which express the interaction between two entities through a relation. For example, sellers and users on online marketplaces in an e-commerce network can be considered as entities and their transactions as relations, e.g. *(userA, buys, productB)*. Graphs differ not only in the type of information they encode, but also in their format and size. In some graphs, nodes or edges are enriched with feature information, for example in the form of text or image data. In addition, graphs can be described by an ontology, which contains rules that the facts in the graph should follow.

However, graphs also pose significant challenges. First, they are often extracted automatically from multiple data sources with limited human intervention. As a result, noise and errors from real-world data can be introduced into the resulting graph, leading to incorrect facts. Second, graphs are often incomplete. The incompleteness may not only be due to the extraction process, but may also result from the sparse structure of graphs. Knowledge graphs typically store only information that is known to be true. False facts are usually not stored explicitly. Storing all facts about a world would result in a large number of facts with potentially redundant information. As a result, it is unclear whether unspecified facts are missing or false.

Given the ubiquity of graph structures, several research areas are actively exploring techniques from symbolic AI and deep learning to exploit them. Tasks of interest are for example information retrieval, question answering, or link prediction. First, symbolic AI uses reasoning techniques to infer new facts or generate proofs for queries. Symbolic AI approaches are based on symbolic representations of knowledge, such as in logic programming. These methods typically assume that the facts in the graph are true and do not consider uncertainty or misinformation. Despite being interpretable, reasoning on symbolic representations is subject to scalability problems for large graphs, which limits its applicability.

Deep learning, in contrast, relies on vector representations and therefore falls into the category of sub-symbolic AI. Over the past decade, deep learning made breakthroughs in a

wide range of tasks across multiple domains. Powered by neural networks and their ability to find patterns in raw data with minimal human intervention, deep learning methods learn from noisy data while being scalable at inference. In the deep learning community as well, graph-structured data is receiving increasing attention. The focus lies on building neural network models that can process the relational structure of graph data. In this context, *graph neural networks* refine vector representations through permutation invariant operations. Often starting from attributed graphs, the node or edge representations are iteratively updated with respect to the local graph neighbourhood. Another approach is *knowledge graph embeddings*. They aim to represent a graph in the vector space by capturing its entities and relations geometrically. As a result, tasks such as link prediction and query answering can be solved efficiently based on distance functions in the Euclidean space. These methods are robust to noise and scale well at inference.

However, the guarantees offered by symbolic approaches are often lost when the graph is translated into vector space, where predictable inference is no longer ensured. Furthermore, prior knowledge, sometimes explicitly formulated in an ontology, is neglected by most approaches. As a result, the predictions in the vector space are not necessarily consistent with the prior knowledge. Furthermore, deep learning models, including those on graphs, are black-box models. Deep neural networks typically contain a large number of learnable parameters. This makes their internal mechanisms intransparent and complicates the interpretation of their predictions.

Recently, the research field of *neuro-symbolic integration* has gained interest since it acknowledges the complementary advantages and disadvantages of sub-symbolic and symbolic AI. In this context, neuro-symbolic AI seeks to combine both paradigms in order to find models that are robust, interpretable, knowledge-aware, scalable and accurate. The goal is to potentially pave the way to trustworthy general artificial intelligence. Frameworks such as DeepProblog [139] and Logic Tensor Networks [14] are promising when it comes to integrating the pattern matching capabilities of deep neural networks with symbolic reasoning about knowledge representations in formal logic. They are used in various areas such as image recognition, visual scene understanding and multi-label classification. There, the neuro-symbolic approach results in higher interpretability and faster convergence compared to purely neural models.

While some neuro-symbolic approaches and concepts are promising for small and confined tasks, their application to large graphs and complex knowledge is still uncertain. In particular, the scalability of state-of-the-art neuro-symbolic methods to large graphs with millions of nodes and edges has not yet been adequately addressed. While the field of knowledge graph embeddings explores how to capture inference patterns such as taxonomies, symmetries, or range and domain constraints, the encoding of complex knowledge is less explored. Furthermore, there is less understanding of how to achieve predictable inference with knowledge graph embedding, or how to incorporate soft rules.

Outline

This thesis investigates how neuro-symbolic AI techniques can be applied to graphs, with the objective of jointly exploiting prior knowledge in formal logic and real-valued vector representations. This thesis is structured as follows. It is divided into two main parts: I. the state-of-the-art and II. the contribution part.

At the beginning of the first part, in **Chapter 1**, preliminary concepts are presented. In **Chapter 2** relevant concepts and methods from symbolic AI are introduced, followed by concepts and methods from sub-symbolic AI and in particular deep learning in **Chapter 3**. In **Chapter 4** methods from neuro-symbolic AI are presented, evaluated and compared.

The second part contains the contributions of this thesis. **Chapter 5** studies the reproducibility of Knowledge Enhanced Neural Networks. Given the relevance of the work on *Knowledge Enhanced Neural Networks* [46] to this thesis, the experiments are reimplemented, reproduced, replicated, and reevaluated. These steps aim to ensure the reliability of the reimplementation for further extensions. General lessons for improving the reproducibility of machine learning methods are summarised. **Chapter 6** introduces the method *Knowledge Enhanced Graph Neural Networks (KeGNN)*, which incorporates knowledge enhancement layers into graph neural networks. Unlike previous work where these layers were used with Multi-Layer Perceptrons, KeGNN leverages graph neural networks. This allows relational information to be captured and increases the capacity of the model. The effectiveness of KeGNN is tested through experiments on various graph datasets. **Chapter 7** builds on the previous chapter and deals with the applicability of knowledge enhancement layers to large graphs, where memory requirements significantly increase. The sampling method called *Restrictive Neighbourhood Sampling (RNS)* is introduced to make KeGNN applicable to large graphs. In addition, experiments with knowledge enhancement layers are carried out on benchmark datasets from the Open Graph Benchmark [93]. **Chapter 8** presents the neuro-symbolic method *RuleKGE* that focuses on knowledge graph embedding training under the open-world assumption. It combines a symbolic reasoner to generate positive and negative facts. They are integrated into the training of knowledge graph embeddings for link prediction. The effectiveness of the method is shown in several experiments on the Family dataset [120].

Part II.

State of the art

1. Preliminaries

This section introduces preliminary concepts and definitions related to graph-structured data and logic that are relevant to this thesis.

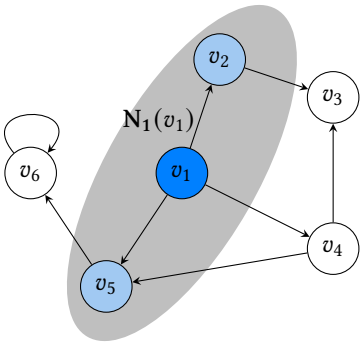
1.1. Graph-structured Data

Graph-structured data has recently received a lot of attention. While traditional standard data formats tend to neglect interactions between entities, these dependencies can be explicitly modelled in graphs. This makes them a powerful and flexible data format. Different types of graphs and relevant concepts are formally introduced in this section.

Definition 1.1.1 (Graph). *A graph is a tuple* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *where* $\mathbf{V} = \{v_1, \ldots, v_n\}$ *is a finite set of* n *nodes and* $\mathbf{E} = \{e_1, \ldots e_m\}$ *is a finite set of* m *edges.*

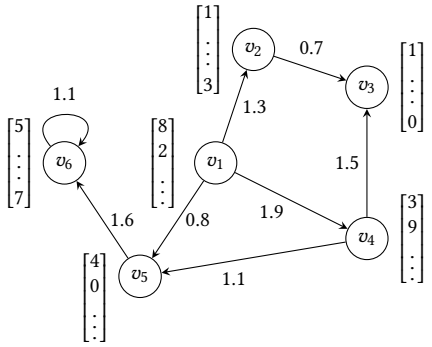
An edge is a tuple (v_i, v_j) that connects the two nodes $v_i \in \mathbf{V}$ and $v_j \in \mathbf{V}$. The edges of a graph are described as adjacency matrix.

Definition 1.1.2 (Adjacency Matrix). *Given a graph* $\mathbf{G}$*, the adjacency matrix is denoted as* $\mathbf{A} \in \{0, 1\}^{n \times n}$*. The* (i, j)*-th entry* $\mathbf{A}_{ij}$ *indicates whether an edge exists between the nodes* v_i *and* v_j *(*$\mathbf{A}_{ij} = 1$*) or not (*$\mathbf{A}_{ij} = 0$*).*



$$\mathbf{A} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Figure 1.1.: A directed graph is shown on the left. The first-order neighbourhood of the node v_1 is the set $\mathbf{N}_1(v_1) = \{v_1, v_2, v_5\}$ and is marked in grey. The degree of v_1 is $\deg(v_1) = 3$. The adjacency matrix of the graph is shown on the right.



$$\mathbf{A} = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{bmatrix} 0 & 1.3 & 0 & 1.9 & 0.8 & 0 \\ 0 & 0 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.5 & 0 & 1.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.6 \\ 0 & 0 & 0 & 0 & 0 & 1.1 \end{bmatrix} \end{array}$$

Figure 1.2.: A directed, weighted graph, node-attributed is shown on the left and its adjacency matrix on the right.

A graph can be *directed* or *undirected*. In directed graphs an edge $(v_i, v_j) \in \mathbf{E}$ leads from node v_i to v_j. The edges in undirected graphs have no orientation. The adjacency matrix $\mathbf{A}$ of an undirected graph is symmetric, meaning $\mathbf{A}_{ij} = \mathbf{A}_{ji}$ for any pair of nodes $v_i \in \mathbf{V}$ and $v_j \in \mathbf{V}$.

Definition 1.1.3 (Neighbourhood). *Given a graph $\mathbf{G}$, the k-order neighborhood of a node $v_i \in \mathbf{V}$ with $k \in \mathbb{N}_0$ is the set of nodes $\mathbf{N}_k(v_i)$ with $\mathbf{N}_k(v_i) = \{v_j \in \mathbf{V} | \operatorname{dist}(v_i, v_j) \leq k\}$. $\operatorname{dist}(v_i, v_j)$ represents the shortest path length in terms of number of edges from node v_i to node v_j. If no path exists between v_i and v_j, v_j is not included in $\mathbf{N}_k(v_i)$. The 0-order neighbourhood $\mathbf{N}_0(v_i)$ of node v_i is the node v_i itself.*

The k-order neighbourhood over a set of nodes $\mathbf{V}$ is $\mathbf{N}_k(\mathbf{V}) = \{\mathbf{N}_k(v) | v \in \mathbf{V}\}$.

Definition 1.1.4 (Node Degree). *Given an undirected graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with adjacency matrix $\mathbf{A}$, the node degree of a node $v_i \in \mathbf{V}$ is $\deg(v_i) = \sum_j \mathbf{A}_{ij}$. Given a directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ with adjacency matrix $\mathbf{A}$, the node degree of a node $v_i \in \mathbf{V}$ is $\deg(v_i) = \sum_j \mathbf{A}_{ij} + \sum_j \mathbf{A}_{ji}$.*

A directed graph with an adjacency matrix is visualised in Figure 1.1. Node and edge features can provide additional real-valued information about nodes and edges.

Definition 1.1.5 (Attributed Graph). *A node-attributed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{X}_\mathbf{V})$ is enriched with node features $\mathbf{X}_\mathbf{V} \in \mathbb{R}^{n \times d_\mathbf{V}}$ with feature dimension d_V. An edge-attributed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{X}_\mathbf{E})$ is enriched with edge features $\mathbf{X}_\mathbf{V} \in \mathbb{R}^{n \times d_\mathbf{V}}$ with feature dimension d_E.*

Single-valued edge features are also called *edge weights*. A graph with edge weights is called *weighted*. In the case of a weighted graph, the adjacency matrix contains the edge weights. A graph is called *node-labelled* if c different labels $\mathbf{Y} \in \mathbb{R}^{n \times c}$ are assigned to each node. A weighted and node-attributed graph is shown in Figure 1.2.

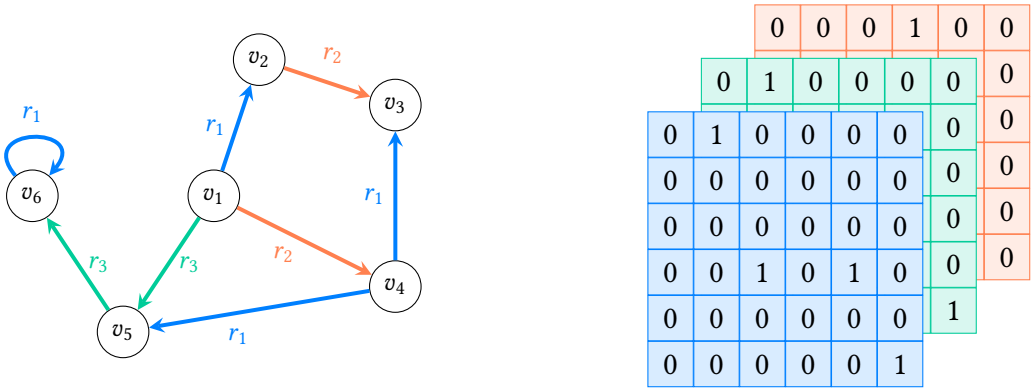


Figure 1.3.: A heterogeneous graph with the relations $\mathcal{T}_E = \{r_1, r_2, r_3\}$ is shown on the left. The edges are denoted as relation-specific adjacency matrices on the right.

Definition 1.1.6 (Heterogeneous Graph). *A heterogeneous graph is defined as a* $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathcal{T}_V, \mathcal{T}_E)$*, where the nodes* $\mathbf{V}$ *and edges* $\mathbf{E}$ *are associated with type functions* $\mathbf{f}_n : \mathbf{V} \mapsto \mathcal{T}_\mathbf{V}$ *and* $\mathbf{f}_e : \mathbf{E} \mapsto \mathcal{T}_\mathbf{E}$*. They assign node and edge types to nodes and edges respectively.* $\mathcal{T}_\mathbf{V}$ *and* $\mathcal{T}_\mathbf{E}$ *are finite sets of node and edge types.*

Heterogeneous graphs with multiple edge types are also called *multi-relational graphs*. In the case of attributed and heterogeneous graphs, the edge and node feature dimension $d_\mathbf{E}$ and $d_\mathbf{V}$ can differ with node and edge type. An example of a heterogeneous graph is shown in Figure 1.3.

Definition 1.1.7 (Homogeneous Graph). *In a homogeneous graph* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$*, all nodes in* $\mathbf{V}$ *are instances of the same node type, and all edges in* $\mathbf{E}$ *are instances of the same edge type.*

1.1.1. Tasks on Graphs

A graph $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{Y})$ with labels $\mathbf{Y}$ is split into a training graph and a test graph. *Inductive* or *Transductive* learning [171] can be applied, see Figure 1.4.

Definition 1.1.8 (Inductive Learning). *In inductive learning, the training graph is* $\mathbf{G}_{train} = (\mathbf{V}_{train}, \mathbf{E}_{train}, \mathbf{Y}_{train})$ *and the test graph is* $\mathbf{G}_{test} = (\mathbf{V}_{test}, \mathbf{E}_{test}, \mathbf{Y}_{test})$*, where* $\{v | v \in \mathbf{V}_{train} \wedge v \in \mathbf{V}_{test}\} = \emptyset$*.*

Definition 1.1.9 (Transductive Learning). *In transductive learning, the training graph is* $\mathbf{G}_{train} = (\mathbf{V}, \mathbf{E}, \mathbf{Y}_{train})$ *and* $\mathbf{V}_{test} \subset \mathbf{V}_{train}$*.*

In the transductive setting, the entire graph is available at training, including edges between train and test nodes. The labels of the test nodes are masked. In the inductive setting, no edges between nodes in the training and the test graph exist.

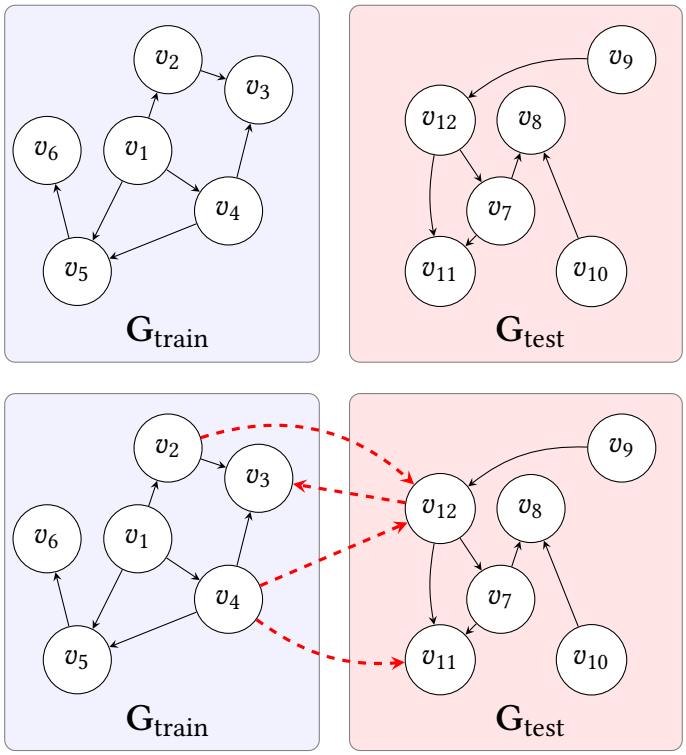


Figure 1.4.: In inductive learning, as shown above, the training graph $\mathbf{G}_{\text{train}}$ and the test graph $\mathbf{G}_{\text{test}}$ are not connected with edges. In transductive learning, as shown below, edges between nodes of the training and test graph *do* exist and are drawn in red.

Several graph-related tasks are subject to research such as node classification, graph clustering, link prediction and graph classification. This thesis focuses on the tasks of *node classification* and *link prediction* [138].

Definition 1.1.10 (Node Classification). The task of node classification is to leverage $\mathbf{G}_{\text{train}}$ with the labels $\mathbf{Y}_{\text{train}}$ to learn a function that assigns labels to the nodes $\mathbf{V}_{\text{test}}$ in the test graph.

Definition 1.1.11 (Link Prediction). Given a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, let $\mathbf{M}$ denote the set of all possible edges between nodes in $\mathbf{V}$. The set $\mathbf{E}'$ contains the unobserved edges between the nodes and is denoted as $\mathbf{E}' = \mathbf{M}/\mathbf{E}$. The goal of link prediction is to leverage the existing edges $\mathbf{E}$ and nodes $\mathbf{V}$ in $\mathbf{G}$ to learn a function that predicts the edges in $\mathbf{E}'$ that are likely to exist.

1.1.2. Knowledge Graph

Knowledge graphs [91, 171, 104] are considered as multi-relational, directed graph structures where nodes are called *entities* and edge types are called *relations*. They store

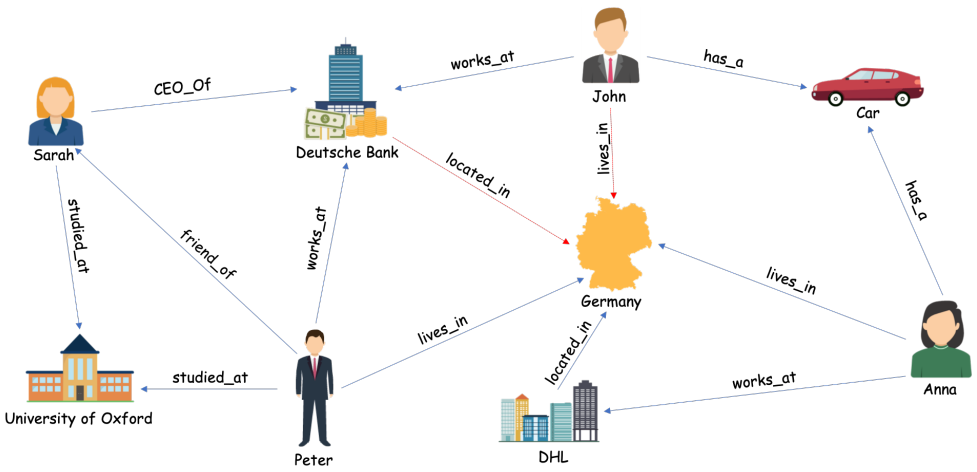


Figure 1.5.: Illustration of an example knowledge graph. The figure is taken from [9].

information in the form of facts, which relations between pairs of entities. The components of a fact are also known as *(head, relation, tail)* or *(subject, predicate, object)*. Entities represent real-world objects or abstract concepts, while relations describe the relationships between them. Well-known knowledge graphs are for example Wikidata [210], YAGO [191] and Freebase [25]. The illustration of a knowledge graph is shown in Figure 1.5.

Definition 1.1.12 (Knowledge Graph). *A knowledge graph is defined as tuple* $\mathbf{K} = (\mathcal{E}, \mathcal{R}, \mathcal{F})$*, where* $\mathcal{E}$ *is a set of entities,* $\mathcal{R}$ *a set of relations and* $\mathcal{F} \subset (\mathcal{E} \times \mathcal{R} \times \mathcal{E})$ *is a set of facts. Facts are stored as triples* (e_h, r, e_t) *with head and tail entities* $e_h, e_t \in \mathcal{E}$ *and a relation* $r \in \mathcal{R}$ *between them.*

The terms *one-to-one*, *one-to-many*, and *many-to-one describe* different characteristics of relations between entities. A one-to-one relation connects a head entity to exactly one tail entity, e.g. `PassportID(h,t)`. A one-to-many relation associates a single head entity with multiple tail entities, e.g. `hasFather(h,t)`. A many-to-one relation is the opposite, e.g. `bornIn(h,t)`. A many-to-many relation links multiple head entities to multiple tail entities, e.g. `sibling(h,t)`.

While knowledge graphs can in principle be enriched with node and edge features [91, 2, 71, 171], the vast majority of benchmarks consist only of a set of facts and are not attributed with node features [149, 191, 31].

Knowledge graphs are known to be incomplete. For example, 71% of individual persons in Freebase do not have an edge to a birthplace and 78% do not have an edge to a nationality [209]. In Wikidata, 50% of the artists do not have a birthplace [209]. This incompleteness has several reasons. First, knowledge graphs are typically built semi-automatically from existing knowledge based on heuristics, extraction patterns or crowdsourcing methods, which can be prone to errors [104, 21, 3, 102, 222]. Second, it would not be reasonable to explicitly store every known fact about a world [39]. This would lead to overhead and

redundant information, when many facts are inferred [79]. The notation of the edges as facts, as opposed to an adjacency matrix, is suitable for supporting incompleteness. Knowledge graphs usually consist only of explicitly known facts, which are assumed to be valid [9], but do not guarantee to contain all true facts. Different assumptions can be made about the implicit facts. Under the *Closed World Assumption (CWA)*, facts that are not contained in the graph are assumed to be false. Under the *Open World Assumption (OWA)*, facts that are not contained in the graph are not known to be true or false [157].

1.1.3. Opportunities

Knowledge graphs have received increasing attention in various domains, as well as from academia and from industry [104]. Knowledge graphs such as YAGO [191], Freebase [25], Wikidata [210] and Nell [31] contain general knowledge, while others such as BioRDF [29] store domain knowledge in the field of life sciences. In addition to open knowledge graphs, there are many enterprise knowledge graphs in industry, such as LinkedIn [86], Bloomberg [145] and IBM [158]. Knowledge graphs, compared to basic data, particularly have the following benefits.

Relational Semantics. Graph-structured data expresses relationships between entities. This feature makes them a rich data source to model context, complex patterns, long-range dependencies and cyclic structures [91].

Flexibility. Since knowledge graphs store information as facts, the definition of a rigid schema can be postponed [91]. This enables graphs to be extended flexibly and to incorporate information from various data sources. Moreover, in contrast to an adjacency matrix, the triple format facilitates the representation of incomplete knowledge and supports efficient storage of large data.

Applications. Given their advantageous properties, knowledge graphs have proven useful across a variety of applications, including question answering [44, 54, 152], complex query answering [171], recommender systems [224, 198], natural language processing [230, 133, 109], biomedical research, fraud detection, and more. In general, they are a central representation to many information systems on the web [39].

1.1.4. Challenges

Despite these advantages, graphs present some challenges that need to be addressed in order to unlock their potential.

Sparse Representation. In contrast to a dense adjacency matrix, facts are sparse information and difficult to manipulate for machine learning algorithms mainly designed for dense matrices [201]. While data such as images and text have a clear grid-like structure, knowledge graphs do not have a start, end or order. This is also known as the *geometric deep learning problem* [27, 212]. To be applicable to graphs, mathematical operations such as convolutions or pooling need to be generalized to non-grid structures. Many algorithms in the field of graph neural networks [114, 196, 80] are only applicable to homogeneous graphs.

Incompleteness. The incompleteness of knowledge graphs is also a major challenge for many algorithms, making it difficult to model them as dense matrices. However, most machine learning algorithms are adapted to dense matrix data that is explicit and complete. For example, common graph neural network methods [114, 212] assume that the graph structure is complete and free of noise [96].

Large Scale. In the era of big data, graphs often contain millions or billions of edges [183]. Therefore, algorithms applied to them need to scale to such dimensions [219, 231, 93, 177]. Many traditional learning algorithms on graphs are NP-complete and are sensitive to the number of nodes [201]. Their application can be infeasible due to their complexity [171].

1.2. Logic

Logic is the systematic study of valid reasoning, inference, and argumentation. It provides a formal framework for formulating claims and analysing their validity. In essence, logic is concerned with the principles of correct reasoning that enable one to draw conclusions, and assess the coherence of statements and propositions [51]. A logical language is a formal system composed of syntax, semantics and inference rules [52]. Logic has a wide range of applications, from philosophy to artificial intelligence. This section briefly introduces *propositional logic*, *first-order logic*, *fuzzy logic* and *description logics*. An overview of the logical languages is summarized in Table 1.2.

1.2.1. Propositional Logic

Propositional logic focuses on propositions which are atoms that are either *True* or *False*, never both and never neither. It is used to represent and reason about simple statements, called *propositions*, and their logical relationships.

Signature. A signature in propositional logic is a set of propositional variables $\Sigma = \{\Phi, \Psi, \ldots\}$..

Name	DL	Description	FOL	Example
Top	$\top$	Thing	$\top, \forall x : \top(x) = \text{true}$	`⊤ ⊑ Male ⊔ Female`
Bottom	$\bot$	Nothing	$\bot, \forall x : \bot\ (x) = \text{false}$	`Male ⊓ Female ⊑⊥`
Subsumption	$A \sqsubseteq B$	A subclass of B	$\forall x : A(x) \rightarrow B(x)$	`Child ⊑ Human`
	$R \sqsubseteq S$	R subproperty of S	$\forall x \forall y : R(x, y) \rightarrow S(x, y)$	`Mother ⊑ Parent`
Equivalence	$A \equiv B$	A equivalent to B	$\forall x : A(x) \leftrightarrow B(x)$	`Person ≡ Human`
Instantiation	$A(i)$	i type A	$A(i)$	`Female(julia)`
Relations	$R(i, j)$	i related to j with R	$R(i, j)$	`Mother(julia,rob)`
Complement	$\neg A$	not A	$\neg A(x)$	`not Mother(julia,rob)`
Intersection	$A \sqcap B$	A and B	$A(x) \wedge B(x)$	`Female ⊓ Parent`
Union	$A \sqcup B$	A or B	$A(x) \vee B(x)$	`Father ⊔ Mother`
Inverse	R^-	inverse of R	$\forall x, y : R(x, y) \rightarrow R^-(y, x)$	`marriedTo ≡ marriedTo⁻`
Composition	$R \circ S$	composition of R and S	$\forall x, y, z : R(x, y) \wedge S(y, z)$	`Brother o Parent ⊑ Uncle`

Table 1.2.: Overview of the syntax and semantics in description logics (DL) and its rewriting in first-order logics (FOL). A and B are concepts in DL and unary predicates in FOL. R and S are relations in DL and binary predicates in FOL. i and j are entities in DL or constants in FOL. x, y and z are variables.

Syntax. Propositions can be combined with logical connectives. Given two propositions Φ and Ψ, the following expressions are also propositions

$$\begin{aligned} &\neg\Phi \\ &\Phi \rightarrow \Psi \\ &\Phi \wedge \Psi \\ &\Phi \vee \Psi \\ &\Phi \leftrightarrow \Psi, \end{aligned} \tag{1.1}$$

where $\wedge$ stands for conjunction, $\vee$ for disjunction, $\rightarrow$ for implication, $\leftrightarrow$ for equivalence and $\neg$ for negation.

Semantics. Given two propositions Φ and Ψ, the meaning of the connectives is defined by truth tables. In propositional logic, a statement that must be true is called a *tautology*, and a statement that must be false is called a *contradiction*.

Example 1.2.1 (Propositonal Logic). Suppose there are two propositions:

$$\begin{aligned} &\Phi : \text{It is raining} \\ &\Psi : \text{I am carrying an umbrella.} \end{aligned} \tag{1.2}$$

These propositions are used to express different relationships, e.g. $\Psi \wedge \Phi$: "It is raining and I am carrying an umbrella". This corresponds to the following truth table.

Φ	Ψ	$\Phi \wedge \Psi$
True	True	True
True	False	False
False	True	False
False	False	False

1.2.2. First-order Logic

Unlike propositional logic, first-order logic is more expressive and allows to formulate more complex relationships using quantification.

Signature. First-order logic consists of pairwise disjoint sets of predicates $\mathcal{P}$, constants $\mathcal{C}$, variables $\mathcal{V}$ and functions $\mathcal{F}$[1].

Syntax. The most basic form of a formula in first-order logic is an *atom*. An atom applies a predicate to a constant and is denoted as $r(t_1, \ldots, t_n)$, where $r \in \mathcal{P}$ is an n-ary relation and $t_1, \ldots, t_n$ are terms. A term t_i is a constant, a variable or a structured term of the form $f(t_1, \ldots, t_q)$ with functor f. Further, a positive or negative atom is called *literal*. First-order logic uses the same connectives as presented for propositional logic in Equation 1.1. In addition, the existential quantifier $\exists$ and the universal quantifier $\forall$ are defined. A variable that is within the scope of a quantifier in a formula is called *quantified* and *free* if it is not.

Given these components, atoms can be combined with logical connectives and quantifiers to form *complex formulae* φ. They can be constructed recursively using the following grammar.

$$\varphi = r\left(s_1, \ldots, s_n\right) | \neg\varphi | \varphi \wedge \psi | \varphi \rightarrow \psi | \varphi \vee \psi | \exists x \varphi \mid \forall x \varphi \tag{1.3}$$

Here, $r \in \mathcal{P}$ is an n-ary predicate and $x \in \mathcal{V}$ is a variable.

Semantics. Constants represent objects or entities and functors represent functions. Variables make abstractions about entities and predicates of the arity n describe a property or relations of n objects in a domain [50]. $\forall x \varphi(x)$ means that the expression φ is valid for all variables x. $\exists x \varphi(x)$ means that at least one object in the domain has the property φ. As in propositional logic, the meanings of the connectives in first-order logic are defined by their truth tables.

[1] This is a different notation from the set of facts $\mathcal{F}$ in a knowledge graph introduced in Section 1.1.2

Grounding. The replacement of variables by constants is called *grounding*. An atom containing only constants is called *grounded atom*. Thus, a *grounded literal* is a positive or negative grounded atom and a *grounded formula* is a formula containing only grounded atoms. A substitution of variables $x_i \in \mathcal{X}$ by constants $c_i \in C$ is defined as $\theta = \{x_1|c_1, \ldots, x_k|c_k\}$. Applying θ to a logical expression φ results in the replacement of all variables $x_i \in \theta$ in the expression by the defined constants $c_i \in \theta$ [50].

Example 1.2.2 (First-Order Logic). The following atoms are defined in first-order logic.

$$\begin{aligned} &\text{Person}(x): && x \text{ is a person.} \\ &\text{Umbrella}(x): && x \text{ is carrying an umbrella.} \\ &\text{Rain}: && \text{it is raining.} \end{aligned} \tag{1.4}$$

Then the statements "Everyone carries an umbrella when it rains" and "There is at least one person who carries an umbrella when it rains" are made.

$$\begin{aligned} &\forall \text{x: Person(x)} \ \wedge \text{Rain} \rightarrow \text{Umbrella(x)} \\ &\exists \text{x: Person(x)} \wedge \text{Rain} \wedge \text{Umbrella(x)} \end{aligned} \tag{1.5}$$

1.2.3. Fuzzy Logic

While the previously introduced logical languages only consider the Boolean truth values *True* and *False*, *Fuzzy logic* [220] is a form of many-valued logic that denotes truth values in a continuous interval of $[0, 1] \subset \mathbb{R}$. Fuzzy logic supports the concept of partial truth, where statements are neither completely true nor completely false. This allows to express vagueness and uncertainty. Fuzzy logic is based on *triangular norm theory* [117]. It is useful to model logical operators as real-valued functions that take into account continuous truth values. In the following, the t-norm and t-conorm are defined.

Definition 1.2.1 (t-norm). *The t-norm is a function* $\top : [0, 1] \times [0, 1] \mapsto [0, 1]$ *that satisfies the following properties:*

$$\begin{aligned} &\textit{Commutativity} && \top(a, b) = \top(b, a) \\ &\textit{Monotonicity} && \top(a, b) \leq \top(c, d) \textit{ if } a \leq c \textit{ and } b \leq d \\ &\textit{Associativitity} && \top(a, \top(b, c)) = \top(\top(a, b), c) \\ &\textit{Identity} && \top(a, 1) = a. \end{aligned}$$

Definition 1.2.2 (t-conorm). *The t-conorm is a function* $\top : [0, 1] \times [0, 1] \mapsto [0, 1]$ *that satisfies the properties of commutativity, monotonicity, associativity and has the neutral element* $\perp$ $(a, 0) = a$.

T-conorms are dual to t-norms. Given a t-norm $\top$, the corresponding t-conorm is defined as $\perp(a, b) = 1 - \top(1-a, 1-b)$. Some examples of t-norm functions with their corresponding t-conorms are the Gödel t-norm, the Łukasiewicz t-norm, and the product t-norm. The way they represent logical operators in fuzzy logic is shown in Table 1.3.

Expression	Product	Łukasiewicz	Gödel
$x \wedge y$	$\top_{\text{Prod}} = x \cdot y$	$\top_{\text{Łuk}} = \max(0, x + y - 1)$	$\top_{\min} = \min(x, y)$
$x \vee y$	$\perp_{\text{Prod}} = x + y - x \cdot y$	$\perp_{\text{Łuk}} = \min(1, x + y)$	$\perp_{\max} = \max(x, y)$
$x \rightarrow y$	$\left(\frac{y}{x}\right), x > 0$	$1 - x + y$	y
$\neg x$	$1 - x$	$1 - x$	$1 - x$

Table 1.3.: Overview of the Product, Łukasiewicz, Gödel t-norm and t-conorm functions for logical operators. x and y are logical expressions.

In fuzzy logic, *quantifiers* are represented as symmetric and continuous aggregation operators of the form $\text{Agg} : \bigcup_{n \in \mathbb{N}} [0,1]^n \rightarrow [0,1]$. Appropriate aggregators for $\exists$ and $\forall$ are A_{pM} and A_{pME} with $p \geq 1$ [14]. They represent the smooth maximum and minimum of n truth values $\{a_1, \ldots, a_n\} \in [0,1] \subset \mathbb{R}$.

$$\exists : A_{pM}(a_1, \ldots, a_n) = \left(\frac{1}{n} \sum_{i=1}^{n} a_i^p \right)^{\frac{1}{p}} \quad p \geqslant 1 \tag{1.6}$$

$$\forall : A_{pME}(a_1, \ldots, a_n) = 1 - \left(\frac{1}{n} \sum_{i=1}^{n} (1 - a_i)^p \right)^{\frac{1}{p}} \quad p \geqslant 1 \tag{1.7}$$

The parameter p controls the flexibility of the operator to outliers.

Soft rules. The notion of degree of truth also allows to formulate *soft rules* in contrast to *hard rules*. Hard rules are usually hand-crafted by experts and are expected to hold without exception. Soft rules are tagged with a score that corresponds to the confidence in the rule. This enables better handling of exceptions. Soft rules with confidence scores can also be automatically extracted from data [65, 146].

1.2.4. Description Logics

Description logics (DL) [12] is a family of knowledge representation languages widely used in the context of knowledge graphs. The key elements are finite sets of *entities* E, *concepts* C and *relations* R. Furthermore, the logical operators *negation* ($\neg$), *equality* ($\equiv$), *intersection* ($\sqcap$), *union* ($\sqcup$) and *logical inclusion* ($\sqsubseteq$) allow building complex formulae. Further, the *top concept* $\top$ is used to make statements about each entity and the *bottom concept* $\perp$ describes the empty set of concepts.[2] Universal ($\forall R.C$) and existential restriction ($\exists R.C$) quantify

[2]The notation of t-norm and t-conorm is distinct from the notation of the top and bottom concept in logic.

over sets of concepts. Description logics is a subset of first-order logic, where the arity of predicates is limited to two. In the terminology of first-order logic, entities correspond to constants, concepts to unary predicates and relations to binary predicates. A summary of the operators in DL and their rewriting in first-order logic is presented in Table 1.2.

```
1  John       rdf:type  Person.                    % Individuals
2  Mary       rdf:type  Person.
3  Jane       rdf:type  Person.
4  Rob        rdf:type  Person.

6  Person     rdf:type  owl:Class.
7  Male       rdf:type  owl:Class.                 % Concepts
8             rdfs:subClassOf  Person.
9  Female     rdf:type  owl:Class.
10            rdfs:subClassOf  Person.

12 hasChild   rdf:type     owl:ObjectProperty.     % Roles
13            rdfs:domain  :Parent.
14            rdfs:range   :Child.
15 isChildOf  rdf:type     owl:ObjectProperty.
16            rdfs:domain  :Person.
17            rdfs:range   :Person.

19 Male       owl:disjointWith  Female.            % TBox Axiom

21 hasChild   owl:inverseOf  isChildOf.            % RBox Axiom
```

Figure 1.6.: An example ontology describing kinship relations.

The axioms formulated in DL are categorized into (1) *assertional axioms (ABox)*, (2) *relational axioms (RBox)*, and (3) *terminological axioms (TBox)* [123]. Abox axioms refer to the facts about the instances in a knowledge graph. They encode knowledge about entities and describe the *concepts* to which individuals belong, such as `City(Paris)`, or the relationships between them, such as `ParentOf(Julia, John)`, also known as *roles*. In contrast, TBox axioms characterise general relationships between concepts and RBox axioms between relations. This allows to formulate complex relationships between relations and concepts such as symmetry, hierarchy, inclusion, transitivity or mutual exclusion.

Ontologies. An ontology is a set of TBox and RBox axioms in DL. Ontologies are formal and explicit specifications of the concepts, entities and relations that exist in a particular domain, see Section 1.1.2. In other words, they describe the semantics of the components in a knowledge graph by formalising conventions about what entities and relations mean in a domain [91]. While the facts in a knowledge graph are seen as premises, ontologies encode general rules, also known as common sense or domain knowledge. In this context, ontologies together with facts allow reasoning on knowledge graphs. This concretely means inferring new facts from existing facts and rules [106]. Ontologies are often

composed of schemas and complex formulae in OWL [205]. A schema describes the high-level semantics that the knowledge graphs follow [91] and includes, for example, range constraints for relations, formulated in RDFS [118]. Since not all knowledge graphs have a schema [170], ontology learning tasks allow to build, refine and learn ontologies [211, 60].

Example 1.2.3 (Ontology). The ontology in Figure 1.6 represents basic concepts and relations in the family domain. The entities `John, Mary, Jane` and `Bob` are instances of the concept `:Person`. The concepts `Female` and `Male` are subclasses of `Person`. The roles `isChildOf` and `hasChild` denote relationships between individuals. The TBox axiom `Male owl:disjointWith Female` states that the concepts `Male` and `Female` are disjoint, i.e. no individual is both male and female. The RBox axiom `hasChild owl:inverseOf isChildOf` asserts that the property `hasChild` is the inverse of the property `isChildOf`.

2. Symbolic Reasoning

Symbolic reasoning refers to reasoning techniques based on formal methods. Logical languages, such as first-order logic and description logics, are a cornerstone of symbolic systems. Knowledge in symbolic systems often refers to rules that are hand-crafted by experts. Symbolic methods are typically deterministic and close to human language, which makes their application explainable and understandable [88, 100]. This section introduces some prominent symbolic reasoning techniques.

2.1. Logic Programming

Logic Programming [16] is a relevant symbolic reasoning technique based on *Horn rules* [92].

Definition 2.1.1 (Horn Rule). *A horn rule is defined as* $\eta \leftarrow B(\beta_1 \wedge \ldots \wedge \beta_n)$*, where the head* η *is a singular atomic formula and the body* B *is a conjunction of atomic formulae* $\beta_1 \wedge \ldots \wedge \beta_n$ *in first-order logic.*

Each atomic formula β_i is a literal of the form $r(t_1, \ldots, t_n)$, where $r \in \mathcal{P}$ is a predicate and $t_1, \ldots, t_n$ are terms. A finite set of rules and facts is known as a *logic program*. A logic program describes a specification of possible theories in a world. The rules serve as constraints that these theories should satisfy.

Given a logic program, *reasoning* describes the process of deriving new facts from existing ones. When the head of a rule is satisfied, the tail can be inferred. An inferred fact is called the *immediate consequence* of the program and the facts. Starting with a set of atomic facts and a logic program, the rules of the program are applied repeatedly to the existing facts and inferred facts until no more new facts are derived. This state is called a *fixpoint* and the procedure is called *forward chaining*. Proof trees show how a new fact is derived from the original facts and rules of a program in a bottom-up manner.

A prominent declarative programming language is *Datalog* [28]. A program in Datalog consists of finite sets of *facts* and *rules*. While facts are grounded atoms, rules contain variables and follow the syntax of Horn rules, see Definition 2.1.1. In Datalog, rules are written as $\eta :- \beta_1, \ldots, \beta_n$. Datalog programs are used for deductive reasoning and are evaluated in a bottom-up manner. The existence of a fixpoint in Datalog is guaranteed for linear rules and rules with stratified negation [5].

Example 2.1.1 (Datalog). Consider the following Datalog program:

```
parentOf(A,B) :- childOf(B,A)
childOf("alice", "bob")
```

The program contains a rule which describes the inverse relation between the concepts `parentOf` and `childOf` and a fact which states that the constant `"alice"` is the child of the constant `"bob"`. By evaluating the Datalog program, the fact `parentOf("bob", "alice")` is an immediate consequence of the program and is inferred.

2.2. Probabilistic Logic Programming

Probabilistic Logic Programming (ProbLog) [202, 63] introduces probabilities to logic programming. A program in ProbLog consists of rules and *probabilistic facts*, where a probability $p \in [0, 1] \subset \mathbb{R}$ is attached to a ground atom f, denoted as $p :: f$ or $Pr(f) = p$. This allows to model uncertainties. Facts are treated as independent Boolean random variables that are considered as true with probability p and as false with probability $1 - p$ with $p \in [0, 1] \subset \mathbb{R}$.

Example 2.2.1 (ProbLog). The following ProbLog program indicates that a burglary or an earthquake occurs with certain probabilities. The person Mary might hear the alarm with probability of 0.5. If Mary hears the alarm, she will call the police.

```
0.1 :: burglary
0.2 :: earthquake
0.5 :: hears_alarm("mary")
alarm :- earthquark
alarm :- burglary
calls(X) :- alarm, hears_alarm(X)
```

Inference in ProbLog means returning the *success probability* q of a ground fact. The calculation of the success probability is based on proofs. A proof of a given fact f' is defined as the minimal set of input facts F that can infer f', and is denoted as $F \in \mathbb{P}(\mathcal{F})$, where $\mathbb{P}$ is the power set of all known facts. The probability of a proof $Pr(F)$ is defined as the product of the truth values p_i of the probabilistic facts f_i as

$$Pr(F) = \prod_{f_i \in F} p_i \prod_{f_i \in \mathcal{F} \setminus F} (1 - p_i) \,. \tag{2.1}$$

Given a set of proofs S_q, the success probability is the joint probability of all proofs in the set S_q: $q = Pr(S_q)$. The set of proofs for a query S_q can be determined during the bottom-up execution of the program based on a Sequential Decision Diagram (SDD) [49] by computing truth values from known facts to truth values of logic statements. This process is called *Weighted Model Counting (WMC)* [112] and is based on *semirings*. Semirings

define binary operations for disjunction $\oplus$ and conjunction $\otimes$ of sets of proofs S_1, S_2, so that the truth values of proofs in a set can be combined. For example, $S_1 \oplus S_2 = S_1 \cup S_2$ or $S_1 \otimes S_2 = \{F \mid F = F_1 \cup F_2, (F_1, F_2) \in S_1 \times S_2, F\}$, where F contains no disjunction conflict. Given these semantics, the set of proofs for a query is constructed in a bottom-up manner:

$$S_q = \bigoplus_{F \text{ derives } q} \left(\bigotimes_{f \in F} S_f \right). \tag{2.2}$$

It is shown in [112] that the bottom-up pass calculation is correct for any commutative semiring, where both the multiplication and the addition operator are associative, commutative, and have a neutral element [112].

Example 2.2.2 (ProbLog Inference). To illustrate the query evaluation in Problog, consider the previous logic program in Example 2.2.1. The query of interest is `calls("mary")`. The corresponding SDD looks as follows.

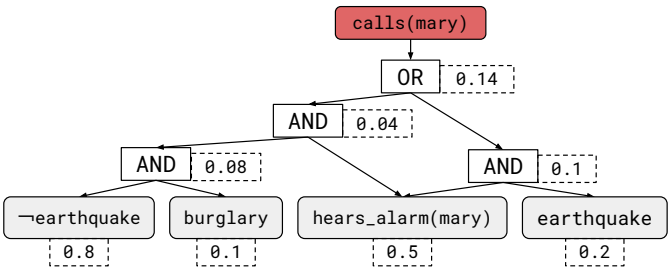


Figure 2.1.: The SDD for the burglary example. The figure is taken from [139].

The query is decomposed in its proofs under the guidance of the SDD.

$$\begin{aligned} &\texttt{calls("mary")} \\ &\leftrightarrow \texttt{hears\_alarm("mary")} \wedge \texttt{earthquake} \\ &\leftrightarrow \texttt{hears\_alarm("mary")} \wedge (\texttt{burglary} \vee \neg\texttt{earthquake}) \end{aligned} \tag{2.3}$$

Given the multiplication as operator for $\otimes$ and addition as operator for $\oplus$, the success probability of the query `calls("mary")` results in

$$P(\texttt{calls("mary")}) = 0.2 \cdot 0.5 + (0.5 \cdot 0.8 \cdot 0.1) = 0.14. \tag{2.4}$$

2.3. Inductive Logic Programming

The central task in *Inductive Logic Programming (ILP)* [153] is to induce rules from the facts in a dataset. It is based on symbolic rule mining techniques that focus on statistical metrics in the data, such as correlations, and generalise them into rules. These rules are useful for various tasks, such as identifying regularities in databases and detecting errors. Likewise,

they can be used for deductive reasoning. They can also be employed in neuro-symbolic systems that rely on input rules, but do not yet have expert knowledge available [226, 222, 40].

A widely used ILP approach is the association rule mining technique *AMIE* [65]. AMIE efficiently generates Horn rules $\eta \leftarrow B$, see Definition 2.1.1, where the head is $r(x, y)$ with relation r. Starting from a set of facts, the rules are iteratively mined according to their *support* and *confidence*.

The support of a rule is

$$\text{supp}(r(x, y) \leftarrow B) := |\{(x, y) : \exists z_1, \ldots, z_m : \vec{B} \wedge r(x, y)\}|, \tag{2.5}$$

where $z_1, \ldots, z_m$ are the variables of the rule apart from x ard y. AMIE uses the *head coverage* h_c which is based on the support and measures the proportion of the occurrence of a rule pattern in the dataset.

$$h_c(r(x, y) \leftarrow B) := \frac{\text{supp}(r(x, y) \leftarrow B)}{|\{(x', y') : r\,(x', y')\}|} \tag{2.6}$$

The denominator is the length of the set of all tuples (x', y') such that $r(x', y')$ holds.

The *confidence* measures the proportion of facts in which the rule applies to the number of cases in which the rule body is satisfied.

$$\text{conf}(r(x, y) \leftarrow B) := \frac{\text{supp}(r(x, y) \leftarrow B)}{|\{(x, y) : \exists z_1, \ldots, z_m : B\}|} \tag{2.7}$$

AMIE adapts a the confidence metric conf_{pca}, where the confidence is not normalized by the entire set of facts, but only by the set of facts that are known to be true and the facts that are assumed to be false:

$$\text{conf}_{pca}(r(x, y) \leftarrow B) := \frac{\text{supp}(r(x, y) \leftarrow B)}{|\{(x, y) : \exists z_1, \ldots, z_m, y' : \vec{B} \wedge r\,(x, y')\}|} \tag{2.8}$$

The rules generated by AMIE must exceed a head coverage threshold, a confidence threshold and match a maximum rule length. In addition, AMIE imposes a language bias as a constraint on rule creation to reduce the search space. This avoids mining rules that are unlikely to predict the existence of a fact. Several extensions to AMIE were proposed that introduce pruning strategies, approximations and parallelization techniques that accelerate rule mining [66, 124]. However, a limitation of AMIE and its extensions is the combinatorial explosion in the rule discovery process.

Furthermore, *Anytime Bottom-Up Rule Learning (AnyBURL)* [146] uses relation-based sampling to harness information for rule learning in a bottom-up manner. Starting with a given relation and an initial path length of two, it collects neighbouring relations to form rules. The sampled paths are generalised to Horn rules. The path length is increased iteratively to learn longer rules. Similar to AMIE, candidate rules are selected based on their confidence scores. There are several extensions to AnyBURL that aim to improve the quality of the rules produced. *Reinforce AnyBurl* [147] uses reinforcement learning and provides confidence and rule length as rewards. *SAFRAN* [159] uses clustering techniques to detect redundant rules.

2.4. Limitations

Although symbolic methods have clear strengths, such as interpretability, they also present some obstacles.

Scalability. A major limitation is scalability. Rule learning approaches require the exploration of a search space that grows exponentially with the number of relations. In the context of large graphs, these methods have scalability limitations [78, 197, 110, 222]. Logic programming methods such as Datalog and Problog rely on weighted model counting, which aims to aggregate the assignments of multiple positive worlds. This amounts to a MAX-SAT problem, which is NP-hard [36].

Specification of prior knowledge. Many symbolic methods, such as Problog or Datalog, require external prior knowledge expressed in a logical language. This includes general rules, but also the probability values for grounded terms in Problog, for example. The provision of handcrafted expert knowledge can be a bottleneck for the system [41]. ILP methods can extract rules with confidence scores from data. However, the reliability of the rules depends on the quality of the data and the choice of the confidence threshold parameter [222].

Noise and uncertainty. Rule-based reasoning techniques have difficulty expressing noise and uncertainty [36, 78, 222, 100, 128]. This limits the reliability of automatically generated rules and their applicability to some real-world datasets, which are known to contain noise. Since information is assumed to be true and deterministic, false information propagates in symbolic reasoning. Although some methods take confidence values or probabilities into account, the performance of the model can be affected if the model is not able to detect and to correct errors and noise. Furthermore, missing data can be a problem in symbolic reasoning methods [119].

Inflexibility. Symbolic methods are successful for static and well-defined problems and sometimes fail to generalise to problems beyond a given domain. They also lack the expressiveness to detect patterns in high-dimensional, unstructured data and necessarily rely on information at a symbolic level [70, 100].

3. Sub-symbolic Reasoning

Sub-symbolic reasoning techniques do not rely on explicit symbols or logical rules. Instead, they exploit the patterns and structure inherent in data by operating in the continuous space without resorting to explicit symbols. Geometric interpretations or distance functions are used to solve reasoning tasks such as node classification and link prediction in a sub-symbolic manner. Parameterised functions are typically learned through a training stage where a differentiable loss function is optimised. A subset of sub-symbolic methods are neural[1] methods, which refer particularly to the use of deep neural networks [88].

This section introduces relevant sub-symbolic techniques in the context of graphs. These include *knowledge graph embeddings*, which represent entities and relations in a graph as vectors in an embedding space. Furthermore, *graph neural networks* is a class of neural networks adapted to the structure of graphs.

3.1. Knowledge Graph Embeddings

The goal of knowledge graph embeddings is to learn a dense representation of the knowledge graph in a continuous low-dimensional vector space that captures the structural properties of the graph. These embeddings are useful as input for various tasks such as entity disambiguation and clustering, as well as for downstream tasks such as question answering and recommendation systems [201]. However, their most prominent use case is link prediction.

In knowledge graph embeddings methods, entities and relations are encoded as lookup matrices. In other words, given a knowledge graph $\mathbf{K} = (\mathcal{E}, \mathcal{R}, \mathcal{F})$, d-dimensinal embedding vectors for the entities in $\mathcal{E}$ and the relations in $\mathcal{R}$ with dimension d are learned through an optimization task [171]. In this way, a fact (h, r, t) is represented in the vector space as $(\mathbf{h}, \mathbf{r}, \mathbf{t})$, where $\mathbf{h}$ and $\mathbf{t}$ are the vector representations of the head and tail entities. The entity and relation embeddings are randomly initialised at the beginning of the training and optimised over several training epochs using gradient descent [111]. As a result, embeddings are found that represent the structure of entities and relations in the training graph. In knowledge graph embedding methods, a *score function* f_{score} is defined:

$$f_{score} : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightarrow \mathbb{R}. \tag{3.1}$$

[1] The terms *neuro* and *neural* are used interchangeably in the literature.

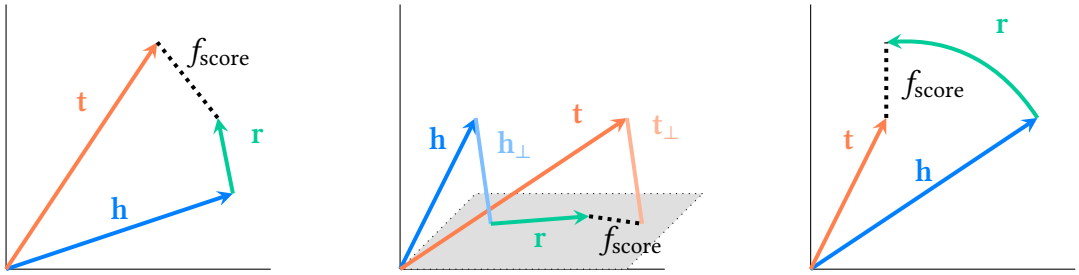


Figure 3.1.: Visualisation of the translational models TransE (left), TransH (center) and RotatE (right) in the two-dimensional space.

It takes as input the entity and relation embeddings of a fact $(\mathbf{h}, \mathbf{r}, \mathbf{t})$ and returns its *plausibility*, in other words the likelihood that the fact is true. The exact definition of the score function depends on the design of the knowledge graph embedding model. Given two facts (h_1, r_1, t_1) and (h_2, r_2, t_2), the first is considered more plausible than the second if $f_{score}(h_1, r_1, t_1) > f_{score}(h_2, r_2, t_2)$. The score function typically encodes a distance in the embedding space and is trained to assign higher scores to true facts than to false facts. True and false facts are called *positive facts* and *negative facts*.

3.1.1. Prominent Knowledge Graph Embedding Methods

In this section, some prominent knowledge graph embedding methods are presented. They differ mainly in the way they represent entities and relations and in their definition of the score function. Numerous knowledge graph embedding methods were published in recent years. For a more comprehensive overview, the following surveys can be considered [173, 91, 201, 43]. In the literature, knowledge graph embedding methods are commonly categorised into *translational models*, *semantic matching models* and *neural models*. Neural models for knowledge graph embeddings are introduced in Section 3.2 together with graph neural networks.

3.1.1.1. Translational Models

Translational models represent entities and relations of a knowledge graph as points in the Euclidean vector space and binary relations as translations or other geometric operations. The plausibility of the facts is calculated with a distance function between the projected entities.

The seminal translational model is *TransE* [26] where entities and relations are represented as vectors $\mathbf{h}, \mathbf{t}, \mathbf{r} \in \mathbb{R}^d$. Facts are modelled as translations from head to tail embeddings, so that $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$. Given a fact (h, r, t), the score function in TransE is defined as the distance between the head and tail vectors after applying the relation as translation:

$$f_{\text{score}}(h, r, t) = -\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_p, \tag{3.2}$$

where p is the L-p norm. Although TransE is scalable due to its simplicity, it has several drawbacks. First, it can only represent one-to-one relations, since it forces different entities to the same representation in many-to-one, one-to-many and many-to-many relations [204, 132]. For example, given the two facts $(h, r, t_1), (h, r, t_2) \in \mathcal{F}$, the score function returns $\mathbf{h} + \mathbf{r} \approx \mathbf{t}_1$ and at the same time $\mathbf{h} + \mathbf{r} \approx \mathbf{t}_2$. This results in $\mathbf{t_1} \approx \mathbf{t}_2$. Furthermore, TransE cannot model symmetric relations. The facts (h, r, t) and (t, r, h) can only be captured simultaneously if $\mathbf{r} = 0$.

As an extension to TransE, *TransH* [204] is proposed, which can model one-to-many, many-to-one and many-to-many relations. Each relation is represented by the normal vector of a hyperplane $\mathbf{w}_r \in \mathbb{R}^d$ and a vector $\mathbf{d}_r \in \mathbb{R}^d$ lying in the hyperplane. The head and tail vectors are at first projected in their relation-specific hyperplanes $\mathbf{h}_\perp = \mathbf{h} - \mathbf{w}_r^\top \mathbf{h} \mathbf{w}_r$ and $\mathbf{t}_\perp = \mathbf{t} - \mathbf{w}_r^\top \mathbf{t} \mathbf{w}_r$. Based on the projected representations, the score in TransH is calculated as

$$f_{\text{score}}(h, r, t) = - \|\mathbf{h}_\perp + \mathbf{d}_r - \mathbf{t}_\perp\|_2^2 . \tag{3.3}$$

TransE and TransH are visualised in Figure 3.1.

In *RotatE* [188], the entities and relations are vectors $\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{C}^d$ in the complex space. Relations are modelled as rotations from the head to the tail entity in the complex space as $\mathbf{t} = \mathbf{h} \odot \mathbf{r}$. $\odot$ denotes the element-wise Hadamard product. With norm-preserving real parts $|r_i| = 1$ for each $i \in \{1, \ldots, d\}$, r_i is expressed as $e^{i\theta_{r,i}}$, which corresponds to a counter-clockwise rotation by $\theta_{r,i}$. RotatE is visualised in Figure 3.1. The score function in RotatE is denoted as

$$f_{\text{score}}(h, r, t) = -\|\mathbf{h} \odot \mathbf{r} - \mathbf{t}\|. \tag{3.4}$$

Unlike TransE and TransH, RotatE can model symmetry by setting all rotation phases in all dimensions to multiples of π, while TransE and TransH force symmetric relations onto the same vector. However, RotatE cannot represent one-to-many, many-to-one and many-to-many relations [3].

BoxE [3] is a spatio-relational embedding model that encodes relations as regions in the embedding space (*boxes*) and entities as a tuple of two vectors $(\boldsymbol{e}, \boldsymbol{b}) \in \mathbb{R}^{2d}$, where $\mathbf{e}$ defines the *base position* of the entity and $\mathbf{b}$ defines its *translational bump* from their base position to the final embedding depending on the co-occurrence of entities in a fact. In other words, entities that co-occur in a fact translate each other. The concept of BoxE is illustrated in Figure 3.2. The entity embedding for an entity $\mathbf{e}_i \in \mathbb{R}^d$ is expressed as

$$\mathbf{e}_i^{(h,r,t)} = (\mathbf{e}_i - \mathbf{b}_i) + \mathbf{b}_h + \mathbf{b}_t. \tag{3.5}$$

The embedding of an entity is fact-dependent, so that different embeddings are introduced for an entity. This is why BoxE is suitable for modelling many-to-many, many-to-one and one-to-many relations. Relations are modelled as rectangles, and a two-ary relation $r \in \mathcal{R}$ introduces two rectangles $\mathbf{r}^{(1)}$ and $\mathbf{r}^{(2)} \in \mathbb{R}^{2d}$. The lower and upper bounds of a relation box are denoted as $\mathbf{l}^{(i)}$ and $\mathbf{u}^{(i)}$ and the box center is defined as $\boldsymbol{c}^{(i)} = 0.5 \left(\boldsymbol{l}^{(i)} + \boldsymbol{u}^{(i)} \right)$. The width of the box is $\boldsymbol{w}^{(i)} = \boldsymbol{u}^{(i)} - \boldsymbol{l}^{(i)} + 1$ and is increased by one in all dimensions. The motivation for BoxE is that entity representations must occur in the box of a relation for a

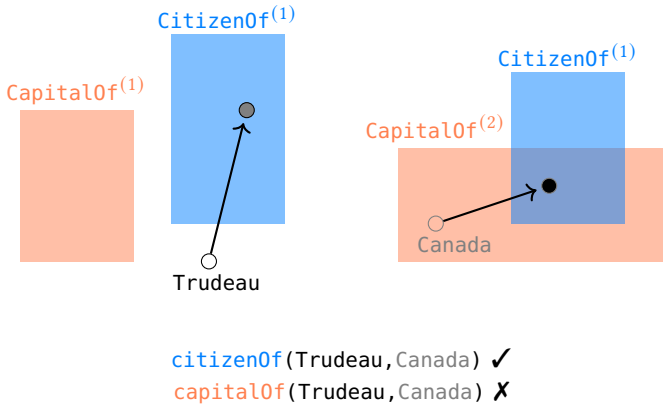


Figure 3.2.: Example of embeddings in BoxE. The entities `Trudeau` and `Canada` translate each other when they co-occur in a fact. The relations `capitalOf` and `citizenOf` have two boxes because they have arity two. The fact `CitizenOf(Trudeau, Canada)` is considered true in the embedding model because both points occur in the boxes of the relation `CitizenOf`. `CapitalOf(Trudeau,Canada)` is considered false because they do not occur both in the box of `CapitalOf`. The figure is inspired from [3].

fact to be true. Therefore, the score function is based on the distance between the entity embedding $\boldsymbol{e}_i^{(h,r,t)}$ and the target relation box $\boldsymbol{r}^{(i)}$:

$$\text{dist}\left(\boldsymbol{e}_i^{(h,r,t)}, \boldsymbol{r}^{(i)}\right) = \begin{cases} \left|\boldsymbol{e}_i^{(e_h,r,e_t)} - \boldsymbol{c}^{(i)}\right| \oslash \boldsymbol{w}^{(i)} & \text{if } \boldsymbol{e}_i \in \boldsymbol{r}^{(i)} \\ \left|\boldsymbol{e}_i^{(e_h,r,e_t)} - \boldsymbol{c}^{(i)}\right| \odot \boldsymbol{w}^{(i)} - \kappa & \text{otherwise.} \end{cases} \tag{3.6}$$

Element-wise multiplication and division are denoted as $\odot$ and $\oslash$. κ is a width-dependent factor. Finally, the score function over all n entities and relation boxes is

$$f_{score}\left(h, r, t\right) = \sum_{i=1}^{n} \left\| \text{dist}\left(\boldsymbol{e}_{\boldsymbol{i}}^{\boldsymbol{r}(e_1,\ldots,e_n)}, \boldsymbol{r}^{(\boldsymbol{i})}\right) \right\|_p . \tag{3.7}$$

Unlike the previous translational models, BoxE can be applied to graphs with n-ary relations [3].

3.1.1.2. Semantic Matching Models

Semantic matching models use tensor decomposition methods to construct a plausibility score for a fact given the entity vectors and a relation matrix.

DistMult [214] represents entities as vectors $\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$ and a relation as diagonal matrix $\mathbf{W}_r \in \mathbb{R}^{d \times d}$. The score of a fact is computed as

$$f_{score}(h, r, t) = \mathbf{h}^T \mathbf{W}_r \mathbf{t} = \sum_{i=1}^{d} \mathbf{h}_i \cdot \text{diag}\left(\mathbf{W}_r\right)_i \cdot \mathbf{t}_i. \tag{3.8}$$

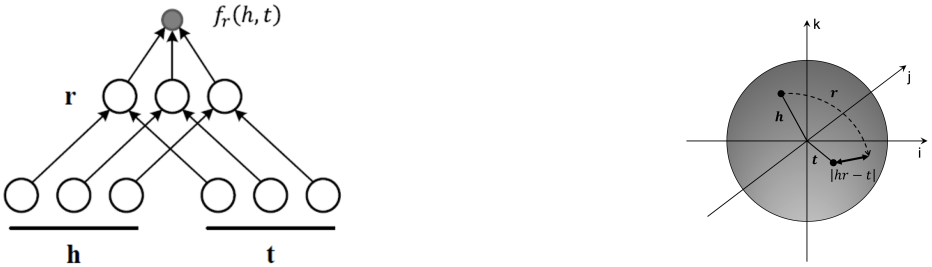


Figure 3.3.: Illustration of DistMult on the left and QuatE on the right. The figure is taken from [30]

The score function of DistMult is shown in Figure 3.3. Antisymmetric relations cannot be modelled, since $f_{\text{score}}(h,r,t) = f_{\text{score}}(t,r,h)$ which forces all relations to be symmetric [79].

ComplEx [194] extends DistMult in the complex space to overcome this limitation. Entities and relations are modelled as vectors in the complex space $\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{C}^k$ and the score function is defined based on the Hadamard product and the real component $Re(\cdot)$ of the complex-valued output vector.

$$f_{\text{score}}(h,r,t) = \text{Re}(\mathbf{h} \odot \mathbf{r} \odot \mathbf{t}) \tag{3.9}$$

Thanks to the commutativity of the Hadamard product in the complex space, ComplEx can model antisymmetric relations.

QuatE [227] uses hypercomplex quaternions, where each value has one real and three imaginary components: $Q = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$. In this notation a, b, c, d are real numbers and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are imaginary parts. Quaternions allow more expressive rotations in two planes to model entities and relations, see Figure 3.3. Given a fact (h, r, t), the representation of the head entity h and the tail entity t are $\mathbf{h} = \left\{a_h + b_h\mathbf{i} + c_h\mathbf{j} + d_h\mathbf{k} : a_h, b_h, c_h, d_h \in \mathbb{R}^k\right\}$ and $\mathbf{t} = \left\{a_t + b_t\mathbf{i} + c_t\mathbf{j} + d_t\mathbf{k} : a_t, b_t, c_t, d_t \in \mathbb{R}^k\right\}$. The relation r is represented by $\mathbf{r} = \left\{a_r + b_r\mathbf{i} + c_r\mathbf{j} + d_r\mathbf{k} : a_r, b_r, c_r, d_r \in \mathbb{R}^k\right\}$. The score function computes the Hamilton product $\otimes$ between the rotated head with a normalized relation vector $\mathbf{r}^\triangleleft$ and the tail entity:

$$f_{score}(h,r,t) = \mathbf{h} \otimes \mathbf{r}^\triangleleft \cdot \mathbf{t}. \tag{3.10}$$

Leveraging representations in the hypercomplex space, QuatE enables richer and more expressive semantic matching between head and tail entities through the Hamilton product. QuatE can be seen as a generalization of DistMult and ComplEx.

3.1.2. Loss Function

The training goal in knowledge graph embeddings is to find a representation of the graph that maximises the plausibility of positive facts while minimising the plausibility of negative facts. Several loss functions $\mathbf{L}$ are considered.

The *Margin-based ranking loss* [26, 204] encourages the discrimination between the scores of positive and negative facts. It is defined as

$$\mathbf{L} = \sum_{(h,r,t)\in\mathcal{F}} \sum_{(h',r',t')\in\mathcal{N}} \max\left(0, f_{score}(\mathbf{h},\mathbf{r},\mathbf{t}) + \gamma - f_{score}\left(\mathbf{h}',\mathbf{r}',\mathbf{t}'\right)\right), \tag{3.11}$$

where $\mathcal{F}$ is the set of positive facts, $\mathcal{N}$ is the set of negative facts and $\gamma > 0$ is a hyperparameter defining the margin between positive and negative facts.

The *Binary cross entropy loss* [55] takes into account the labels $l : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightarrow \{0,1\}^n$ for positive and negative facts and measures the distance between them and the predicted scores for the facts $f_{score} : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \rightarrow \mathbb{R}^n$. The binary cross entropy loss over n facts is defined as

$$\mathbf{L} = -\frac{1}{n}\sum_i \left(l_i \cdot \log\left(f_{score}(\mathbf{h},\mathbf{r},\mathbf{t})_i\right) + (1-l_i)\cdot\log\left(1 - f_{score}(\mathbf{h},\mathbf{r},\mathbf{t})_i\right)\right). \tag{3.12}$$

Other popular loss functions used in the context of knowledge graph embeddings are the mean squared error loss or the adversarial sampling loss [188, 151].

3.1.3. Negative Sampling

As mentioned above, knowledge graphs usually contain only positive facts. However, negative facts are essential during training to avoid overgeneralization to the positive facts. For this reason, negative facts are commonly generated through sampling under the *stochastic local closed world assumption* [26]. Assuming that the unobserved facts are false, a set of negative facts $\mathcal{N}$ is created by randomly replacing the head or tail entity of a positive fact with another entity in the graph [26]. This way, a set of negative facts $\mathcal{N}$ is obtained:

$$\begin{aligned} \mathcal{N}_t(h,r) &= \{(h,r,t') \mid t' \in \mathcal{E} \wedge t' \neq t\} \\ \mathcal{N}_h(r,t) &= \{(h',r,t) \mid h' \in \mathcal{E} \wedge h' \neq h\} \\ \mathcal{N} &= \bigcup_{(h,r,t)\in\mathbf{K}} \mathcal{N}_t(h,r) \cup \mathcal{N}_h(r,t). \end{aligned} \tag{3.13}$$

$\mathcal{N}_h$ is the set of negative facts with corrupted heads and $\mathcal{N}_t$ is the set of negative facts with corrupted tails. Together, they form the set of negative facts $\mathcal{N}$. In the following, this method is termed *uniform negative sampling* [9].

The generation of negative facts is acknowledged to be a challenging problem [121, 229, 228, 215]. The random replacement of head and tail entities carries the risk of generating false negative facts. However, since the number of true positive facts in a graph is generally orders of magnitude smaller than the potential set of true negative facts ($|\mathcal{N}| \gg |\mathcal{F}|$), the probability is small [9]. Some works introduce techniques that are based on the structure of the graph to decrease the probability of generating false negative facts [7, 229]. For example, Bernoulli negative sampling [204] is introduced, where the characteristic of the relation (one-to-many, many-to-one) is decisive for replacing the head or the tail.

3.1.4. Evaluation

Knowledge graph embedding methods are evaluated with ranking-based metrics in the context of a link prediction task [171, 9]. They quantify how successfully the model can complete an incomplete fact by scoring candidate entities. A distinction is made between *head prediction* $(?, r, t)$ and *tail prediction* $(h, r, ?)$, where missing heads or tails are predicted. Some works also consider *relation prediction* $(h, ?, t)$ [125]. Since the scores are only meaningful in a comparative sense, *rank-based metrics* are used. For this purpose, the set of negative facts $\mathcal{N}$ is created by uniform negative sampling based on the facts in the test set, see Section 3.1.3. Then, the score function of the knowledge graph embedding model is used to score all facts, including the true test facts. The *rank* of a fact is its position in the sorted list of scores. The positive test facts should preferably receive a higher rank than the negative test facts.

Rank-based metrics aggregate the ranks of a set of positive facts $\mathcal{F}$ in the test set into one metric. They measure the ability of a model to discriminate positive and negative facts. The following rank-based metrics are commonly used for knowledge graph embedding evaluation in the literature.

- The *Mean Rank (MR)* is the average rank of the positive fact $f \in \mathcal{F}$ against the negative facts:

$$\mathrm{MR}(\mathcal{F}) = \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} \mathrm{rank}(f). \tag{3.14}$$

  The smaller its value, the better the model performance.

- The *Mean Reciprocal Rank (MRR)* is defined as the mean of the reciprocal ranks:

$$\mathrm{MRR}(\mathcal{F}) = \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} \frac{1}{\mathrm{rank}(f)}. \tag{3.15}$$

  Unlike MR, MRR has a fixed range of values from 0 to 1, which makes it easier to interpret. Higher values indicate better performance.

- The *Hits@k* denotes the proportion of positive facts whose rank does not exceed a constant value k with $k > 0$ among all positive facts:

$$\mathrm{Hits@k}\ (\mathcal{F}) = \frac{|\{f \in \mathcal{F} \mid \mathrm{rank}(f) \leq k\}|}{|\mathcal{F}|}. \tag{3.16}$$

  The smaller k, the stricter is the metric. The Hits@k also lies between 0 and 1 where larger values indicate better performance.

3.1.5. Model Expressiveness and Inductive Capacity

In principle, knowledge graph embedding methods are seen as dimensionality reduction methods. The fewer dimensions and less expressiveness a model has, the more regularities are captured in the representations. However, this carries the risk of unintended inferences [79]. It is therefore desirable that a model with enough parameters can theoretically capture the entire training graph and accurately distinguish positive and negative facts. A model with this property is called *fully expressive* [3, 108].

Definition 3.1.1 (Full expressiveness). *Given the set $\mathcal{W}$ of all possible facts over a finite set of relations $\mathcal{R}$, a knowledge graph embedding model $\mathcal{M}$ is fully expressive if, for any two disjoint sets $\mathcal{F} \subseteq \mathcal{W}$ of positive facts and $\mathcal{N} \subseteq \mathcal{W}$ of negative facts, there exists a finite-dimensional model configuration for $\mathcal{M}$ that maps all facts in $\mathcal{W}$ to* True *and all facts in $\mathcal{F}$ to* False.

While the expressiveness of a model determines whether the training graph can be fully captured with a sufficient number of parameters, the *inductive capacity* refers to the ability of a model to generalize beyond the training set. The inductive capacity of knowledge graph embedding methods is studied by theoretically analyzing whether common inference patterns can be captured. An *inference pattern* is a specification of a logical property that can exist in a knowledge graph, which, once learned, allows further inferences from existing facts. The following inference patterns are commonly studied in the literature [3, 108, 30].

Definition 3.1.2 (Symmetry). *Given a relation $r \in \mathcal{R}$, a symmetry pattern is a rule of the form $\forall \mathrm{x}, \mathrm{y} : r(x, y) \rightarrow r(y, x)$, where $r \in \mathcal{R}$.*

Definition 3.1.3 (Antisymmetry). *Given a relation $r \in \mathcal{R}$, an antisymmetry pattern is a rule of the form $\forall \mathrm{x}, \mathrm{y} : r(x, y) \rightarrow \neg r(y, x)$, where $r \in \mathcal{R}$.*

Definition 3.1.4 (Inversion). *Given the relations $r_1, r_2 \in \mathcal{R}$, an inversion pattern is a rule of the form r_2 if $\forall \mathrm{x}, \mathrm{y} : \mathrm{r}_2(\mathrm{x}, \mathrm{y}) \rightarrow \mathrm{r}_1(\mathrm{y}, \mathrm{x})$, where $r_1 \neq r_2 \in \mathcal{R}$.*

Definition 3.1.5 (Composition). *Given the relations $r_1, r_2, r_3 \in \mathcal{R}$, a composition pattern is a rule of the form $\forall \mathrm{x}, \mathrm{y}, \mathrm{z} : \mathrm{r}_2(\mathrm{x}, \mathrm{y}) \wedge \mathrm{r}_3(\mathrm{y}, \mathrm{z}) \rightarrow \mathrm{r}_1(\mathrm{x},$ where $\mathrm{z})$, $r_1 \neq r_2 \neq r_3 \in \mathcal{R}$.*

Definition 3.1.6 (Mutual exclusion). *Given the relations $r_1, r_2 \in \mathcal{R}$, a mutual exclusion pattern is a rule of the form $\forall \mathrm{x}, \mathrm{y}: \mathrm{r}_1(\mathrm{x}, \mathrm{y}) \wedge \mathrm{r}_2(\mathrm{x}, \mathrm{y}) \rightarrow \bot$, where $r_1 \neq r_2 \in \mathcal{R}$.*

Definition 3.1.7 (Hierarchy). *Given the relations $r_1, r_2 \in \mathcal{R}$, a hierarchy pattern is a rule of the form $\forall \mathrm{x}, \mathrm{y}: \mathrm{r}_2(\mathrm{x}, \mathrm{y}) \wedge \mathrm{r}_2(\mathrm{x}, \mathrm{y}) \rightarrow \mathrm{r}_1(\mathrm{x}, \mathrm{y})$, where $r_1 \neq r_2 \in \mathcal{R}$.*

Pattern	BoxE	ComplEx	DistMult	QuatE	RotatE	TransE	TransH
Symmetry	✓	✓	✓	✓	✓	✗	✗
Antisymmetry	✓	✓	✗	✓	✓	✓	✓
Inversion	✓	✓	✗	✓	✓	✓	✓
Composition	✗	✗	✗	✗	✓	✓	✓
Hierarchy	✓	✓	✓	✓	✗	✗	✗
Mutual exclusion	✓	✓	✓	✓	✓	✓	✓
Full expressiveness	✓	✓	✗	✓	✗	✗	✗

Table 3.1.: The inference patterns captured by the presented knowledge graph embedding methods. The results are taken from the literature [3, 79, 188, 30].

	Space Complexity	Time Complexity
TransE	$O(\|\mathcal{E}\|d + \|\mathcal{R}\|d)$	$O(d)$
TransH	$O(\|\mathcal{E}\|d + \|\mathcal{R}\|d)$	$O(d)$
RotatE	$O(\|\mathcal{E}\|d + \|\mathcal{R}\|d)$	$O(d)$
BoxE	$O(\|\mathcal{E}\| + 2\|\mathcal{R}\|d)$	$O(2d)$
DistMult	$O(\|\mathcal{E}\|d + \|\mathcal{R}\|d)$	$O(d)$
ComplEx	$O(\|\mathcal{E}\|d + \|\mathcal{R}\|d)$	$O(d)$
QuatE	$O(\|\mathcal{E}\|d + \|\mathcal{R}\|d)$	$O(d)$

Table 3.2.: Comparison of the space and time complexity of knowledge graph embedding methods. $|\mathcal{E}|$ is the number of entities in the graph and $|\mathcal{R}|$ is the number of relations. d is the dimension of the vector representation. The results are taken from the literature [9].

3.1.6. Comparison of Knowledge Graph Embedding Methods

This section gives an overview of how the presented knowledge graph embedding methods compare with each other in the aspects of expressiveness, complexity and inductive capacity. The results are taken from the literature [3, 79, 188, 30] and are summarized in Table 3.1.

Expressiveness. While DistMult, RotatE, TransE, TransH and DistMult are known to be not fully expressive, BoxE was shown to be fully expressive [3]. The semantic matching models ComplEx and QuatE are fully expressive, but generally less interpretable than the translational models [3].

Complexity. The above-mentioned approaches also differ in terms of the number of parameters and their time complexity, see Table 3.2. All the presented embedding methods have a constant time complexity. Regarding space complexity, BoxE additionally stores the translational bump vectors.

Inductive Capacity. Further, the methods differ in their inductive capacity. Regarding *symmetry*, TransE and TransH cannot represent symmetric relations, since they would force the relation vector of a symmetric relation to be zero. On the contrary, RotatE can represent symmetry if the rotation consists only of multiples of π. In DistMult, relations

are inherently symmetric, since $f(h,r,t) = f(t,r,h)$ holds for any relation. BoxE can capture symmetric relations by defining equal boxes. QuatE can represent symmetric relations by setting the imaginary components to zero.

Antisymmetry can be captured by the translational models TransE, TransH, BoxE and RotatE. Since all relations in DistMult are symmetric, it cannot handle antisymmetry. ComplEx and QuatE, however, can canpture antisymmetry by using only imaginary embeddings.

The translational models TransE, RotatE and TransH can capture *inversion* when $r_1 = -r2$. However, limitations occur when it comes to multiple inverse relations, such as $r_1(x,y) \leftrightarrow r_2(y,x), r_2(x,y) \leftrightarrow r_3(y,x), r_3(x,y) \leftrightarrow r_1(y,x)$. This implies the symmetry of $r1$: $r_1(x,y) \leftrightarrow r_1(y,x)$, which cannot be captured by TransE and TransH, while it can be captured by RotatE. BoxE captures inversion by setting $r_1^{(1)}$ and $r_2^{(2)}$ as well as $r_1^{(2)}$ and $r_2^{(1)}$ to identical boxes. DistMult cannot capture inversion, while ComplEx and QuatE can [3, 227, 194].

The translational models TransE, TransH and RotatE cannot capture *hierarchy* as this would implicitly enforce relational equivalence. The semantic matching models can capture hierarchy, but not in a generalized way. To simultaneously satisfy the rules $r_1(x,y) \rightarrow r_3(x,y)$ and $r_2(x,y) \rightarrow r_3(x,y)$, it must hold that $r_1(x,y) \rightarrow r_2(x,y)$ or $r_2(x,y) \rightarrow r_1(x,y)$ [79]. BoxE can capture hierarchies when the box of r_1 encapsulates the box of r_2.

Composition can be captured by translational models. In TransE and TransH, a relation r_3 is composed of r_1 and r_2 if $r_1 + r_2 = r_3$. In RotatE, this is the case for $r_1 \odot r_2 = r_3$. However, generalized compositions with multiple relations cannot be captured as $r_1(x,y) \wedge r_2(y,z) \rightarrow r_3(x,z)$ and $r_1(x,y) \wedge r_4(y,z) \rightarrow r_3(x,z)$ force $r_2 = r_4$ [3]. The design of the translational bumps prevents BoxE from capturing compositions [3]. It is shown that ComplEx and DistMult cannot capture composition patterns [79].

The aforementioned translational models TransE, TransH, RotatE and BoxE can all capture *mutual exclusion.* ComplEx and DistMult cannot capture mutual exclusion for several relations as $r_1(x,y) \wedge r_2(x,y) \rightarrow \perp$ and $r_1(x,y) \wedge r_3(x,y) \rightarrow \perp$ enforces $r_2 = r_3$.

3.1.7. Limitations

While knowledge graph embedding methods have shown various capabilities in capturing relationships between entities in knowledge graphs, they also suffer from a number of limitations.

Transductive learning. First, most knowledge graph embedding approaches are inherently transductive. They can only learn embeddings for entities and relations observed in the training data. Inference about unseen entities poses a significant challenge. One way to enable inductive inferences about unseen nodes is to use features such as text and images, in node-attributed graphs [2, 225, 193]. However, the majority of knowledge graph embedding approaches are not designed to process such information.

Robustness to noise. As mentioned in Section 1.1.2, knowledge graphs may be subject to noise. In contrast, many state-of-the-art knowledge graph embedding methods do not explicitly take noise in the training graph into account. This assumption is often unrealistic, potentially leading to bias in the learned embeddings and performance degradation [23].

Interpretability. The interpretability of knowledge graphs may be compromised in the vector space. The learned embeddings do not directly translate into human-understandable representations. Interpreting the exact meaning of individual dimensions or components in the embedding space can be challenging. In general, interpretability decreases as the complexity of the embedding space increases.

Incompleteness. Knowledge graphs are typically trained under the local closed-world assumption where negative facts are generated with uniform negative sampling, see Section 3.1.3. However, given that many real-world knowledge graphs are highly incomplete, this procedure has the potential to introduce false negative facts. This can lead to bias in the learned embeddings and incorrect inferences [23]. Furthermore, the commonly used rank-based metrics are inappropriate under the open world assumption and become misleading and inconsistent in the context of incomplete data [216]. In addition, the ability to learn patterns in a knowledge graph depends on the prevalence of the pattern in the training graph. Incomplete information can therefore lead to the failure to capture patterns at inference [171, 78].

Capturing semantics. Although the ability of knowledge graph embeddings to capture common inference patterns is widely discussed in the state-of-the-art [3, 188, 30], capturing compositionality and especially transitivity as general patterns remains a challenging problem [23]. Furthermore, complex constraints are often neglected [36]. For this reason, knowledge graph embeddings often fail to be compliant with prior knowledge.

Neglecting prior knowledge. Further, many knowledge graphs follow a given schema or ontology that formulates prior knowledge about the relations and entities in the graph, as mentioned in Section 1.1.2. Common knowledge graph embedding approaches focus only on facts and ignore ontological information [36, 79, 77, 102]. Consequently, there is no guarantee that the learned embeddings obey the rules of prior knowledge, as this depends on the facts observed during training as well as on the inductive capacity of the chosen model. Thus, knowledge graph embeddings risk losing semantics in the embedding space and do often not provide predictable inference [83].

3.2. Graph Neural Networks

While knowledge graph embeddings find unique embeddings for entities and relations that capture the structure of a knowledge graph, the idea of *Graph Neural Networks (GNNs)* [180] is to refine node representations *guided* by the graph topology. In essence, GNNs transform node vector representations by iteratively updating them with permutation

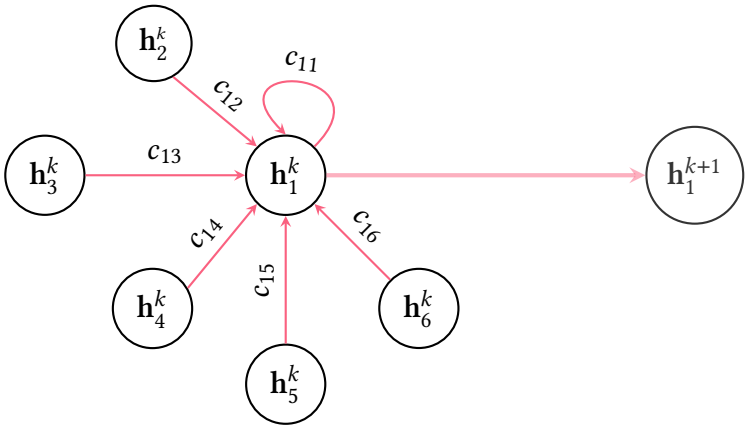


Figure 3.4.: Visualisation of the update of the representation $\mathbf{h}_1$ of the target node v_1 in the k-th GCN layer. The representations of the first-order neighbours $\mathbf{h}_2^k, \ldots, \mathbf{h}_6^k$ are weighted with normalization factors $c_{u,v}$ and aggregated to derive the updated representation $\mathbf{h}_1^{k+1}$ for v_1.

invariant *message passing layers*. GNNs typically expect as input a node-attributed graph. A node v to be represented is called *target node*. In detail, the target node receives messages from its adjacent nodes that are used to update the target node's vector representation $\mathbf{x}^k$. In vector notation, the function of the k-th message passing layer is formalized as

$$\mathbf{h}_v^{k+1} = \texttt{combine}\big(\mathbf{h}_v^k, \texttt{aggregate}\big(m_{v,u} | u \in \mathbf{N}_1(v)\big)\big). \tag{3.17}$$

Here, $\mathbf{h}_v^k \in \mathbb{R}^d$ is the d-dimensional vector representation of the target node v of the previous layer. The neighbourhood $\mathbf{N}_1(v)$ is the first-order neighbourhood of v. In the following notations, the index is omitted for readability. The messages $m_{v,u}$ are defined based on the node representations $\mathbf{h}_v, \mathbf{h}_u$ and the edge features $\mathbf{e}_{v,u}$. The operators `combine` and `aggregate` denote functions where `aggregate` is permutation invariant.[2]

An arbitrary number L of GNN layers can be stacked. This way, node representations are refined by incorporating L-hop neighbourhood information which provides a strong inductive bias on the graph structure. The functions `aggregate` and `combine` contain learnable parameters that are optimized in an end-to-end supervised manner in the context of downstream tasks such as node classification. GNN methods differ in the way they aggregate information across their layers. In the following, *graph convolutional networks*, *graph attention networks* and *relational graph neural networks* are introduced.

3.2.1. Graph Convolutional Networks

Graph Convolutional Networks (GCNs) [115] are known for generalizing the convolution operation in convolutional neural networks [122] to graphs. They update node representations by considering the weighted sum of the local neighbourhood. The parameters are

[2]The notation $\mathbf{h}_v$ is not to be confused with the notation h for the head entity of a fact in Section 3.1. $\mathbf{h}_v$ in this section describes a vector representation for any node in a graph

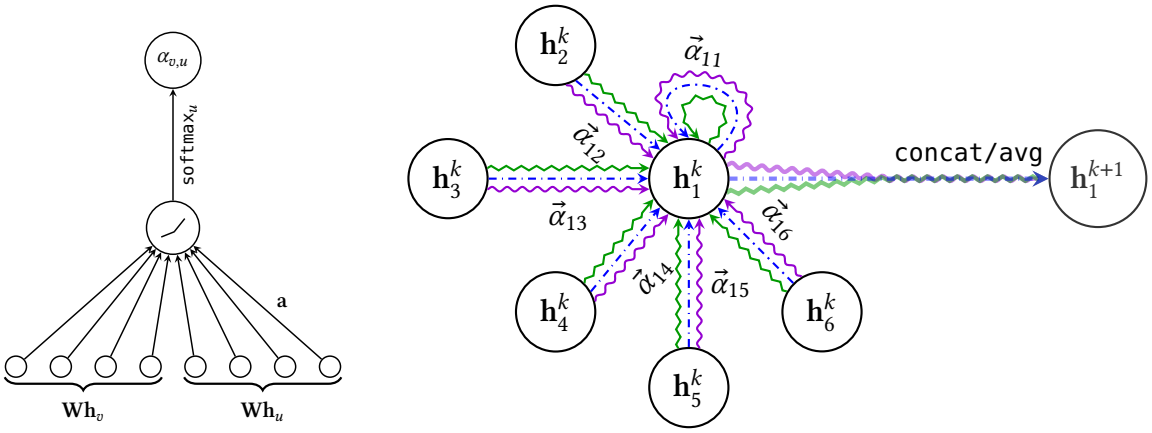


Figure 3.5.: Visualisation of the update of the representation $\mathbf{h}_1$ of target node v_1 in the k-th GAT layer. *Left:* Illustration of the attention mechanism $\mathbf{a}(\mathbf{Wh}_v, \mathbf{Wh}_u)$ in GAT, parametrized by a weight vector $\mathbf{a} \in \mathbb{R}^{2d}$. *Right:* An illustration of multi-head attention with $K = 3$ by node v_1 on its neighbourhood. Different arrow styles and colors describe different attention heads. The aggregated features from each head are concatenated or averaged to obtain $\mathbf{h}_1^{k+1}$. The figure is adapted from [196].

shared across all nodes in the graphs. The function of the k-th GCN layer for target node v is formalized as

$$\mathbf{h}_v^{k+1} = \sigma\left(\sum_{u \in \mathbf{N}(v)} \frac{1}{c_{v,u}} \mathbf{W}^k \mathbf{h}_u^k\right) \tag{3.18}$$

with $c_{u,v} = \sqrt{|\mathbf{N}(u)| \cdot |\mathbf{N}(v)|}$ and activation function $\sigma(\cdot)$. The update of the node representation $\mathbf{h}_v^k$ to $\mathbf{h}_v^{k+1}$ is visualised in Figure 3.4. In matrix notation, the function of a GCN layer is based on the normalised adjacency matrix $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$ of the input graph:

$$\mathbf{H}^{k+1} = \sigma\left(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^k\mathbf{W}^k\right). \tag{3.19}$$

Here, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ is the adjacency matrix with self-connections in the form of the identity matrix $\mathbf{I}$. Self-connections take the target node's representation of the previous layer into account. The diagonal matrix $\tilde{\mathbf{D}}$ where $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$ is used to normalize the updates of a node. The matrices $\tilde{\mathbf{D}}, \mathbf{I}, \tilde{\mathbf{A}}, \mathbf{A}$ have the dimension $\mathbb{R}^{n \times n}$. Further, $\mathbf{W}^k \in \mathbb{R}^{d_k \times d_{k+1}}$ represent the trainable weight matrices. The input and output dimensions of the layer are d_k and d_{k+1}. For GCN, the `aggregate` function is the weighted average of the neighbouring node representations. The `combine` function is the sum of the aggregated messages with the node representation itself, normalized by the node degree.

3.2.2. Graph Attention Networks

While in GCNs the impact of neighbouring nodes is normalized by the node degree, *Graph Attention Networks (GATs)* [196] employ the *attention mechanism* [195] to learn the

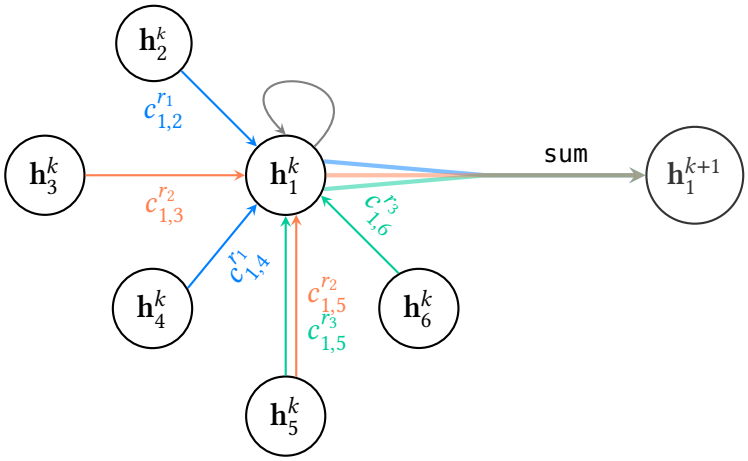


Figure 3.6.: Visualisation of the update of the representation $\mathbf{h}_1$ of target node v_1 in the k-th RGCN layer. The graph has three different types of relations: r_1, r_2 and r_3. The relation-specific neighbourhoods are $\mathbf{N}^{r_1}(v) = \{v_2, v_4\}$, $\mathbf{N}^{r_2}(v) = \{v_3, v_5\}$ and $\mathbf{N}^{r_3}(v) = \{v_5, v_6\}$ with the respective normalization factors $c_{v,u}^{r_1}$, $c_{v,u}^{r_2}$ and $c_{v,u}^{r_3}$. The weight matrices $\mathbf{W}^{r_1}$, $\mathbf{W}^{r_2}$ and $\mathbf{W}^{r_3}$ are relation-specific.

importance of a neighbouring node for the target node. To this end, a shared attention mechanism $\mathbf{a} : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}$ computes *attention coefficients*

$$e_{v,u} = \mathbf{a}(\mathbf{W}\mathbf{h}_v, \mathbf{W}\mathbf{h}_v) \tag{3.20}$$

that determine the importance of the feature vector of node v for node u. The attention coefficients are calculated for the first-order neighbours and normalized with the softmax function:

$$\alpha_{v,u} = \text{softmax}_u\left(e_{v,u}\right) = \frac{\exp\left(e_{v,u}\right)}{\sum_{j \in \mathbf{N}(v)} \exp\left(e_{v,j}\right)}. \tag{3.21}$$

The function of the k-th GAT layer is

$$\mathbf{h}_v^{k+1} = \sigma\left(\sum_{u \in \mathbf{N}(v)} \alpha_{v,u} \mathbf{W}^k \mathbf{h}_u^k\right). \tag{3.22}$$

To stabilize the learning process, *multi-head attention* can be applied. Therefore, the attention mechanism is executed K times independently and the results are averaged or concatenated. The computation of the attention weights and the function of a GAT layer are visualised in Figure 3.5.

3.2.3. Relational Graph Neural Networks

The presented methods GCN and GAT are limited to homogeneous graphs. As extension to this, *Relational Graph Neural Networks (RGCNs)* [181] apply the concept of GCN to multi-relational graphs by introducing relation-specific weight matrices $\mathbf{W}_r^k \in \mathbb{R}^{d_k \times d_{k+1}}$. In

an RGCN layer, the node features are transformed with the normalized sum of adjacent edges:

$$\mathbf{h}_v^{k+1} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{u\in\mathbf{N}^r(v)\setminus v}\frac{1}{c_{v,u}^r}\mathbf{W}_r^k\mathbf{h}_u^k + \mathbf{W}_0^k\mathbf{h}_v^k\right). \tag{3.23}$$

The update function of a RGCN layer is shown in Figure 3.6. The set of neighbour nodes of v under the relation $r \in \mathcal{R}$ is denoted as $\mathbf{N}^r(v)$. $c_{v,u}^r$ is a relation-specific normalization hyperparameter. The function of an RGCN layer is illustrated in Figure 3.6. Since the number of trainable parameters increases rapidly with the number of relations, potentially leading to overfitting, regularization methods are often applied in the context of RGCN [181].

3.2.4. Neural Knowledge Graph Embeddings

Neural network layers and particularly graph neural network layers can also be used to embed knowledge graphs. RGCNs, for example, are commonly used as link prediction methods. While they can be applied in an inductive setting on node-attributed graphs, they can also be used in an transductive context to learn and refine entity vectors. In this case, the entity representations are randomly initialized and optimized together with the weight matrices during training. In this context, the RGCN layers in Equation 3.23 serve as *encoders* to update node representations and DistMult in Section 3.1 is employed as a *decoder* to obtain a link prediction score. In this setting, RGCN is categorized as a neural knowledge graph embedding method, since the layers of the graph neural network update the representations of the entities. However, RGCN as a knowledge graph embedding method is limited to the inductive capacity of the DistMult decoder.

Beyond RGCN, other methods based on neural network layers exist that can be used refine knowledge graph embeddings. Prominent examples are *Neural Tensor Networks* [187], *ConvE* [55] or *ConvKB* [155]. Furthermore, the experiments in [130] show that message passing techniques are not necessarily helpful in the context of knowledge graph embedding learning and link prediction and that equal performance can be achieved with simple feedforward layers.

3.2.5. Limitations

Despite their effectiveness in refining representations of nodes based on graph structure, graph neural networks have some limitations.

Oversmoothing and Oversquashing. With an increasing number of GNN layers, the message passing calculation includes more nodes. In this case, the representations of different nodes may converge to equal vectors, making them less distinguishable. This problem is known as *oversmoothing* [199, 213, 175]. Moreover, the aggregation of too much information into a single vector can lead to a representation that is less informative, which is

known as *oversquashing*. [15]. Therefore, the choice of an appropriate number of GNN layers is important for learning meaningful representations.

Multi-relational and Heterogeneous Graphs. Some GNN methods are limited to certain types of graphs. Many prominent graph neural networks, including GCN and GAT, are designed for homogeneous graphs. However, many graphs in practice are multi-relational, including knowledge graphs. Applying methods for homogeneous graphs to heterogeneous graphs leads to the aggregation of disparate information, which can result in reduced performance [32]. RGCN and other methods [223, 203] encode multiple relations in a graph with separate matrices. However, these models are heavier in terms of parameters.

Interpretability. As a subcategory of deep neural networks, GNNs are used to learn meaningful feature representations with their message passing layers [4]. Although they are effective at capturing graph structure, they are difficult for humans to understand and interpret because of their large number of parameters. Furthermore, no strong guarantees are given that encoding with GNN is faithful and trustworthy at inference.

Scalability. Deep GNN architectures with many message passing layers can increase the computational cost of training and inference. As the number of parameters in the model grows, so does the computational complexity, making it difficult to scale GNNs to large graphs. Further, the message passing itself aggregates the node neighbourhood, which grows exponentially with the number of layers in the network. Several methods tackle this problem with sampling techniques [80, 59, 221] or distributed training [62].

4. Neuro-Symbolic Reasoning

The previous chapters have introduced and exemplified relevant techniques and concepts in the field of symbolic and sub-symbolic AI. Historically, the fields of symbolic and sub-symbolic AI developed separately [100]. While symbolic AI was the dominant paradigm in AI research before 1980, it received less attention after the major breakthroughs in deep learning around the year of 2012 [122].

The two lines of research differ significantly. In sub-symbolic AI, problems are formalised as a quantitative and differentiable objective function that is optimised using gradient descent. This allows for efficient pattern recognition and feature extraction from high-dimensional data without human intervention. For this reason, sub-symbolic approaches are well suited for perception problems where raw sensor data is directly processed, such as image and speech recognition. Sub-symbolic approaches are robust to noisy and imperfect data. Furthermore, once the model parameters are trained, the methods usually scale well at inference.

Symbolic AI does not achieve good marks in these aspects. Many approaches do not scale well at inference, which limits their applicability in the context of real-world use cases. Further, they rely on exact and symbolic presentations of knowledge that often fail to capture noise and uncertainty [88, 141]. Nevertheless, despite the recent hype around machine learning and deep learning, concerns are raised. Some voices call for more trust, security, interpretability and accountability for deep neural networks, which are often criticised as *black-box models* [68, 69, 142, 190]. In addition, sub-symbolic approaches are highly dependent on data [189]. While they can process high-dimensional data, they also *require* a rich and extensive data source. Their usefulness is limited in cases where data is scarce and expensive. Even when expert knowledge is available, sub-symbolic AI does often not provide mechanisms to exploit it. Furthermore, sub-symbolic methods are prone

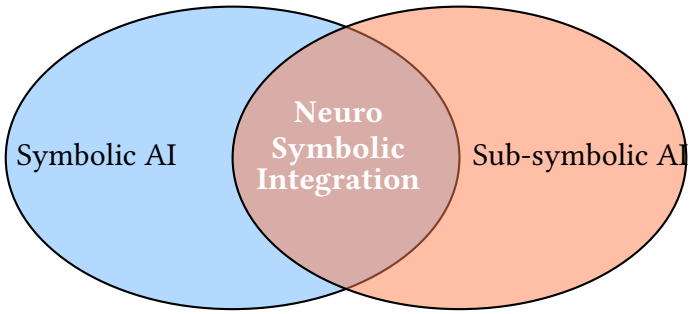


Figure 4.1.: Neuro-Symbolic Integration = Symbolic AI + Sub-symbolic AI

Symbolic	Sub-symbolic
Discrete representations	Continuous vector representation
Language-like representations	Numeric representations
Reasoning	Learning
Rigid and static	Flexible and adaptive
Human intervention	Automatic pattern extraction
Prior knowledge	Inductive bias
Feature engineering	Raw sensor data
Requires no data/small data	Requires big data
Precise input	Noisy or incomplete input
Reasoning problems	Perceptual problems
Interpretable	Black-box
Knowledge transfer	Overfitting
Not scalable at inference	Scalable at inference

Table 4.1.: Overview of the characteristics, strength and weaknesses of symbolic and sub-symbolic AI. Some points are inspired from [100].

to overfitting and often suffer from limited generalization capacity beyond the training data distribution [18].

Symbolic AI methods provide an answer to these limitations, despite their own previously mentioned shortcomings. With their symbolic representations in formal logic, they offer a high degree of comprehensibility and provable correctness that goes beyond statistical assessment. The ability to explain and reason about intermediate steps and decision making makes symbolic AI approaches inherently interpretable. Furthermore, the use of symbolic representations allows for leveraging expert knowledge, which is often compositional and can be shared between closely related domains. Symbolic knowledge can also help in situations where less data is available.

Thus, the strengths and weaknesses of both lines of research complement each other. In view of this, *neuro-symbolic AI* [19, 88, 126, 67] explores the combination of both paradigms in a favourable way, with the objective to leverage their mutual strengths and circumvent their respective limitations, as visualised in Figure 4.1. Neuro-Symbolic AI has recently gained increasing attention [107, 17, 90]. With an eye on the future, the field has been described as a "path forward to much stronger AI systems" and even as a "major stepping stone towards human-level artificial intelligence" [179]. The combination of sub-symbolic and symbolic AI is also biologically motivated, as it strongly resembles human intelligence and learning. Humans can learn from experience as well as from explanation. While the former is intuitive and implicit, the latter is explicit and cognitive. These systems are also designated as *System 1 (fast thinking)* and *System 2 (slow thinking)* [105]. The pattern-matching abilities in neural networks learned through repetitive training correspond to System 1, while symbolic reasoning corresponds to System 2.

4.1. Desiderata of Neuro-symbolic AI

Since the aim of neuro-symbolic AI is to combine the advantages of both fields in a favourable way, some desired aspects of neuro-symbolic AI approaches are summarised as follows.

- **knowledge-aware**
  Neuro-symbolic methods should be able to take into account prior knowledge, such as expert and common sense knowledge.

- **robust**
  Neuro-symbolic approaches should be applicable to high-dimensional and raw data that may be imperfect, with noise and incompleteness. Noise in data is understood as random or irrelevant variations that are not part of the pattern to be learned. A model is considered robust if its performance remains stable or is only slightly affected by noise.

- **scalable**
  Neuro-symbolic methods should be scalable. Their applicability should go beyond toy problems and consider real-world scenarios that often involve large amounts of data. Sub-symbolic techniques could benefit from symbolic knowledge to scale up training, while symbolic techniques could benefit from sub-symbolic techniques to speed up inference.

- **interpretable**
  A model is considered as interpretable if humans can understand the cause and reasoning steps of its predictions and decisions [150]. Symbolic techniques should contribute interpretable representations and understandable reasoning processes to neural black-box models.

- **accurate**
  Neuro-symbolic methods should perform as well as or better than comparable purely symbolic or sub-symbolic methods. This also means that neuro-symbolic methods can potentially achieve similar results with the use of fewer resources.

A key challenge in neuro-symbolic integration is the *symbol grounding problem* [84]. While sub-symbolic approaches rely on continuous and differentiable vector representations, symbolic AI operates on discrete and language-like representations. Translating representations from one to another while preserving relevant information is relevant for the design of neuro-symbolic AI approaches.

In the state-of-the-art, several ways are proposed to achieve the integration of symbolic and sub-symbolic AI approaches. In the following sections, several neuro-symbolic methods will be presented. Since the focus in this thesis lies on graph-structured data, Section 4.2 introduces general neuro-symbolic frameworks and discusses their applicability to graphs. Then, Section 4.3 presents methods specifically designed for knowledge graphs.

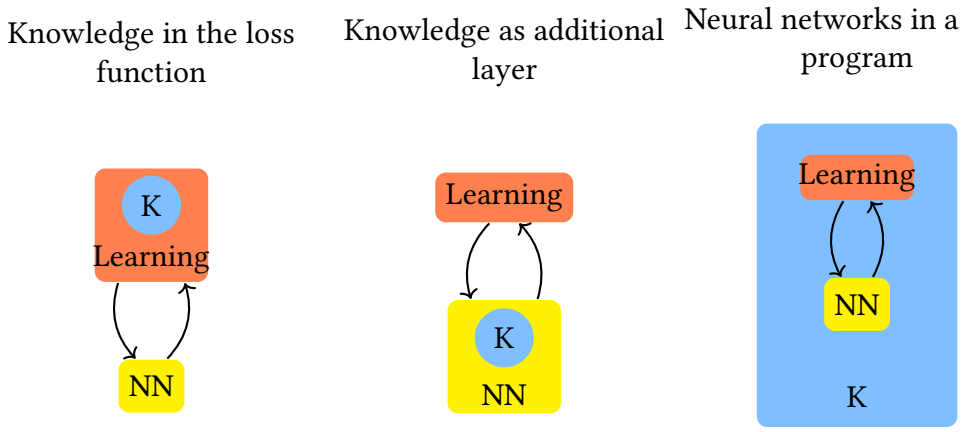


Figure 4.2.: Schematic illustration of ways to integrate knowledge with neural networks for general neuro-symbolic frameworks.

4.2. Prominent Neuro-symbolic Frameworks in the Context of Graph Data

In this section, the neuro-symbolic frameworks *Neural Probabilistic Programming* [139, 95], *Logic Tensor Networks* [14] and *Knowledge Enhanced Neural Networks* [48] are discussed. The main difference between these approaches is the way in which they address the symbolic grounding problem and integrate their symbolic and sub-symbolic components, as visualizes in Figure 4.2. Neural Probabilistic Programming approaches employ neural predicates in a logic program. Logic Tensor Networks represent a logic program in fuzzy logic and encode it as a loss function of a differentiable learning problem. Knowledge Enhanced Neural Networks encode logical rules as differentiable layers that modify the predictions of a neural network. The following literature can be consulted for a more extensive overview [88, 50, 13, 67].

4.2.1. Neural Probabilistic Programming

DeepProblog [139] is a neural probabilistic programming language based on ProbLog, see Section 2.2. In essence, it extends ProbLog with *neural predicates*. While ProbLog assumes that the fact probabilities $p :: f$ are known, neural predicates act as perception model $\mathcal{M}_\theta$ with learnable parameters θ, which transforms input data into a probabilistic fact. Therefore, the output of a neural network Y given an input X is normalized to $[0, 1] \subset \mathbb{R}$ and interpreted as probability in the ProbLog program. Neural predicates thus act as a bridge between the logic program and the perception of the input data. A dataset in DeepProblog is considered as a set of tuples Q consisting of sensor data $\mathbf{x} \in \mathbb{R}^d$, and labels with desired success probabilities q for the queries. For each data point of the dataset $(\mathbf{x}, q)$, the neural predicates transform $\mathbf{x}$ into a probabilistic fact $p(\mathbf{x}) :: f$ that is captured by a Datalog program.

Example 4.2.1 (DeepProblog). Consider the MNIST addition as an example. Given two images of handwritten digits, the task is to determine the sum of the digits in the images. The dataset contains tuples of two images and the sum of the digits in the images, e.g. ⟨ ( , ), 8 ⟩. Two neural predicates are used to recognise the handwritten digits in the images and predict one of the values in $\{0, \ldots, 9\}$. A neural predicate is formalized as follows:

$$\mathcal{M}_\theta(\texttt{mnist\_net}, \mathbf{x}, [0, ..., 9]) :: \texttt{digit}(X, Y).$$

For example, the output for the image  would be a sequence of ten truth values

$$0.01 :: \texttt{digit}(\text{}, 0), 0.3 :: \texttt{digit}(\text{}, 1), \ldots, 0.1 :: \texttt{digit}(\text{}, 9),$$

that are interpreted as probabilistic facts in the DeepProbLog program. For example, the fact `0.3 :: digit(` `, 1)` describes that the image is recognized as digit `1` with probability 0.3. Further, the concept of addition is introduced in the logic program.

```
1    nn(mnist_net, X, [0, ... , 9]) :: digit(X,Y)
2    addition(X,Y,Z) :- digit(X, "N1"), digit(Y, "N2"), Z = N1 + N2
```

Inference in DeepProblog works essentially as in Problog, based on weighted model counting, but the prediction of the neural predicates are used as fact probabilities, see Section 2.2. However, in order to determine the parameters of the neural predicates, DeepProbLog relies on a training stage. The goal is to learn the parameters θ of the perception model $\mathcal{M}_\theta$. The training objective is the loss $\mathbf{L}$ of the output probability of the program and the desired success probability q of the query, averaged over all samples $(p(\mathbf{x}), q) \subset Q$.

$$\arg\min_{\mathbf{x}} \frac{1}{|Q|} \sum_{(q,p)\in Q} \mathbf{L}\left(P^{\theta}_{X=\mathbf{x}}(p), q\right). \tag{4.1}$$

The final success probability is calculated in the logic program $P^{\theta}_{X=\mathbf{x}}(p)$ In DeepProbLog, only the success probability of the query is known and used as supervision. Since no intermediate labels are given for the neural predicates, the gradients must be derived in the logic component in order to perform backpropagation and update the neural network parameters during training, despite the lack of direct supervision. To define gradient updates in logic, DeepProbLog relies on a *gradient semiring*. It determines how probabilities and their differentiation are handled when reasoning with a logic program. The elements of gradient semiring are tuples $\left(p, \frac{\partial p}{\partial x}\right)$, where p is the probability of a probabilistic fact and $\frac{\partial p}{\partial \theta}$ is the partial derivative of that probability with respect to the parameter θ. Then, the semiring addition $\oplus$, multiplication $\otimes$ and their neutral elements are defined as

$$\begin{aligned}
\left(a_1, \overrightarrow{a_2}\right) \oplus \left(b_1, \overrightarrow{b_2}\right) &= \left(a_1 + b_1, \overrightarrow{a_2} + \overrightarrow{b_2}\right), e^{\oplus} = (0, \overrightarrow{0}) \\
\left(a_1, \overrightarrow{a_2}\right) \otimes \left(b_1, \overrightarrow{b_2}\right) &= \left(a_1 b_1, b_1\overrightarrow{a_2} + a_1\overrightarrow{b_2}\right), e^{\otimes} = (1, \overrightarrow{0}).
\end{aligned} \tag{4.2}$$

The second part of the tuple shows the gradients based on derivative rules. By modelling the bottom-up evaluation of a query in the SDD in a differentiable way, gradients are

backpropagated to the parameters of the neural predicates, even though their outcome is only indirectly supervised in the form of the final outcome of the query.

Experiments on several use cases show that DeepProblog converges faster than pure neural networks and is more robust to noise [139]. However, the size of the proof set to be considered for evaluating a query $|S_q|$ increases exponentially with the number of input facts. This poses a scalability problem, relevant for both inference and training, and limits DeepProblog to small use cases [95, 178].

Extending DeepProbLog, *Scallop* [95] builds on DeepProbLog and sets out to make inference and learning more scalable. Essentially, it proposes a *top-k semiring* to infer the success probability of a given query in a more efficient way. The idea of the top-k proof semiring is to approximate the calculation of the success probability by including only the k most likely proofs, where $k \geq 1$ is a hyperparameter. Therefore, the operators $\otimes^k$ for conjunction and $\oplus^k$ for disjunction are redefined:

$$S_1 \otimes^{(k)} S_2 = \mathtt{Top}_k \left(S_1 \otimes S_2\right), \quad S_1 \oplus^{(k)} S_2 = \mathtt{Top}_k \left(S_1 \oplus S_2\right). \tag{4.3}$$

Prior to the probability calculation, the proofs are ranked by their likelihood and only the top k ranked proofs are considered. The resulting set of proofs is determined as

$$\tilde{S}_q = \bigoplus_{F \text{ derives } q}^{(k)} \left(\bigotimes_{f \in F}^{(k)} S_f \right). \tag{4.4}$$

This way, the calculation of the success probability of a query q is approximated by considering only the set of the k most likely proofs $\tilde{S}_q$ instead of all proofs S_k: $Pr(q) = Pr(S_q) \approx \Pr(\tilde{S}_q)$. This modification results in a constant complexity of $\left|\tilde{S}_q\right| = O(k)$. Experiments show that Scallop scales significantly better than DeepProblog without sacrificing accuracy [95].

4.2.1.1. Application to Graph Data

In [95], DeepProblog and Scallop are applied to the kinship reasoning task from a natural language context on the ClutRR dataset [184]. Given a natural language fragment about a set of characters and kinship relations between them, the task is to reason about an implicit relation provided as query. The query relation itself is not contained in the text, but can be inferred from the relations described in the text using a logic program. In the task, only a label is given for the implicit relationship in the query. This, together with a logic program about kinship relations, can be used to learn the parameters of a language model for extracting the facts from the text. In the experiments in [95], the parameters of a RobertA [134] model for text embedding learning and an MLP as relation extractor are optimised.

Example 4.2.2 (Kinship Reasoning with Scallop). Consider an instance of the Clutrr dataset, which consists of a query and a text extract in Figure 4.3. Here, the text describes

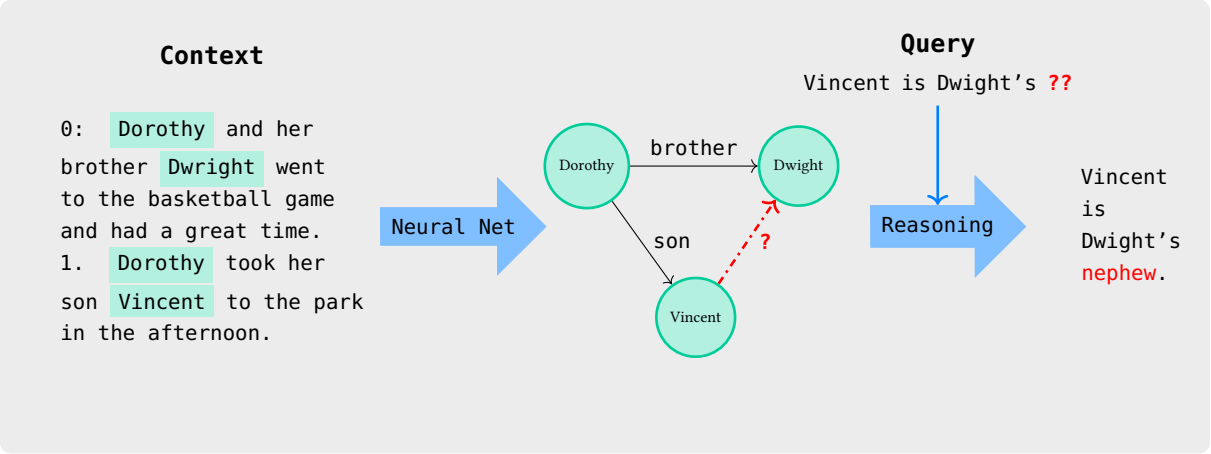


Figure 4.3.: Illustration of the kinship reasoning task with Scallop on the Clutrr dataset.

the three characters Dorothy, Dwright and Vincent. The query to be answered is how Vincent and Dwright are related to each other. Only supervision for the query relation is available during training. First, the RobertA model returns text embeddings. Then, for all pairs of entities that occur in a sentence, the MLP outputs a score for each possible kinship relationship in the domain, e.g. `[0.4, (daughter, dwight, dorothy)]`. This results in $2^2 \cdot 21$ probabilities with 21 relations and 2 characters per sentence. These probabilistic facts are given to the logic program, which in this example contains compositional rules of the family relations, e.g. `(daughter - daughter - granddaughter)`. The probability of the query relation can be inferred with the logic program. The loss between the query result and the prediced probability can be computed and the gradients for the MLP and RobertA parameters are backpropagated.

In the Clutrr example, the output of the neural network contains probabilities for all possible pairs of characters described in the text and all possible relations in the domain. For n unique characters and m relations, $2^n \cdot m$ probabilities must be computed, which has exponential complexity. However, in the Clutrr example, the text snippets contain few different characters. Furthermore, even if the same character name is used in several text samples with different queries, they are not considered to be the same entity and are assumed to be independent. Consequently, the Clutrr example is not representative for the application of Scallop to large graphs and avoids the scalability problem resulting from the combinatorial explosion in the SDD evaluation.

4.2.2. Logic Tensor Networks

Logic Tensor Networks (LTN) [14, 57] is a neuro-symbolic framework that learns neural network parameters by expressing knowledge as constraints in first-order logic and optimising their joint satisfaction.

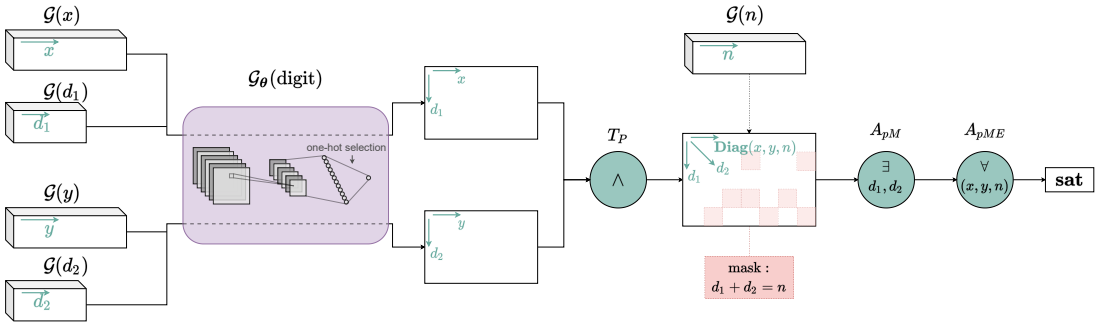


Figure 4.4.: Illustration of the tensor computation graph of the MNIST addition example in LTN. The figure is taken from [14].

LTN is based on *Real Logic* [14]. A theory in real logic is denoted as $\mathcal{T} = \langle \mathcal{K}, \mathcal{I}(\cdot \mid \theta) \rangle$. It consists of first-order logic formulae $\mathcal{K}$ in a logical language $\mathcal{L}$ with finite sets of variables $\mathcal{X}$, constants C, functions F and predicates $\mathcal{P}$. The knowledge applies to a domain $\mathcal{D}$. Further, $\mathcal{I}(\cdot \mid \theta)$ is a parametric *interpretation*[1] of all the symbols and operators in the logical language in the real-valued domain.

Definition 4.2.1 (Interpretation). *An interpretation $\mathcal{I}$ of a first-order logical language $\mathcal{L}$ is a function from the signature of $\mathcal{L}$ to the real numbers that satisfies the following conditions:*

1. $\mathcal{I}(c) \in \mathbb{R}^n$ *for every constant symbol* $c \in C$;
2. $\mathcal{I}(f) \in \mathbb{R}^{n \cdot m} \longrightarrow \mathbb{R}^n$ *for every function* $f \in F$ *with arity* m;
3. $\mathcal{I}(P) \in \mathbb{R}^{n \cdot m} \longrightarrow [0, 1]$ *for every predicate* $P \in \mathcal{P}$ *with arity* m.

Thus, constants are mapped to vectors in $\mathbb{R}^n$, m-ary functions are mapped to m-ary real functions, and m-ary predicates are mapped to fuzzy subsets of $[0, 1] \subset \mathbb{R}$. Complex formulae are built from these components with a set of connectives and quantifiers $\{\wedge, \vee, \neg, \rightarrow, \leftrightarrow, \forall, \exists\}$. They are interpreted with fuzzy logic operators, see Section 1.2.3.

Example 4.2.3 (Logic Tensor Networks). The MNIST addition example from Section 4.2.1 is modelled in LTN as follows [14]. The tensor computations are illustrated in Figure 4.4. The predicate `digit(x,d)` denotes the truth value for a digit in an image, and the digit addition is formulated as

$$\forall (x, y, n) : (\exists d_1, d_2 : d_1 + d_2 = n \,,\, (\text{digit}\,(x, d_1) \wedge \text{digit}\,(y, d_2)))\,,$$

[1] In [14] this mapping is called *grounding* but does not correspond to the term grounding in logic as described in Section 1.2.2. *Interpretation* is used to distinguish the terms.

where x, y are variables for the first and the second image, d_1 and d_2 are the digits in the images and n is the sum of the digits. The language is interpreted in real logic:

$\cdot\ \mathcal{I}(\text{images}) = [0,1]^{28\times28\times1}$, *MNIST images that have* 28×28 *pixels and a greyscale value.*
$\cdot\ \mathcal{I}(\text{results}) = \mathbb{N}$
$\cdot\ \mathcal{I}(\text{digits}) = \{0, 1, \ldots, 9\}$
$\cdot\ \mathcal{I}(x) \in [0,1]^{m\times28\times28\times1}, \mathcal{I}(y) \in [0,1]^{m\times28\times28\times1}, \mathcal{I}(n) \in \mathbb{N}^m$
$\cdot\ \mathcal{I}(d_1) = \mathcal{I}(d_2) = \langle 0, 1, \ldots, 9\rangle$
$\cdot\ \mathcal{I}(\text{ digit } \mid \theta) : x, d \mapsto \texttt{onehot}(d)^\top \cdot \texttt{softmax}(\texttt{CNN}_\theta(x))$

$\texttt{CNN}_\theta$ is a convolutional neural network [122] with 10 output neurons. The operator $\texttt{onehot}(d)$ converts the label d into a one-hot encoded vector.

Given this interpretation $\mathcal{I}$, all components of the theory can be represented in the real-valued domain. A *maximum satisfiability problem* can be formulated with the grounded rules and predicates. Hence, learning in LTN is defined as the process of finding the parameter values θ^* that maximize the satisfiability of the theory $\mathcal{T}$ w.r.t. a given aggregator

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}\in\boldsymbol{\Theta}}{\text{argmax}} \text{SatAgg}_{\phi\in\mathcal{K}} \mathcal{I}_{\boldsymbol{\theta}}(\phi). \tag{4.5}$$

Here, Θ is the parameter search space and $\phi \in \mathcal{K}$ is a formula. In this example, the parameters of the CNN classifying the digits in the images are learned by optimising the satisfiability loss function. Although there is no direct supervision of the digit classification, the satisfaction of the addition constraint is part of the constraint satisfaction optimisation problem. In some cases, a regularisation term is applied to the parameter set to keep the learned parameters small.

At inference, the learned parameters for the predicates are kept constant. Given any input query, they can serve as functions that return truth values. Further, the confidence of complex formulae is evaluated by grounding them to the constants in the domain and using the semantics of real logic.

4.2.2.1. Application to Graph Data: Learning Embeddings with LTN

LTN was proposed as a framework for learning embeddings in graphs [14]. The entities of a domain are initialised with random vectors, which are trainable parameters. The set of formulae $\mathcal{K}$ over a domain is expressed in real logic and encoded as a loss function, see Equation 4.5. By optimising the loss functions, the vector embeddings are refined, as well as the parameters for the predicates. During inference, the learned embedding vectors and functions can be queried. Also, formulae can be queried and truth values returned for them during inference. In [14], embedding learning with LTN is illustrated using the Smoker-Friends-Cancer Example [172].

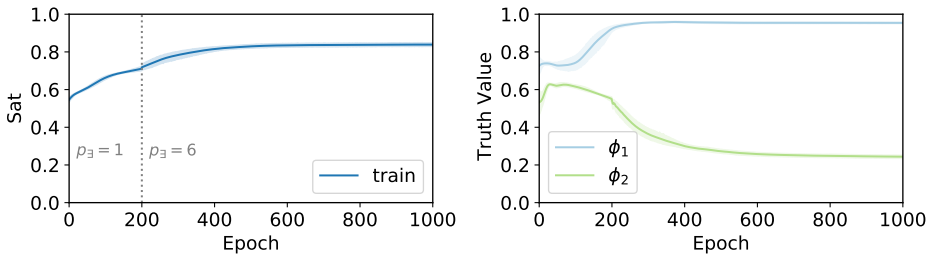


Figure 4.5.: The satisfiability levels during the training of LTN on the left and the truth values of the formulae ϕ_1 and ϕ_2 during training on the right. The figure is taken from [14].

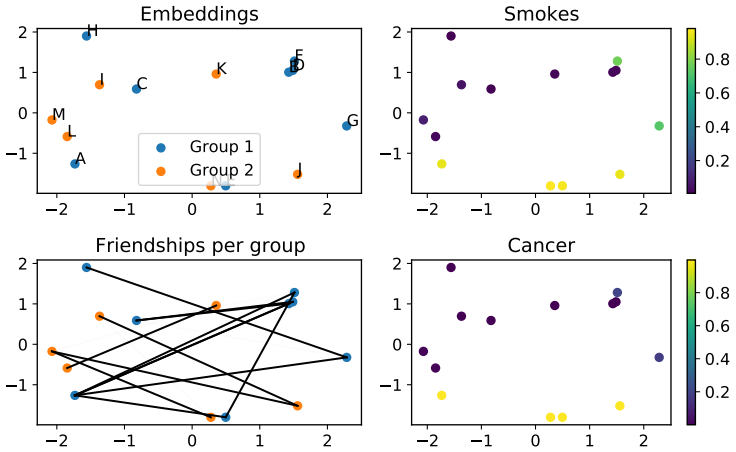


Figure 4.6.: The results of the experiments on embedding learning with LTN. The truth values for the predicates `Smoker` and `Cancer` are plotted on the right. The learned embeddings for the entities in the groups $\mathcal{E}_1$ and $\mathcal{E}_2$ and their friendship relations are plotted on the right. The figure is taken from [14].

Example 4.2.4 (Embedding learning with Logic Tensor Networks). The Smoker-Friends-Cancer Example contains 14 entities that are separated into two groups $\mathcal{E}_1 = \{a, b, \ldots, h\}$ and $\mathcal{E}_2 = \{i, j, \ldots, n\}$. The following entities are smokers: $\mathrm{S} = \{a, e, f, g, j, n\}$. All other entities are non-smokers. Friendship relations between people are also given: $\mathrm{F} = \{$(a, b), (a, e), (a, f), (a, g), (b, c), (c, d), (e, f), (g, h), (i, j), (j, m), (k, l), (m, n)$\}$. They are non-reflexive and symmetric. For the entities in $\mathcal{E}_1$ it is known whether they suffer from cancer or not: $\mathrm{C} = \{a, b\}$. For the entities in $\mathcal{E}_2$ it is unknown whether they have cancer or not. Furthermore, general axioms can be formulated about the domain. The general axioms are: (1) smoking habits are shared among friends, and (2) smoking causes cancer. This results in a set of axioms.

- $F(u, v) \quad \text{for} (u, v) \in \mathrm{F}$
- $\neg F(u, v) \quad \text{for } (u, v) \notin \mathrm{F}, u > v$
- $S(u) \quad \text{for } u \in \mathrm{S}$

- $\neg S(u) \quad \text{for } u \in (\mathcal{E}_1 \cup \mathcal{E}_2) \setminus \mathbf{S}$
- $C(u) \quad \text{for } u \in \mathbf{C}$
- $\neg C(u) \quad \text{for } u \in \mathcal{E}_1 \setminus \mathbf{C}$
- $\forall x \neg F(x, x)$
- $\forall x, y(F(x, y) \rightarrow F(y, x))$
- $\forall x \exists y F(x, y)$
- $\forall x, y((F(x, y) \wedge S(x)) \rightarrow S(y))$
- $\forall x(S(x) \rightarrow C(x))$
- $\forall x(\neg C(x) \rightarrow \neg S(x))$

Note that from a logical point of view, the formulae are not simultaneously satisfiable. There are people who smoke and do not get cancer, or non-smokers in a group of friends who smoke. Therefore, the formulation in fuzzy logic is crucial to express the degree of truth.

In the real-valued space, the formulae are interpreted as follows. The embeddings are intialized with random vectors $\{\mathbf{v}_\theta(a), \ldots, \mathbf{v}_\theta(n)\}$ of dimension $\mathbb{R}^5$. Multilayer perceptrons (`MLP`) represent the predicates S, F, C in real logic.

- $\mathcal{I}$ (people) $= \mathbb{R}^5$. *The model is expected to learn embeddings in* $\mathbb{R}^5$.
- $\mathcal{I}(a \mid \theta) = \mathbf{v}_\theta(a), \ldots, \mathcal{I}(n \mid \theta) = \mathbf{v}_\theta(n)$. *Every individual is associated with a randomly initialized vector of 5 real numbers.*
- $\mathcal{I}(x \mid \theta) = \mathcal{I}(y \mid \theta) = \{\mathbf{v}_\theta(a), \ldots, \mathbf{v}_\theta(n)\}$
- $\mathcal{I}(S \mid \theta) : x \mapsto \texttt{sigmoid}(\texttt{MLP}_{S,\theta}(x))$, where $\texttt{MLP}_{S,\theta}$ has 1 output neuron.
- $\mathcal{I}(F \mid \theta) : x, y \mapsto \texttt{sigmoid}\left(\texttt{MLP}_{F,\theta}(x, y)\right)$, *where* $\texttt{MLP}_{\mathrm{F},\theta}$ *has 1 output neuron.*
- $\mathcal{I}(C \mid \theta) : x \mapsto \texttt{sigmoid}(\texttt{MLP}_{C,\theta(x)}$, *where* $\texttt{MLP}_{C,\theta}$ *has 1 output neuron.*

Together they form a *SatAgg* loss function, see Equation 4.5.

The learned embeddings are plotted in Figure 4.6[14]. It can be seen that the formula smoking implies cancer is inferred for entities in $\mathcal{E}_2$ and changed for entities f and g, as they are inconsistent with the rule set. After learning, the models $\texttt{MLP}_S$, $\texttt{MLP}_C$ and $\texttt{MLP}_F$ can be queried. For example, $\texttt{MLP}_C$ is queried to predict the predicate Cancer for the entities in $\mathcal{E}_2$. The truth values of formulae can also be queried. Figure 4.5 shows that satisfiability of the axioms and of the following formulae increases and converges during training

$$\begin{aligned} \phi_1: &\quad \forall p : S(p) \rightarrow C(p) \\ \phi_2: &\quad \forall p, q : (S(p) \vee S(q)) \rightarrow F(p, q) \end{aligned} \tag{4.6}$$

4.2.3. Knowledge Enhanced Neural Networks

Knowledge Enhanced Neural Networks (KENN) [48] integrate prior knowledge in the form of logical formulae into a neural network by adding *knowledge enhancement layers* to the network architecture. The purpose of these layers is to refine predictions in order to align them with the prior knowledge.

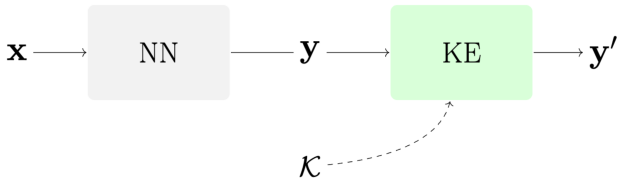


Figure 4.7.: The architecture of knowledge enhanced neural networks. A base neural network (NN) makes intial predictions that are updated by one or more knowledge enhancement layers (KE). The figure is taken from [48].

The architecture of KENN is illustrated in Figure 4.7. It essentially consists of two modules that are end-to-end differentiable. First, a *base neural network* implements a function that produces predictions $\mathbf{Y} \in \mathbb{R}^{n \times c}$ for c classes given some input data $\mathbf{X} \in \mathbb{R}^{n \times d}$ with feature dimension d. Second, one or more *knowledge enhancement layers* are stacked on top of the base neural network that are associated with a finite set of prior knowledge formulae $\mathcal{K}$. These knowledge enhancement layers aim to refine the predictions with respect to the knowledge in $\mathcal{K}$.

KENN expects formulae $\phi \in \mathcal{K}$ to be *clauses* which are disjunctions of k literals. They are based on a first-order logical language consisting of constants $\mathcal{C}$ and predicates $\mathcal{P}$. As the clauses in $\mathcal{K}$ formulate general knowledge they contain no constants and be universally quantified. Each knowledge enhancement layer implements a function that updates the predictions of the base neural networks's last layer. It therefore uses a *t-conorm boost function* $\delta_\phi : \mathbb{R}^k \to \mathbb{R}^k_+$ that increase the satisfaction of a formula ϕ measured in fuzzy logic, see Section 1.2.3. The softmax function is used as differentiable approximator of the maximum function, which represents disjunction in Gödel logic, see Section 1.2.3. The suggested *refinements* Δ^ϕ_{ij} to increase the satisfaction of a clause ϕ are calculated as

$$\Delta^\phi_{ij} = \delta^\phi(\mathbf{Z})_{ij} = w_\phi \cdot \text{softmax}(\mathbf{Z})_i = w_\phi \cdot \frac{e^{\mathbf{Z}_{ij}}}{\sum_{l=1}^{q} e^{\mathbf{Z}_{il}}}. \tag{4.7}$$

The refinements $\Delta^\phi \in \mathbb{R}^{n \times q}$ are computed on the matrix of preactivations $\mathbf{Z}$ before applying the activation function σ. This is done to ensure that the final predictions lie in $[0, 1]$. The t-conorm boost function δ^ϕ in Equation 4.7 is applied to each cell $\mathbf{Z}_{ij}$ and operates on the values $j \in \{1, \ldots, q\}$ contained in a row. A row has q prediction classes that correspond to the number of unary predicates in the logical language. The parameter w_ϕ denotes a *clause weight* that is associated with each clause in $\phi \in \mathcal{K}$. The clause weights are optimized during training and can be interpreted as the importance of a clause for improving the final predictions. The refinements Δ^ϕ from each clause $\phi \in \mathcal{K}$ are aggregated and added to the preactivations of the base neural network to produce the final predictions $\mathbf{Y}' \in \mathbb{R}^{n \times q}$:

$$\mathbf{Y}' = \sigma\left(\mathbf{Z} + \sum_{\phi \in \mathcal{K}} \Delta^\phi\right). \tag{4.8}$$

In the learning process, the loss function is based on the given labels and the updated predictions $\mathbf{Y}'$ instead of the predictions of the base neural network $\mathbf{Y}$. This way, the clause weights are part of the optimization.

Example 4.2.5 (Knowledge Enhanced Neural Networks). Recall the Smoker-Friends-Cancer Example in Section 4.2.4. Here, it is adapted to formulate a multi-label classification task where the classes are not mutually exclusive but have dependencies. Constants denote people annotated with a feature vector $\mathbf{x} \in \mathbb{R}^d$. They are categorised whether they are smokers $S(x)$ and whether they have cancer $C(x)$. The following clause ϕ_0 : $\forall(\mathrm{x}) : \neg\mathrm{S}(\mathrm{x}) \vee \mathrm{C}(\mathrm{x})$ describes a dependency between the predicates S and C. The base neural network returns truth values $\mathbf{Y}$ with their preactivations $\mathbf{Z}$, e.g. $\mathbf{Z}_{[a,C]} = 0.5$, $\mathbf{Z}_{[a,S]} = 0.3$ for a constant a. These are fed into the t-conorm boost function for clause ϕ_0 to determine Δ^{ϕ_0} with $w_{\phi_0} = 0.5$. The grounded literal $\neg S(a)$ is negated, so its preactivation is multiplied by -1. This leads to the following refinements:

$$\Delta_a^{\phi_0} = 0.5 \cdot \begin{bmatrix} 1 & -1 \end{bmatrix} \odot \mathrm{softmax}\left(\begin{bmatrix} 0.5 \\ 0.3 \end{bmatrix}\right)^T = 0.5 \cdot \begin{bmatrix} \frac{e^{0.5}}{e^{0.5}+e^{0.3}} & -\frac{e^{0.3}}{e^{0.5}+e^{0.3}} \end{bmatrix} \tag{4.9}$$

Then, the updated preactivations with the refinements are calculated as

$$\mathbf{Z}'_{[a,C]} \approx 0.75$$
$$\mathbf{Z}'_{[a,S]} \approx 0.06.$$

It can be seen that the knowledge enhancement layer acts on the preactivations by increasing the vallue for the atom $C(a)$ and decreasing the value for $S(a)$.

4.2.3.1. Application to Graph Data

In the architecture of KENN presented above, the knowledge enhancement layer operates on a matrix of preactivations that serves as grounding for unary predicates. Binary predicates must be taken into account in order to handle graph data. The principles of the knowledge enhancement through a t-conorm boost function remain unchanged. In [46] changes to the data structure are proposed to take binary predicates into account. The set of clauses is divided into unary and binary clauses: $\mathcal{K} = \mathcal{K}_U \cup \mathcal{K}_B$. Unary and binary predicates are denoted as $P_U \in \mathcal{P}_U$ and $P_B \in \mathcal{P}_B$ *Binary clauses* contain two different variable, for example $\phi_1(x, y) : \forall xy : \neg S(x) \vee \neg F(x, y) \vee S(y)$ with the binary predicate Friends $F(x, y)$.

For unary predicates, the rows in the preactivation matrix $\mathbf{Z}$ correspond to the constants and the columns to the unary predicates. Thus, each possible grounded atom corresponds to a cell in the matrix. The key challenge with binary predicates is that the grounding of a binary predicate refers to *two* constants instead of one. Therefore, binary groundings are expressed as a matrix $\mathbf{Z}_B \in \mathbb{R}^{m \times |\mathcal{P}_B|}$. Each row has a pair of keys (s_x, s_z) and each cell in the matrix corresponds to the grounding of a binary predicate. It has binary keys (s_x, s_z) per row and a value as grounding for a binary predicate $P_B(x, y)$.

To bring unary and binary predicates into a representation to which the t-conorm boost function can be applied, unary predicates are *binarised*. A unary predicate P_U is formalized as two binary predicates $P_B^x(x, y)$ and $P_B^y(x, y)$, where only the first (second) entry matters. For example, $\phi^1(x, y)$ is binarised to $\forall xy : \neg S^x(x, y) \vee \neg F(x, y) \vee S^y(x, y)$. Then, unary and binary predicates are combined into a matrix on which the clause enhancers for binary clauses can operate. Therefore, the keys of the unary matrix are *joined* on the keys of the matrix with the binary predicates (s_x, s_y). It can happen that two groundings of a binary clause share a common grounded term. For example, $\phi^1(a, b) = \neg S(a) \vee \neg F(a, b) \vee S(b)$ and $\phi^1(b, c) = \neg S(b) \vee \neg F(b, c) \vee S(c)$ share the grounded atom $S(b)$. In this case, the refinements provided by both groundings are aggregated and added up for identical keys, corresponding to the following *group-by* operation for a repeated grounded unary predicate $P_U(a)$:

$$\Delta_{P(a)} = \sum_{b \neq a} \left(\sum_{\substack{\phi \in \mathcal{K}_B \\ P^x \in \phi}} w_\phi \cdot \Delta^\phi_{[a,b],P^x(a,\cdot)} + \sum_{\substack{\phi \in \mathcal{K}_B \\ P^y \in \phi}} w_\phi \cdot \Delta^\phi_{[b,a],P^y(\cdot,a)} \right). \tag{4.10}$$

The notation $\Delta^\phi_{[a,b],P^x(a,\cdot)}$ indicates that the row for the constant pair $[a, b]$ and the column for the binarized unary predicate $P^x(a, \cdot)$ from Δ^ϕ are selected. Refinements from predicates that do not appear in clause are set to zero. Finally, to obtain the updated predictions $\mathbf{Y}'$, the refinements from unary clauses $\Delta_{\mathcal{K}_U,P(a)}$, and from binary clauses $\Delta_{\mathcal{K}_B,P(a)}$ for a grounded predicate $P(a)$ are added to the preactivations $\mathbf{Z}_{P(a)}$ and are activated with σ:

$$\mathbf{Y}'_{P(a)} = \sigma \left(Z_{P(a)} + \sum_{\phi \in \mathcal{K}_U} \Delta^\phi_{P(a)} + \sum_{\phi \in \mathcal{K}_B} \Delta^\phi_{P(a)} \right). \tag{4.11}$$

Example 4.2.6 (Knowledge Enhanced Neural Networks with Binary Predicates). With these changes, the binary predicate `Friends` is introduced to the Smoker-Friends-Cancer Example above [46]. The example is illustrated in Figure 4.8. The domain consists of three constants (people) $C = \{a, b, c\}$ and the unary predicates $\mathcal{P}_U = \{S, C\}$ and the binary predicate $\mathcal{P}_B = \{F\}$. The constants are described as a graph, as shown below, where the edge weights represent the groundings for the binary predicates and the initial predictions represent the groundings for the unary predicates. In Figure 4.9, the unary groundings $\mathbf{Z}_U$ and binary groundings $\mathbf{Z}_B$ are denoted as matrices. The join operation represents the binarization of the predicates $S(x)$ and $C(x)$ to $S^x(x, y), S^y(x, y)$ and $C^x(x, y), C^y(x, y)$ so that they are represented in a matrix with the binary groundings. The knowledge enhancer for binary clauses is applied to the joined matrix shown in Figure 4.9. The group-by operation collects the refinements that refer to common grounded atoms. In case the set of unary clauses is not empty, refinements computed from unary clauses would be added, as described in Equation 4.11.

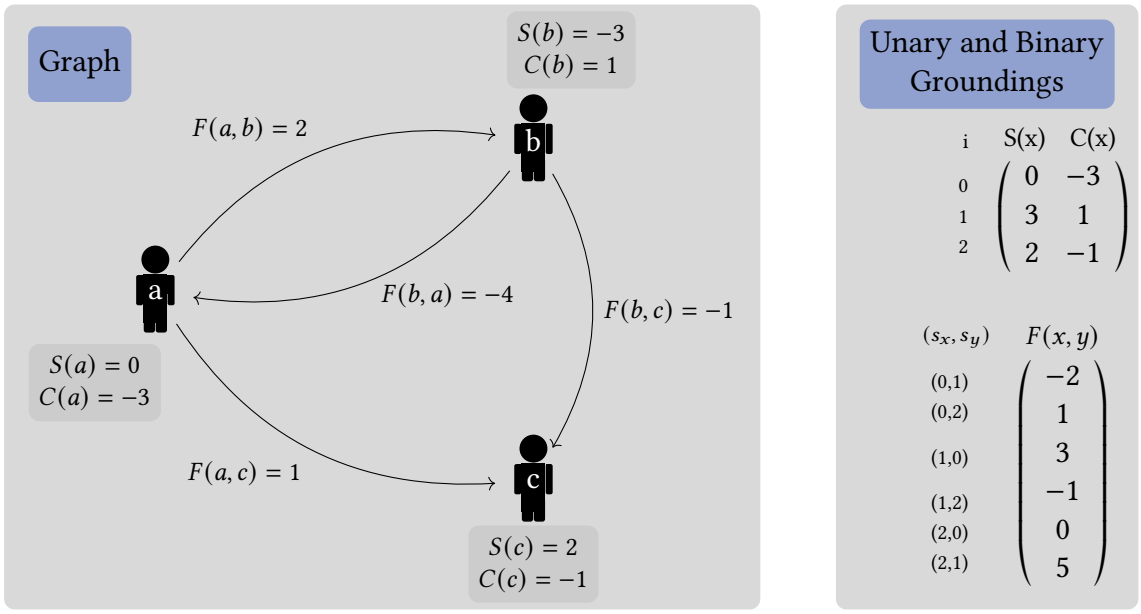


Figure 4.8.: Illustration of KENN on the Smoker-Friends-Cancer example. *Left:* The graph of three people a, b, c with preactivation values for `Smoker` (S) and `Cancer` (C) as well as `Friendship` (F). *Right:* The interpretation of the graph as grounded unary and binary predicates.

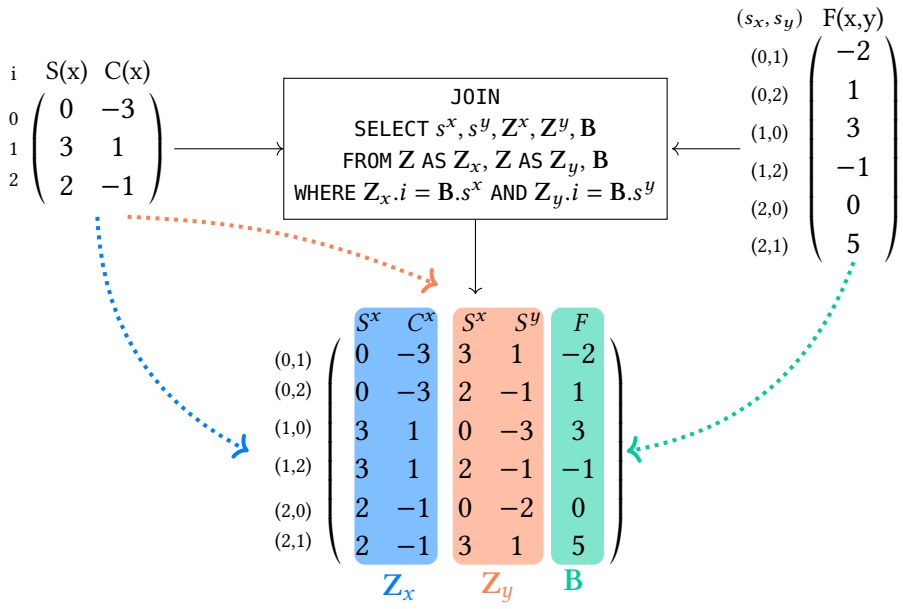


Figure 4.9.: Illustration of the binarization of the unary predicates S and C to S^x, S^y and C^x, C^y and their join with F on the keys (s_x, s_y).

4.2.4. Conclusion

The previous section introduced three approaches in the field of neuro-symbolic integration.

Regarding graphs, all of them offer the possibility to encode binary predicates and thus to model relations in their logical language. It is shown that LTN is theoretically suitable for learning embeddings for entities in graphs. Scallop was tested in applications involving reasoning with binary predicates. Scallop was used in the context of a reasoning task on kinship relations. KENN was used to refine predictions for unary predicates by incorporating information from binary clauses.

However, open questions remain when applying these methods to graphs. The first open question is how to consider the open-world assumption on graphs when some information is not explicit and negation has to be taken into account. Related to this, the second major concern is scalability on large graphs with a high number of entities and relations.

For LTN, the grounding process can be expensive for a large number of entities and facts. Also, a large number of axioms leads to a complex loss function. LTN also requires to ground negated statements, that are usually not explicitly encoded in knowledge graphs. For Neural Probabilistic Programming, the inference process relies on the closed world assumption. The resolution of the logic program at inference can also become challenging when many grounded terms have to be considered, including negated terms. Furthermore, to obtain the probabilistic facts from the neural predicates, an exponentially growing number of entity pairs must be evaluated, which is infeasible for large graphs. In KENN, the matrix with groundings grows quadratically (n^2) with the number of possible combinations of n nodes in the graph. Since the representation is dense, a truth value is needed for each pair of nodes that are *not* connected.

In conclusion, the use cases presented are based on small, limited problems, resulting in dense graphs that are fully connected or even toy examples with a small number of entities and relations. They do not resemble real-world graphs such as knowledge graphs. For this reason, the application of the presented neuro-symbolic methods to them still needs to be addressed.

4.3. Neuro-Symbolic Reasoning on Graphs

Since the methods of the previous section have some major drawbacks when applied to large graphs, this section presents neuro-symbolic methods that are developed specifically for knowledge graphs. These methods are organised into the following categories: (1) *rule learning*, (2) *knowledge-driven graph augmentation*, (3) *knowledge as constraints on the embedding space*, and (4) *knowledge as regularization terms in the loss function*. For a broader overview, the following surveys are recommended [52, 222].

4.3.1. Rule Learning

For the sake of completeness, the first category concerns rule learning approaches. Methods in this category leverage neuro-symbolic techniques to extract rules from data. The majority of these methods train a neural component to obtain rule confidence scores or guide the rule mining strategy. Some exemplary methods in this category are *ExpressGNN* [94], *pLogicNet* [167], *pGat* [85], *ItEr* [89], *RNNLogic* [168], *Drum* [176] or *Neuro-Symbolic Class Expression Learning* [53]. As the focus of this thesis lies on the integration of prior knowledge and not on the extraction of knowledge in form of rules, methods of this category are not presented in detail.

4.3.2. Knowledge-driven Graph Augmentation

Models in the category knowledge-driven graph augmentation use reasoning iteratively in the training process. The purpose of the symbolic module is to *augment* the graph with additional facts on which the sub-symbolic module is trained. The symbolic and sub-symbolic components are applied sequentially or can interact with each other. Typically, the sub-symbolic methods used here are knowledge graph embedding methods, as mentioned in Section 3.1, combined with a symbolic reasoning module.

In *KGE**[106] a reasoner and a knowledge graph embedding model inform each other by iteratively feeding the output of one component as input to the other. The reasoner is based on a set of Horn rules that encode user-specific or ontological domain knowledge. Starting from the rules and the explicit facts, the reasoner applies forward chaining until a fixpoint is reached, in order to augment the explicit facts with inferred facts. The score function of a knowledge graph embedding method is used to evaluate the plausibility of the explicit and inferred facts. The most plausible facts are fed back to the reasoner and the process is repeated iteratively. To reduce the number of facts to be evaluated by the knowledge graph embedding, KGE* only scores the most relevant facts. To determine the relevance of facts, densely connected areas in the graph are taken into account using clustering techniques. In the experiments with KGE* [106] the knowledge graph embeddings TransE [26], HolE [156], ComplEx [194], RotatE [188] and DistMult [214] and the reasoner Pellet [186] are used. However, KGE* is described as a model agnostic framework that does not depend on a specific knowledge graph embedding method or reasoner. In terms of prior knowledge, KGE* is restricted to monotonic rules to ensure that the inferred facts never contradict the existing facts. It focuses only on the generation of positive facts. It also expects a set of predefined hard rules.

Similar to KGE*, *UniKer* integrates a knowledge graph embedding method with a reasoner in an iterative manner to infer implicit facts. The number of inferred facts depends on the number of explicit facts. Motivated by this, knowledge graph embeddings are used to find more plausible facts to be given to the reasoner and to increase the coverage of the rules. While KGE* only increases the number of facts, UniKer extends the method by also *removing* the least $\lambda\%$ plausible facts with the lowest scores. This allows UniKer to be robust to noise in the form of incorrectly introduced facts and to avoid the propagation

of contradictions by the reasoner. Since the number of facts to be potentially evaluated is large, UniKer uses *lazy inference*. Only the facts are scored that occur in the body of rules and are potentially useful to the reasoner. In addition to KGE*, UniKER focuses on monotonic Horn rules. UniKer does not rely on the availability of prior knowledge, as it uses automatically generated rules from AMIE+ [66]. UniKer is agnostic to the knowledge graph embedding method. However, in the experiments [36] TransE, DistMult and RotatE are used.

While the above methods focus on the set of positive facts, *ReasonKGE* [102] uses prior knowledge to generate a set of negative facts in a more reliable way. It iteratively identifies inconsistent predictions by a knowledge graph embedding model through *consistency checking*. As noted in Section 3.1.3, negative facts are usually sampled randomly under the local closed-world assumption. This does not prevent the introduction of potentially false negative facts. ReasonKGE iteratively generates a set of negative facts based on the ontology, the explicit facts and a knowledge graph embedding model. The method starts with standard training of a knowledge graph embedding model based on random negative sampling [26]. Then a consistency check based on DLITE [11], a lightweight extension of description logics is performed to identify inconsistent predictions. Once inconsistent predictions are found, facts that have a similar neighbourhood to the inconsistent fact are sampled, resulting in a set of negative facts. This procedure is called *dynamic sampling*. The set of negative facts is then fed back into the next training iteration of the knowledge graph embedding model.

4.3.3. Knowledge as Constraints on the Embedding Space

Another way of introducing knowledge is in the form of constraints on the embedding space. In fact, the inference patterns covered by knowledge graph embeddings can already be seen as such constraints. For example, in DistMult all relations are modelled as symmetric relations, which already represents the encoding of prior knowledge in the embedding space. However, it is not obvious how to introduce general and complex knowledge into the embedding space. In addition, this strong bias towards a particular inference pattern may sometimes not be desirable.

The translational embedding method *BoxE*, see Section 3.1, goes further in this direction and allows the explicit injection of inference patterns as prior knowledge. This method is in the following referred to as *BoxE with rule injection (BoxE + RI)*. A rule $\eta \leftarrow B$ is *injected* if the model is configured to force η to hold whenever the expression in the body B holds [3]. To achieve this, a strong bias is introduced into the embedding model by manually adjusting the boxes of some relations according to the injected rules. This bias ensures that the rule is enforced even during inference. To inject symmetry and inversion rules for relations r_1 and r_2, their boxes are constrained to $\mathbf{r}^{(1)} \overset{!}{=} \mathbf{r}^{(2)}$ for symmetry, or $\mathbf{r}_1^{(1)} \overset{!}{=} \mathbf{r}_2^{(2)}$ and $\mathbf{r}_2^{(1)} \overset{!}{=} \mathbf{r}_1^{(2)}$ for inversion. It is shown in [3] that hierarchy and intersection rules can also be enforced. However, the injection of negated and composed rules is not supported.

4.3.4. Knowledge as Regularization Terms in the Loss Function

Methods in this category introduce knowledge through the loss function. The strategy is to penalize solutions that are inconsistent with the knowledge and thus provides incentives to find knowledge compliant representations during optimization.

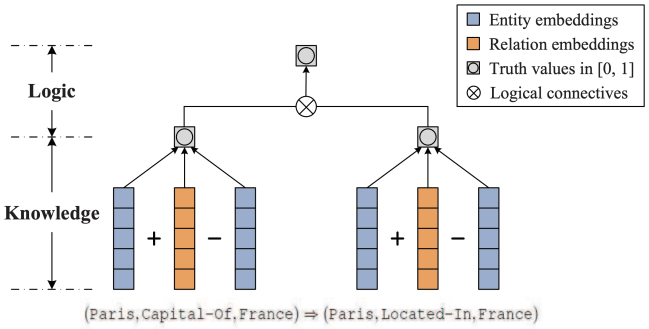


Figure 4.10.: Visualisation of KALE. The facts are interpreted as grounded atoms and normalized scores as their truth value. Logical operators are represented in fuzzy logic. The figure is taken from [77].

TransOWL [42] builds on TransE and encodes knowledge as terms in the loss function. The knowledge comes from an RDFS schema and typically includes class information, relational equivalence, inversion and hierarchy. The penalty terms take into account how TransE models facts in the embedding space, where the representation of the tail entity corresponds to the representation of the head entity plus the relation vector. In line with this, the term $\lambda \sum_{r_1=r_2 \in \mathcal{T}_{\text{equiv}}} \|\|\mathbf{r_1} - \mathbf{r_2}\|\|$ is added to the score function of TransE, where the relations r_1 and r_2 that are known to be equal ($r_1 = r_2$), For inverse relations, $-\lambda \sum_{r_1=r_2^- \in \mathcal{T}_{\text{inv}}} \|\|\mathbf{r_1} + \mathbf{r_2}\|\|$ is added to the score function. The parameter λ controls the impact of the penalty term. This way, relations are penalized that do not correspond to the rule in the embedding space and increase the overall loss. The goal is to push the representations towards vectors that satisfy the rules in the vector space. TransOWL also criticises uniform negative sampling, see section 3.1.3, and highlights the risk of generating false negatives. To mitigate this issue, TransOWL uses a reasoner [103] to introduce negative facts based on the knowledge provided. With this feature, TransOWL carries elements of the methods in the category knowledge-driven graph augmentation in Section 4.3.2.

Joint Embedding of Instances and Ontological Concepts (JOIE) [82] considers knowledge graphs from both an *instance view* and an *ontological view*. While the instance view considers the facts in a knowledge graph, the ontological view considers general knowledge for the relations and entities in the fact set. Facts and general knowledge are embedded in separate vector spaces called *intra-view models*. The knowledge graph embedding models TransE, HolE [156] and DistMult are used as intra-view models in JOIE. In addition, a *cross-view model* $f_{CT}(\cdot)$ bridges the embeddings of both spaces, thereby linking instance embeddings with concept embeddings. In the cross-view model, the representation of an entity is trained to be close to the representation of the concept type, see Figure 4.11.

Special attention is drawn to modeling hierarchies for the property `subclass_of` in the ontological view embeddings. Therefore, an additional loss term is introduced that models pairs of hierarchical concepts with a non-linear transformation between class and subclass concepts. The loss function of JOIE is composed of the losses of the intra-view models

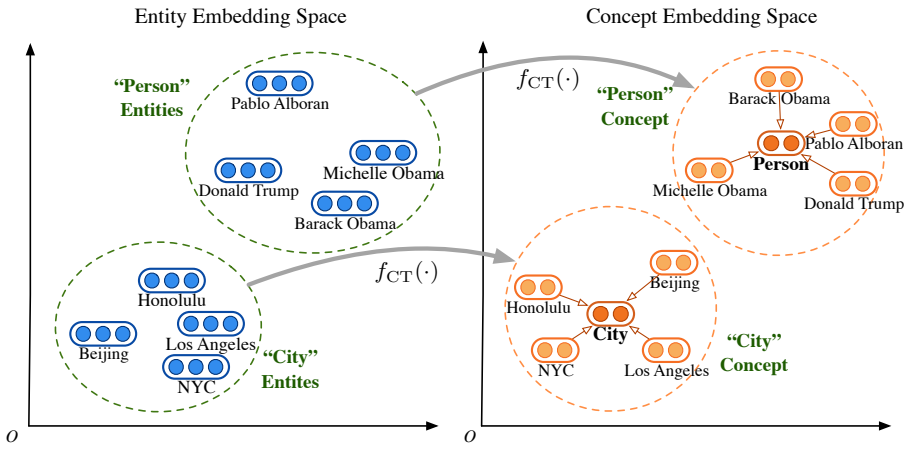


Figure 4.11.: Visualisation of JOIE. The embeddings in the intra-view space, namely entity embedding space and concept embedding space, are translated with a cross-view transformation function f_{CT}. The figure is taken from [82].

(including the hierarchy term) and the loss of the cross-view model.

KALE (Embeddings by jointly modeling Knowledge And Logic) [77] introduces prior knowledge to knowledge graph embeddings by jointly embedding the *grounded formulae* and *facts* of a knowledge graph in the same latent space. KALE is visualised in Figure 4.10. In contrast to the previous mentioned approaches TransOWL and JOIE that focus on simple RDFS schema rules, KALE supports first-order logic rules. The facts in the knowledge graph are interpreted as grounded atomic formulae in first-order logic. KALE employs TransE [26] to compute scores for these grounded atomic formulae. The entities are constrained so that the truth values returned by TransE fall into $[0, 1] \subset \mathbb{R}$ to mimic truth values. To represent complex formulae in the same space, KALE encodes logical operators in fuzzy logic. In this way, truth values of grounded complex formulae can be composed of atomic formulae. The larger a truth value, the better a grounded rule is satisfied. In the following, a global loss over the satisfaction of the grounded formulae in fuzzy logics and the atomic formulae as facts in the graph can be defined. KALE relies on a margin-based ranking loss. Uniform random sampling is not only conducted to obtain negative facts, but also to obtain negative grounded rules [26].

RUle-Guided Embedding (RUGE) [78] learns knowledge graph embeddings iteratively through the guidance from soft rules. It takes as input the sets of explicit facts, implicit facts, and soft rules with confidence scores. The soft rules and their confidence scores are learned with AMIE [65]. Per iteration, RUGE alternates between a *soft label prediction stage* and a *embedding rectification stage*. The goal of the soft label prediction stage is to find scores for the implicit facts. ComplEx is used as the knowledge graph embedding

	Name	r	s	k	i	knowledge
General Frameworks	DeepProblog [139]	✓	✗	✓	✓	FOL formulae
	Scallop [95]	✓	(✗)	✓	✓	FOL formulae
	LTN [14]	✓	(✓)	✓	(✓)	FOL formulae
	KENN [48]	✓	(✓)	✓	(✓)	FOL clauses
Knowledge-driven graph augmentation	KGE* [106]	✗	(✗)	✓	(✗)	Horn rules
	UniKer [36]	✓	(✗)	✓	(✗)	Horn rules
	ReasonKGE [102]	✓	(✗)	✓	(✗)	Horn rules
Knowledge as constraints on the embedding space	BoxE+RI [3]	✓	✓	✓	(✓)	Hierarchy Inversion Symmetry Intersection
Knowledge as Regularization on the Loss function	TransOWL [42]	✓	✓	✓	✗	Symmetry Equivalence Inversion
	JOIE [82]	✓	✓	✓	✗	Hierarchy Types
	KALE [77]	✓	(✓)	✓	✗	FOL formulae
	RUGE [78]	✓	(✓)	✓	✗	Horn rules

Table 4.2.: Overview of the neuro-symbolic methods presented in this thesis with respect to the desiderata of neuro-symbolic AI: **r**obust, **s**calable, **a**ccurate, **k**nowledge-aware, **i**nterpretable.

model. They should match the scores produced by ComplEx and at the same time match the rules as much as possible. Given rules with confidence and embeddings from the previous iteration, the goal is to predict a soft label for the implicit fact using the current embeddings and groundings of the rules. The grounded rules are used as constraints on the soft labels. The conditional truth values of the grounded rules given the soft labels are recursively computed using product fuzzy logic. The confidence scores are used to set the tolerance for a rule violation. For the purpose of scalability, RUGE only grounds facts that occur in the body of a rule, as described for UniKer. In the embedding rectification stage, the labelled and unlabelled facts (with their known labels and soft labels from the previous step) are used to update the embeddings. A global loss over explicit and implicit facts is minimized. The goal is to align the final embeddings on the one hand with the known labels but also with the knowledge coming from the rules injected into the soft labels.

4.4. Summary and Perspective

With regard to the presented methods, an important question is whether the combination of symbolic and sub-symbolic methods is successful in bringing the best components of both fields. It is now discussed qualitatively whether the proposed methods meet the desiderata *robust*, *scalable*, *interpretable*, *knowledge-based* and *accurate* defined in Section 4.1. A summary is presented in Table 4.2. A fair assessment of the *accurate* criterion would require a quantitative analysis, which is beyond the scope of this section. Therefore, this criterion will not be discussed.

Knowledge-aware. All of the methods presented enable the incorporation of prior knowledge. However, the expressiveness of the knowledge to be encoded as rules in the model differs significantly. The general frameworks DeepProblog, Scallop and LTN accept expressive first-order logical rules including quantifiers and aggregators, while KENN is limited to universally quantified clauses in first-order logic. DeepProblog is based on Prolog which only uses Horn clauses.

The knowledge used in knowledge-driven graph augmentation approaches is monotonic Horn rules, which allow additional facts to be inferred and augment the initial graph.

Methods that fall into the category of knowledge as constraints on the embedding space or in the loss function are mostly limited to simple inference patterns typically found in RDFS schemas. They refer to patterns that should hold for certain relations, such as symmetry, hierarchy, inversion and relational equivalence. However, KALE uses fuzzy logic to contribute to the satisfaction of first-order logic rules in the loss function. RUGE uses possibly soft Horn rules with confidence values at the soft label prediction stage.

Robust. In terms of robustness, the methods differ in the way they can handle errors and noise in the input data. The general frameworks KENN and LTN are rather robust to noise in the input data, since the logic formulae are fully translated to differentiable neural network components in the training phase. In particular, KENN uses trainable clause weights for each rule, which can learn to ignore apparently incorrect rules. In neural probabilistic programming approaches, the neural predicates are also noise-tolerant to the input data.

Knowledge-driven graph augmentation approaches rely on a symbolic reasoner, which is not noise tolerant by default and carries the risk of propagating errors in the program leading to false inferences. Furthermore, non-monotonic Horn rules do not allow for contradictions with the previously obtained set of facts, which does not allow for error correction. KGE* relies on a simple reasoner and is therefore not robust to noise, since the reasoner could amplify the noise by propagating wrong facts. In contrast, UniKer not only infers new facts, but also removes the least plausible facts. In this way, inconsistent facts can be removed. In ReasonKGE, facts must pass a consistency check. Facts that are likely to be false are included as negative facts. This makes ReasonKGE robust to noise.

The methods from the categories knowledge as constraints on the embedding space embedding and knowledge as regularization term in the loss function essentially represent knowledge in the embedding space and are therefore robust to noise.

However, robustness to noise means that the performance of a model remains stable or is only slightly affected in the presence of noisy input data. The amount and type of noise that a model can tolerate needs to be investigated experimentally. Even sub-symbolic models that rely on a training stage can be affected by dominant noise in the data.

Interpretable. Another desirable feature of neuro-symbolic methods is interpretability. Among the methods presented, neural probabilistic programming approaches are the only category that use reasoning with a logic program at inference. Therefore, the reasoning process remains interpretable and the symbolic knowledge is not lost in the translation to the embedding space. However, KENN and LTN integrate rules into the model and the loss function during training. Particularly in LTN, it is difficult judge the effect and interaction of the logic components after inference. The same disadvantage is present in KALE and RUGE. In KENN, the clause weights can still give some insight into the importance of rules, but there is no guarantee that they will hold in the predictions at inference.

For the knowledge-driven graph augmentation approaches, the rules affect the training phase and are not included at inference. Furthermore, the impact of a rule is hard to be quantified. For example, if the body of a rule is never true, zero additional facts will be generated by that rule. In this case, the rule has no effect. For this reason, the interpretability of methods in this category is limited. In BoxE+RI, the boxes for some relations that follow rules are set manually. This introduces a strong bias that provides predictable inference and interpretable decisions with respect to the affected relations. However, the decisions about other relations may not be interpretable, and the rule language of the injected rules is limited to simple inference patterns.

TransOWL basically introduces knowledge as constraints in the loss term during training. The parameter λ controls the influence of knowledge. However, this does not lead to reliable predictive inference or more interpretable representations than in TransE. Similarly, the pure vector representations learned in TransOWL and JOIE are not easier to interpret than pure sub-symbolic knowledge graph embedding vectors, especially when the embedding dimension is high.

Scalable. DeepProblog has to solve a reasoning problem in both training and inference, which is computationally expensive when many possible worlds have to be considered. Scallop tackles this problem by proposing approximated proof evaluation with the top-k algorithm. However, the costly reasoning process in training and evaluation remains.

UniKer, ReasonKGE and KGE* use a reasoner during training. While these components are costly at training, the methods rely on the trained embeddings and scale well at inference.

TransOWL and BoxE+RI, which used embedding and loss function constraints, scale well during training and inference. RUGE, KALE and LTN need to ground universally quantified rules during training, which can be expensive, but this step is not necessary at inference. The time complexity of KENN depends on the rules and predicates considered. Since the clauses are assumed to be independent, their enhancement calculation can be parallelized, leading to a logarithmic time complexity depending on the number of entities and clauses [46].

Part III.

Contribution

Outline of Contribution

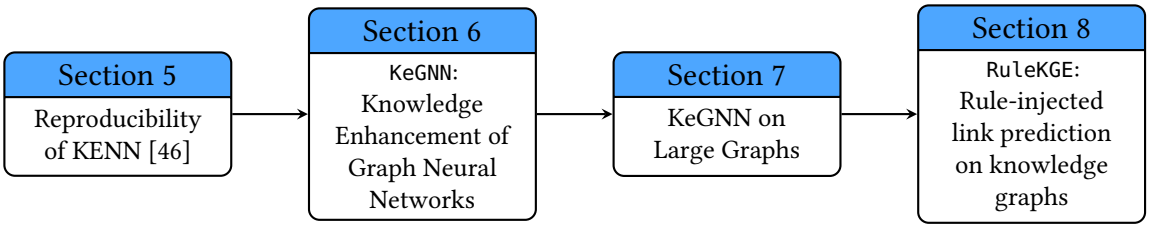


Figure 4.12.: Outline of the Contribution

The main contributions of this thesis are illustrated in Figure 4.12. Since the work of Knowledge Enhanced Neural Networks for relational domains [46] is relevant to this thesis, **Chapter 5**, deals with the reproducibility of the experiments and results obtained with the method. Therefore, the experiments are reimplemented and then reproduced, replicated and reevaluated. The aim of these steps is to ensure that the reimplementation is reliable for further extensions. In addition, general lessons are drawn to improve the reproducibility of machine learning methods.

In **Chapter 6**, the method *Knowledge Enhanced Graph Neural Networks (KeGNN)* is proposed, which uses knowledge enhancement layers in the context of graph neural networks. KeGNN can be seen as a variant of Knowledge Enhanced Neural Networks. In previous work, knowledge enhancement layers are used with an MLP, which is not powerful enough to capture the graph structure in the representations. In contrast, graph neural networks can also capture relational information at the level of the basic neural network, thus increasing the overall model capacity. The effectiveness of the KeGNN is investigated in experiments on various graph data sets.

While the previous chapters focus on knowledge enhancement on small graphs, **Chapter 7** deals with the applicability of knowledge enhancement layers to large graphs. A method called *Restrictive Neighbourhood Sampling (RNS)* is introduced to solve the problem of an exponentially increasing number of nodes with respect to the number of knowledge enhancement layers. In addition, experiments with knowledge enhancement layers on large graphs are carried out on benchmark datasets from the Open Graph Benchmark [93].

Previous methods have mainly dealt with node classification and homogeneous graphs under the closed-world assumption. However, most knowledge graphs are multi-relational and often incomplete. In this context, **Chapter 8** presents the neuro-symbolic method *RuleKGE* that focuses on heterogeneous knowledge graphs viewed under the open-world

assumption. It combines a symbolic reasoner to generate positive and negative facts that are integrated into the training of knowledge graph embeddings for link prediction. In the experiments, the approach is evaluated with different rule sets on the Family dataset.

5. Reproducibility Study on Knowledge Enhanced Neural Networks

This chapter focuses on the reproducibility of Knowledge Enhanced Neural Networks (KENN) introduced in Section 4.2.3 and the experiments with binary clauses [46]. The motivation behind this is to extend KENN to larger graphs. To this end, we re-implement the system available in Tensorflow in the graph-specific library PyTorch Geometric [61]. There are various other reasons for reimplementing a system besides for an extension [192, 8]. For that purpose, we provide a progressive and general approach for extending any machine learning method by first going through *reproducing*, *replicating* and *reevaluating* its results. This approach aims at ensuring that the reimplementation is reliable. It is tempting to skip these steps and jump directly to the extension of the method. On the contrary, we argue that this approach allows to better understand potential problems at the specific stage where they occur. Further, we identify the obstacles encountered when conducting these steps and how they may be overcome. The particular case on which we detail and demonstrate our approach allows us to derive general lessons learned. In general, this work can contribute to improve reproducibility in the field of machine learning.

The work in this chapter is published [208].

> Werner, L., Layaïda, N., Genevès, P., Euzenat, J. and Graux, D. (2024). Reproduce, Replicate, Reevaluate. The Long but Safe Way to Extend Machine Learning Methods. Proceedings of the AAAI Conference on Artificial Intelligence, 38(14), 15850-15858. https://doi.org/10.1609/aaai.v38i14.29515

5.1. Reproducibility in Machine Learning

Experiment reproducibility is a cornerstone of scientific research. It is so for the experimenters, because it allows them to be more confident in the results they claim. It is so for the research community, because it allows other researchers to stand on a solid ground in developing their own work. This leads to more reliable and superior research results, which is beneficial for the society as a whole.

In the machine learning community, strong incentives emerge to provide the necessary information to reproduce results [164]. More generally, this is a basic open science requirement to increase accountability and reproducibility. Reproducibility guidelines

	Same software	Different software
Same datasets	**Reproduce**	**Replicate**
Different datasets	**Reevaluate**	

Table 5.1.: Overview of the steps reproduce, replicate and reevaluate

and checklists focus on encouraging researchers to provide more detailed documentation of their published work [99, 98, 1]. However, it is not obvious that such guidelines are sufficient to independently reproduce experiments. Indeed, random operations (non-determinism) may be introduced in several places [33], hyperparameters may change the behavior of models [87, 137], components may exchange information in imprecise ways, the data processing pipeline may be overlooked, etc. [135, 64]. All these factors can compromise the reproducibility of experiments.

Various terminologies are considered for reproducibility in computer science [58, 72, 174, 165, 75]. We use and enrich the terminology adopted by the ACM [6], as summarized in Table 5.1:

- **'repeat'** means reexecuting an experiment with the same code, the same parameters, the same data by the same experimenter;
- **'reproduce'** means performing an experiment with the same software, the same parameters and the same data but by a different experimenter;
- **'replicate'** means performing an experiment independently on the same data but using different software and by a different experimenter;
- to these, we add **'reevaluate'** which means performing an experiment independently on different data.

When necessary, 'reproducibility' is also used as a term covering all such activities.

Given the high degree of non-determinism of machine learning techniques in general, the problem of reproducing results has gained importance with the increase of interest in the field. Recent surveys [76] show that even papers published at prestigious conferences are not sufficiently documented and report on a reproducibility crisis [74]. An empirical study [169] reveals difficulties in independently replicating published papers. Due to increased awareness, several conferences, including AAAI, promote reproducibility by encouraging authors to provide source code, experimental descriptions and datasets of their papers through reproducibility guidelines and checklists [164, 99, 1].

Further, there is a surge in documented reproduction and replication attempts, particularly encouraged through reproducibility challenges [164, 185, 140]. Such challenges ask interested individuals to replicate results from recent papers. Difficulties such as the lack of detailed documentation, e.g. hyperparameters [10] or even failure to reproduce [24] are frequently reported. Their motivation differs, as the goal in this chapter is not to achieve

reproducibility for its own sake. Here, experiments are reproduced to ensure that the reimplemented system retains the function of the knowledge enhancement in the initial implementation.

5.2. Experiments with Knowledge Enhanced Neural Networks

KENN [48] is a neuro-symbolic method that integrates a base neural network (NN) with knowledge enhancement layers in an end-to-end differentiable way. While KENN was tested on several use cases [48, 73], particular interest is drawn to experiments with binary clauses [46]. Since the aim is to extend KENN with more graph learning capabilities, it is necessary to ensure that the work stands on firm ground. For that purpose, the experiments performed by the authors are reproduced in this thesis.

KENN was applied to a node classification task on the Citeseer dataset [136], which is a homogeneous, node-attributed and labelled graph. The Citeseer dataset consists of scientific papers belonging to one of the six computer science research classes and citations among them. The dataset is modelled as a graph where nodes represent papers and edges represent citations. The prior knowledge provided to the knowledge enhancement layers encodes the assumption that two papers that cite each other have the same class. According to the pattern

$$\forall x \forall y : \neg \text{Class}(x) \vee \neg \text{Cite}(x, y) \vee \text{Class}(y), \tag{5.1}$$

one logical clause is defined for each class and forms a set of prior logic clauses. While the predictions of the base neural network are used as interpretation of the unary predicates representing the classes in the real-valued domain, the citations between two papers are known a priori. For this reason, the predictions of the binary predicate $\text{Cite}(x, y)$ are set to true (1.0) if an edges exists.

In [46], the authors report the performance of KENN for experiments on multiple training set sizes (10%, 25%, 50%, 75%, 90%). For each size, 100 independent runs are conducted. KENN is compared to the standalone base neural network and to Neural Machines (RNM) [142] and Semantic-Based Regularization (SBR) [56]. The experiments support the following hypotheses:

H1 KENN consistently outperforms the base neural network for all training set sizes.

H2 The performance gain due to the knowledge enhancement is larger when training data is limited.

H3 KENN leads to similar or superior results compared to RNM and SBR.

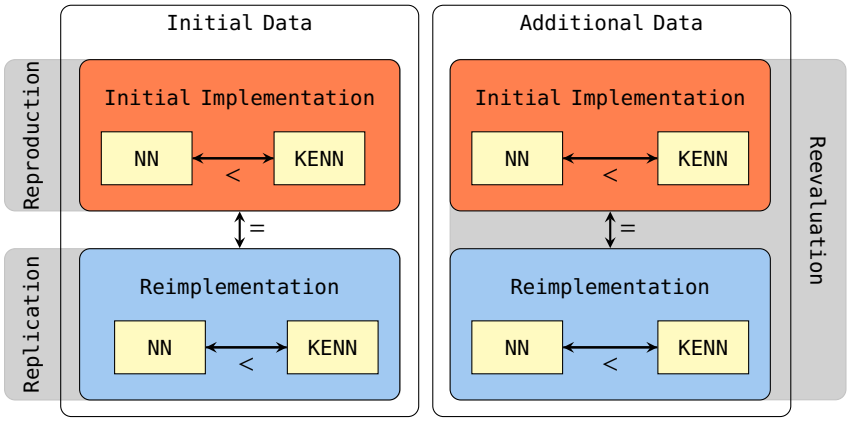


Figure 5.1.: Overview of the methodology

5.3. Methodology

We extend KENN in the framework PyTorch [161] in conjunction with the library PyTorch Geometric [61]. To this end, a comprehensive reproducibility study is the best way to ensure that KENN is extended on a reliable basis. With this objective in mind, as illustrated in Fig. 5.1, the following steps are conducted:

1. We **reproduce** the results obtained with the initial implementation;
2. We reimplement the initial experiments and **replicate** them;
3. We **reevaluate** the experiments on additional datasets.

For each step, we detail the required information, the obstacles we had to overcome and how we achieved to overcome them. Furthermore, we evaluate whether the KENN results are confirmed and report the lessons learned.

We refer to the following publicly available material. The reproduction and replication is based on the results reported in the paper [46] (that we call the *initial paper*) and on the reported experiments[1] (that we call the *initial experiments*). The experiments make use of the Python package KENN2[2],[3]. In addition, we also extract information from other related papers [48, 47]. We preserve, as far as possible, the input (dataset) and output format (results) of the initial experiment. In this thesis, the focus lies on the transductive experiments with KENN. In the following, we refer to our implementation as *reimplementation*.

All experiments in this thesis are conducted on a machine running an Ubuntu 20.4 equipped with an Intel(R) Xeon(R) Silver 4114 CPU 2.20GHz processor, 192G of RAM and one GPU Nvidia Quadro P5000. The reimplementation of the experiments is available at a dedicated

[1] https://github.com/rmazzier/KENN-Citeseer-Experiments
[2] https://github.com/DanieleAlessandro/KENN2
[3] https://github.com/HEmile/KENN-PyTorch

Git repository[4] under the hash `a894cae297b47f6d1acfa8e8dab99603f7b5e996`. It contains the following elements: (*a*) the source code, (*b*) the execution instructions, (*c*) the software requirements, (*d*) the result evaluation script, (*e*) the raw files of the experiment results. Further, the adaptions of the initial implementation to Pubmed and Cora is included.

5.4. Evaluation Criteria

In order to judge whether we have successfully reproduced, the question of evaluating reproducibility arises. For this purpose, two reproducibility targets are defined.

As common in natural sciences, **qualitative reproducibility** refers to checking whether experimental results support the hypotheses of the initial paper and come to the same conclusions. Coming rather from an engineering perspective, **quantitative reproducibility** aims at obtaining the same or close results as the initial experiment. In Fig. 5.1, they are respectively noted as > and =. In the context of the experiments with KENN, qualitative reproducibility refers to checking if the confirmed hypotheses in the initial paper are still supported by the reimplementation.

Regarding qualitative reproducibility, we focus on the hypotheses $H1$ and $H2$. The verification of $H3$ would involve reproducing the results of the baselines SBR and RNM which is beyond the scope of the work in this chapter. To test $H1$ and $H2$, we adopt the procedures used in the initial paper. For $H1$, a one-sided independent Student t-test with significance threshold 0.01 is employed to assess the superiority of the mean accuracy of KENN over the base neural network. For $H2$, which establishes a relation between the training set size and the delta of base neural network and KENN, no precise evaluation procedure is mentioned. However, the absolute difference between the mean test accuracies for the experiments is observed.

In terms of quantitative reproducibility, absolute equality of the results is hard to be expected, since even reexecuting the initial implementation does not lead to absolutely equal numbers. This may be due to differences in software or hardware or the absence of fixed seeds [33, 162, 166]. Therefore, we evaluate the steps of replication and reevaluation by examining the distributions of the reported test results and in particular their similarity. In order to test their equality, we compute the two-sided Kolmogorov-Smirnov goodness-of-fit test (KS-test) [143], which checks whether two samples are drawn from the same distribution. If the p-value of the test is below the significance threshold, there is evidence that the results of the experiments are not drawn from the same distribution. It has to be mentioned that the KS-test can only provide significant results in measuring inequality.

[4] `https://gitlab.inria.fr/tyrex-public/reproducibility-aaai24`

	Reported results			Initial Implementation		
train	NN	KENN	Delta	NN	KENN	Delta
10%	0.544	0.652	0.108	0.540 (0.067)	0.651 (0.017)	0.110 (0.067)
25%	0.629	0.702	0.073	0.630 (0.018)	0.702 (0.012)	0.072 (0.012)
50%	0.680	0.744	0.065	0.681 (0.187)	0.745 (0.012)	0.064 (0.021)
75%	0.733	0.788	0.055	0.733 (0.025)	0.791 (0.016)	0.058 (0.026)
90%	0.759	0.808	0.049	0.758 (0.027)	0.807 (0.022)	0.049 (0.028)

Table 5.2.: Reproduction Results

5.5. Reproduction

As a first step, the initial experiments with KENN are reproduced. Therefore, (*a*) the executable, (*b*) the dataset, (*c*) the instructions and environment information to run the executable and (*d*) the procedure to collect and interpret the results are required. Given these elements, it is possible to process the instructions to reproduce the experiment.

5.5.1. Pitfalls and Workarounds

The paper [47] provides the KENN2[2] package. It contains the implementation of the knowledge enhancement layers in Tensorflow but not the experiments. Through an exchange with the authors, we gained access to the source code and data of the experiments which were made publicly available[1]. They also pointed to another paper [46] which further documents the experiments. The hyperparameters in the public implementation correspond to the results reported in [46] but not to the results found in [47].

The repository provides (*a*) a reference to the KENN2 package, (*b*) a link to the dataset, (*c*) a README file with instructions to run the code, and (*d*) a Jupyter notebook to analyse the results. The initial experiments use a dataset that is included in the repository instead of referring to an external, widely available version of the dataset. A link to the data source was provided, but it was inaccessible. Neither the origin of the data nor any filtering or pre-processing steps were documented. The instructions to run the experiments are reasonably complete, including a description of the required Python modules as a requirements file and the full command line to be run. However, the requirements only contain a list of packages without their version number. This can be a critical concern as Python packages evolve frequently, sometimes compromising compatibility. The Python version is not defined either which can be crucial for the successful execution of the experiments. We inferred it from the description of the KENN2 package. In both papers [47, 46] experimental results are reported in numeric tables. Despite seeded random operations, we observe

subtle variations in the results. We note a non-deterministic $set()$ operation which may introduce variation. As a result, exact quantitative reproducibility cannot be guaranteed.

5.5.2. Results

After achieving a fully functional environment setup, we successfully run the experiments. The obtained results are reported in Table 5.2. In comparison to the results initially provided [46], we add information such as the standard deviation of the test accuracies over all runs and the p-values for a t-test.

Qualitative Reproducibility. For $H1$, KENN significantly outperforms the base neural network for all training set sizes ($p \ll 0.01$) which is consistent with the authors' claims. Regarding $H2$, we observe that the reported difference (column delta) in mean accuracy between KENN and the base neural network is larger for smaller training set sizes. Note that, since the initial experiments use paired samples, the means of the deltas can be reported, though we can only report the difference of the means.

Quantitative Reproducibility. The reported and reproduced results cannot be compared statistically, since the full sample of the reported results is not available.

5.5.3. Lessons Learned

L1: Provide a Precise Identification of the Software Environment, Source Code and Dataset. A first lesson learned is that the complete description of the software environment including version numbers of all modules should be provided. Furthermore, the datasets used should be described precisely, including information on their origin and on any applied pre-processing. Both are critical information to reproduction.

L2: Automate the Steps for Reproduction. A second lesson learned is that the reproduction should be made as automatic as possible by the initial authors. Such an automation would even be useful for the initial authors to be able to repeat their experiments routinely.

L3: Provide Contact Information and Be Reachable. The reachability and assistance of the authors was essential to clarify open questions and to get access to key components for reproducibility such as the experiment source code.

L4: Distinguish Different Experiments. A unique reference to the experiments was missing. In fact, the authors published a first version of a paper [48] and then several others [46, 47] which contain improvements and additional experimental settings. Still, all papers point to the same repository. This complicates the reproduction of the experiments of each paper. It would be a good practice to specify for each paper the version of the repository or to use another repository.

Parameter	Value	Paper	Code	Defaults
NN - Number of Hidden Layers	3	✓	✓	
NN - Number of Hidden Neurons	50	✓	✓	
NN - Hidden Layer Activation	ReLu	✓	✓	
NN - Output Layer Activation	Linear	✓	✓	
KENN - Clause Weight Initialization	Constant, 0.5	✓	✓	
KENN - Binary Preactivations	500	✓	✓	
KENN - Number of KE Layers	3		✓	
KENN - Range Constraint	[0.0, 500.0]		✓	
Epochs	300		✓	
Batch Size	Full-batch		✓	
Loss function	Categorical Cross-entropy		✓	
Early Stopping - Patience	10		✓	
Early Stopping - Min Delta	0.001		✓	
Dropout Rate	0.0, no dropout		✓	
Learning Rate	0.001			✓
NN - Weight Initialization	Random, Glorot uniform			✓
NN - Bias Initialization	Constant, Zeroes			✓
Optimizer	Adam [113], $\beta_1 = 0.9, \beta_2 = 0.99, \epsilon = 10^{-7}$			✓

Table 5.3.: The set of hyperparameters used in the initial experiments and how they were recovered.

5.6. Replication

In this section, the reimplementation in PyTorch and the replication of the experiments is detailed. Again, the obtained results are compared to the results of the published and reproduced experiments. To reimplement the system, we first identify the main components of the method by examining, on the one hand, the concepts of KENN introduced in the initial paper. On the other hand, we examine the code of the initial implementation to recover necessary information that is underspecified or not explicitly mentioned in the paper. As main components we identify (*a*) the data pre-processing, (*b*) the model definition, (*c*) the training loop, and (*d*) the hyperparameter definition. We reimplement these components as follows.

Data pre-processing. We use the same Citeseer dataset and prior knowledge as provided for the initial experiments. We preserve the data splitting procedure, as well as the rebalancing of the prediction classes.

Model Definition. The KENN model consists of two stacked components: The base neural network and the knowledge enhancement layers. In the initial implementation, the base neural network is implemented with Keras [38] and the knowledge enhancement layers are imported from the KENN2[2] package written in Tensorflow. We reimplement both the base neural network and use the knowledge enhancement layers from the KENN2 package[3].

Training Loop. In the training loop, we replace the optimizer and the loss function implemented in Tensorflow by their PyTorch equivalent.

Hyperparameter Definition. We identify the hyperparameters in the experiments and their values. To keep track of hyperparameters in a clean manner, we connect the reimplementation to an experiment tracking tool [22].

5.6.1. Pitfalls and Workarounds

At first sight, the reimplementation in PyTorch seems straightforward. However, we struggle in identifying the hyperparameters used in the initial implementation. The relevant information has to be gathered from various sources. The set of revealed hyperparameters, their assignments and how we recovered them is summarized in Table 5.3.

A first subset of hyperparameter values is explicitly named in the initial paper including the architecture of the base neural network, the initialization of the clause weights and the binary preactivation value. An additional subset of parameters is defined in a modifiable script in the repository of the initial implementation. This script contains information on early stopping (and related parameters), the number of epochs, as well as the size of the validation set. Additional parameter values are required that are not mentioned in the paper, nor in the code documentation. By reviewing the source code, we recover the number of knowledge enhancement layers and the batch size, for example. An additional subset of hyperparameters is implicitly introduced and defined by framework-specific functions and their default assignments. These default values are found in the Tensorflow software documentation. From there, we recover the weight initialization of the dense layers in the base neural network and the optimizer-specific parameters.

Determining the values of some additional parameters turns out to be even more challenging. To get useful insights, we examine the isolated behavior of each component to ensure that they produce the same output, given some input. While randomness is essential to training neural networks, it complicates the analysis of the algorithm behavior. For the inspection of some components, we temporarily simplify operations and replace random numbers by fixed values. This allows to identify that linear layers are by default differently initialized in Tensorflow and in PyTorch. In Tensorflow, they are initialized randomly following a Glorot uniform distribution and the bias is initialized with zeroes. However, in PyTorch the weights and biases are randomly initialized following the uniform distribution.

5.6.2. Results

After recovering the full set of required parameter values and completing the reimplementation of KENN, we now assess the replicability of the experiments. The results of the replication are summarized in Table 5.4.

	Reimplementation			
train	NN	KENN	Delta means	p-values KS Test
10%	0.550 (0.042)	0.629 (0.076)	0.079	0.001
25%	0.632 (0.016)	0.676 (0.088)	0.044	0.024
50%	0.681 (0.014)	0.741 (0.028)	0.060	0.583
75%	0.729 (0.022)	0.785 (0.039)	0.056	0.054
90%	0.748 (0.026)	0.806 (0.020)	0.058	0.702

Table 5.4.: Replication results

Qualitative Reproducibility. We first check whether $H1$ and $H2$ hold in the replicated experiment. We observe that the p-values of the t-test comparing the test accuracies for all training set sizes fall below the significance threshold in favor of $H1$ which is thus supported by the replication. For $H2$, considering the deltas of the mean test accuracies across the training set sizes, no clear relationship between training set size and its effect on the knowledge enhancement is apparent. This may be due to a reproducibility failure or to the lack of an explicit procedure to test $H2$ in the initial work. In conclusion, we are able to confirm $H1$, but we are not able to replicate the results concerning $H2$.

Quantitative Reproducibility. The average accuracies of both implementations are relatively close (the highest observed difference is 0.02). However, their distributions are not identical. The similarity of the test accuracy distributions is computed against the test accuracy of the reproduced results, since we do not have the initial results from the authors of KENN. The p-values of the KS-test are listed in the right column. For the 10% training size, a significant difference ($p < 0.01$) between the distributions of the reproduced and replicated results is detected. For the remaining training set sizes, no difference appears significant. Hence, we consider our results as reasonably similar to those of the initial implementation, at least for large sample sizes. The accuracy obtained with 10% training set size appears less variable than the one on larger training sets.

5.6.3. Lessons Learned

L5: Document Hyperparameters Exhaustively. The identification of all hyperparameters together with their assignments is critical for replication as they can considerably affect the results and thus the conclusions drawn from them. While some hyperparameters such as learning rate and batch size are standard in deep learning methods, custom models often define their own hyperparameters, such as KENN's clause weights. Since they affect different components, hyperparameters are often declared at different stages in the experiment, which complicates their identification. In particular, some hyperparameters are implicitly defined as default parameters in the used library and are easily overlooked. A complete configuration file with the exhaustive list of hyperparameters, their description, and their value is key to replicability.

L6: Provide Clear Procedures to Check Claims. To evaluate qualitative reproducibility, we rely on the authors' procedures for verifying their hypotheses. With respect to $H2$, the lack of a precise procedure complicates the evaluation of the replicability. Clear procedures for the verification of claims are essential to ensure their replicability.

L7: Define Standards to Evaluate Reproducibility. In order to evaluate quantitative reproducibility, we use the KS-test that can only give statistical evidence that two distributions are not equal. Other metrics are proposed in [163]. To the best of our knowledge, no community standard on the evaluation of quantitative reproducibility exists. A community-agreed standard is needed to determine when two distributions of results are considered equivalent.

5.7. Reevaluation

Having reproduced and replicated the experiments with KENN, we now reevaluate the method on two more datasets Cora [144] and Pubmed [218]. The goal of this reevaluation is first, to check if the implementations behave robustly in the same way on different datasets and second, to evaluate whether the hypotheses for KENN are valid on other datasets. These datasets are also citation graphs. While Cora and Citeseer have a similar size, Pubmed is significantly larger. Similarly to the experiments on Citeseer, the prior knowledge encodes a relation of paper category and citations. To avoid variations in the results due to hyperparameters, we use the set identified in Table 5.3 for all datasets. In general, hyperparameters should be determined separately for each dataset in order to obtain models with the best possible prediction quality. However, we are mainly concerned with obtaining the same results instead of the best possible results.

5.7.1. Results

The results for Cora and Pubmed are shown in Table 5.5 and 5.6. When applying KENN to Pubmed, we encounter a performance issue with the data pre-processing in the initial implementation. We modify the pre-processing to improve its scalability while maintaining its functionality.

Qualitative Reproducibility. $H1$ is supported in both implementations for Cora ($p \ll 0.01$). Concerning $H2$, the deltas in the initial implementation indicate a relationship between decreasing training set size and knowledge enhancement. However, this difference is rather uncertain for the reimplementation. For $H1$ on PubMed, the p-values of the t-tests are not below the significance threshold across all training set sizes, for the initial implementation only for training set sizes 10% and 25% and for the reimplementation for training set sizes 10%, 50%, 75% and 90%. Hence, no significant performance is gained with KENN. Furthermore, no monotonically increasing benefit of knowledge enhancement

	Initial Implementation			**Reimplementation**			
train	NN	KENN	Delta	NN	KENN	Delta means	p-values KS-Test
10%	0.530 (0.029)	0.750 (0.017)	0.220 (0.030)	0.576 (0.016)	0.766 (0.010)	0.190	$8.9\ 10^{-4}$
25%	0.606 (0.018)	0.800 (0.012)	0.193 (0.016)	0.647 (0.009)	0.819 (0.012)	0.173	$8.4\ 10^{-10}$
50%	0.652 (0.013)	0.833 (0.009)	0.187 (0.017)	0.678 (0.009)	0.831 (0.013)	0.152	$5.9\ 10^{-1}$
75%	0.691 (0.016)	0.850 (0.014)	0.159 (0.018)	0.686 (0.013)	0.833 (0.015)	0.147	$2.9\ 10^{-4}$
90%	0.715 (0.027)	0.871 (0.016)	0.156 (0.028)	0.743 (0.012)	0.913 (0.017)	0.170	$9.2\ 10^{-11}$

Table 5.5.: Reevaluation results on the Cora dataset

	Initial Implementation			**Reimplementation**			
train	NN	KENN	Delta	NN	KENN	Delta means	p-values KS-Test
10%	0.333 (0.096)	0.405 (0.002)	0.071 (0.096)	0.326 (0.098)	0.404 (0.003)	0.077	$3.9\ 10^{-1}$
25%	0.341 (0.108)	0.416 (0.002)	0.075 (0.108)	0.380 (0.080)	0.414 (0.004)	0.034	$7.1\ 10^{-2}$
50%	0.409 (0.095)	0.443 (0.004)	0.033 (0.096)	0.364 (0.133)	0.441 (0.005)	0.077	$1.3\ 10^{-1}$
75%	0.447 (0.143)	0.498 (0.004)	0.051 (0.143)	0.414 (0.176)	0.495 (0.007)	0.081	$1.3\ 10^{-1}$
90%	0.505 (0.011)	0.510 (0.008)	0.004 (0.011)	0.504 (0.015)	0.504 (0.015)	-0.0001	$5.9\ 10^{-1}$

Table 5.6.: Reevaluation results on the Pubmed dataset

for smaller training set sizes is observed with both implementations, therefore *H2* is not supported by Pubmed.

Quantitative Reproducibility. Comparing the distributions of the test accuracies in both experiments, the p-values of the KS-tests are below the significance threshold and thus suggest rejection for all training dimensions except 50%. Therefore, we have statistical evidence for inequality and thus have not reached qualitative reproducibility on Cora. However, for Pubmed, the behavior in both implementations is aligned. The KS-test suggests no significant difference between both implementations. In conclusion, quantitative reproducibility is considered achieved on Pubmed.

Reproduction	**L1** Provide a precise identification of the software environment, source code and dataset **L2** Automate the steps for reproduction **L3** Provide contact information and be reachable **L4** Distinguish different experiments
Replication	**L5** Document hyperparameters exhaustively **L6** Provide clear procedures to check claims **L7** Define standards to evaluate reproducibility
Reevaluation	**L8** Conduct hyperparameter search **L9** Verify results on other datasets

Table 5.7.: Summary of the lessons learned

5.7.2. Lessons Learned

L8: Conduct Hyperparameter Search. As expected, with the hyperparameter values considered earlier KENN does not achieve competitive results on Cora and Pubmed in comparison to the state-of-the-art [160]. Hyperparameter tuning significantly affects the results of the experiment. With an appropriate set of hyperparameters for Cora and Pubmed, better results with KENN may be obtained.

L9: Verify Results on Other Datasets. Overall, the reevaluation experiments show that KENN improves the accuracy on the Cora dataset, but not on the Pubmed dataset. Furthermore, by reevaluation on PubMed, we can detect performance issues and address them, which makes the code applicable to larger datasets in general. As a lesson learned, it is derived that the application of a model to various datasets can either strengthen results by showing robustness or reveal weaknesses of methods that occur on specific datasets.

5.8. Conclusion and Outlook

In this chapter, we first progressively reproduced, replicated and reevaluated the experiments with KENN before extending it. In terms of qualitative reproducibility, in most experiments the hypothesis that KENN outperforms the base neural network ($H1$) is supported. The relationship between training set size and accuracy ($H2$) is observed for the reproduced experiments but less clear for the replicated and reevaluated experiments. We summarize all the lessons learned in Table 5.7.

In a broader context, the progressive approach in this work can be incorporated in a general workflow of extending a related method in machine learning while considering reproducibility steps, as illustrated in Fig. 5.2. While it is tempting to directly jump from a task to the development of a new method, the inclusion of the reproduce, replicate and reevaluate steps can increase the trust in experimental results. To the three steps conducted in this work, *record* and *repeat* can be added to better integrate the new (extended) method in the state-of-the-art (SOTA). Recording provides a sufficiently detailed documentation to make the new method accessible and reproducible for the community. Repeating the

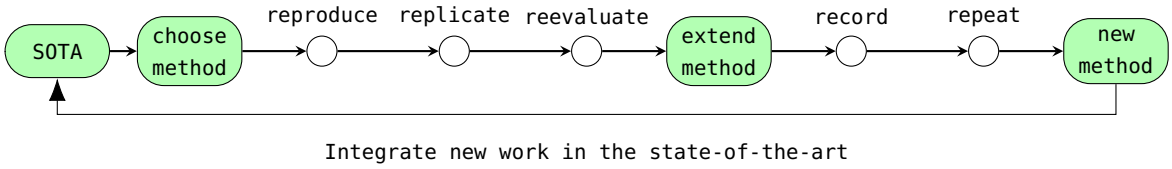


Figure 5.2.: The general pipeline of extending machine learning methods with the reproducibility steps.

experiment can serve as a self-check of documentation and/or automation. Fig. 5.2 displays an ideal situation in which the path to extension is flawless. In reality, each step may fail and lead the design back to the previous action. However, it helps to isolate the cause of failure at the earliest possible stage.

Further, we note that KENN predominantly satisfies common reproducibility guidelines. However, we encounter difficulties in reproducing, replicating and reevaluating this method. This shows that even though guidelines impose an additional burden for developers, easily and clearly reproducible work reduces the effort required for extending previous work and therefore accelerates advancements. In this sense, reproducibility guidelines should be continuously evaluated and modified (if needed) to ensure that they effectively serve their purpose. From our point of view in the context of this work, reproducibility checklists could improve by integrating some of our lessons learned.

Moreover, common reproducibility guidelines are mainly oriented towards reproduction, which ensures that published results correspond to those obtained from the code. Given strong automation, this may only require to clone and run. Even if replication and reevaluation outweigh the effort of reproduction, in return, they provide more insight into the details of an implementation or a method. Moreover, reimplementations make methods more available to the community. In this sense, efforts towards replication and reevaluation should continue to be encouraged, e.g. through reproducibility tracks or challenges [154, 97].

In addition to the general conclusions, this chapter has provided some insights with relevants to the further course of this work. In particular, the obtained results were verified with the stepwise approach and the reimplementation was tested for reliability. In addition, the portability of the results to other data sets was examined. This study has increased the understanding of the method and provides a good basis for adding extensions based on the reimplementation.

The lessons learned from this chapter are applied in particular to the following experiments of this thesis in order to make them reproducible. Thus, we make the code of all experiments publicly available, list the parameters used and the software requirements, provide a hash and conduct significance tests to evaluate hypotheses.

6. KeGNN: Knowledge Enhancement of Graph Neural Networks

The previous chapter focused on the reproducibility of the experiments with KENN [46], where the knowledge enhancement layers are stacked onto a simple multi-layer perceptron (MLP). However, an MLP is not powerful enough to incorporate graph structure into the representations. Thus, relational information can only be introduced by binary predicates in the clauses at the knowledge enhancement part.

In this chapter, the neuro-symbolic approach *Knowledge enhanced Graph Neural Networks (KeGNN)* is presented to conduct node classification given graph data and a set of prior knowledge in form of first-order logic clauses. In KeGNN, knowledge enhancement layers [46] are stacked on top of a *GNN* and adjust its predictions in order to increase the satisfaction of some prior knowledge. In addition to the parameters of the GNN, the knowledge enhancement layers contain learnable clause weights that reflect the impact of the prior knowledge on the predictions. Both components form an end-to-end differentiable model. KeGNN can be seen as an extension to KENN [46] that was successfully applied to semantic point cloud segmentation, image segmentation and multi-label classification [46, 45, 73]. In addition, KeGNN integrates a graph neural network as more expressive basis. Here, KeGNN is instantiated in conjunction with two well-known GNNs: Graph Attention Networks [196] and Graph Convolutional Networks [114]. KeGNN is applied to the benchmark datasets for node classification Cora, Citeseer, PubMed [218] and Flickr [221].

The work of this chapter is published [207].

> L. Werner, N. Layaïda, P. Genevès and S. Chlyah, "Knowledge Enhanced Graph Neural Networks," 2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA), Thessaloniki, Greece, 2023, pp. 1-10, doi: 10.1109/DSAA60987.2023.10302495.

It has been further presented at the 1st International Workshop on Knowledge-Based Compositional Generalization (KBCG) held in conjunction with IJCAI 2023[1].

[1] https://knowledgeai.github.io/

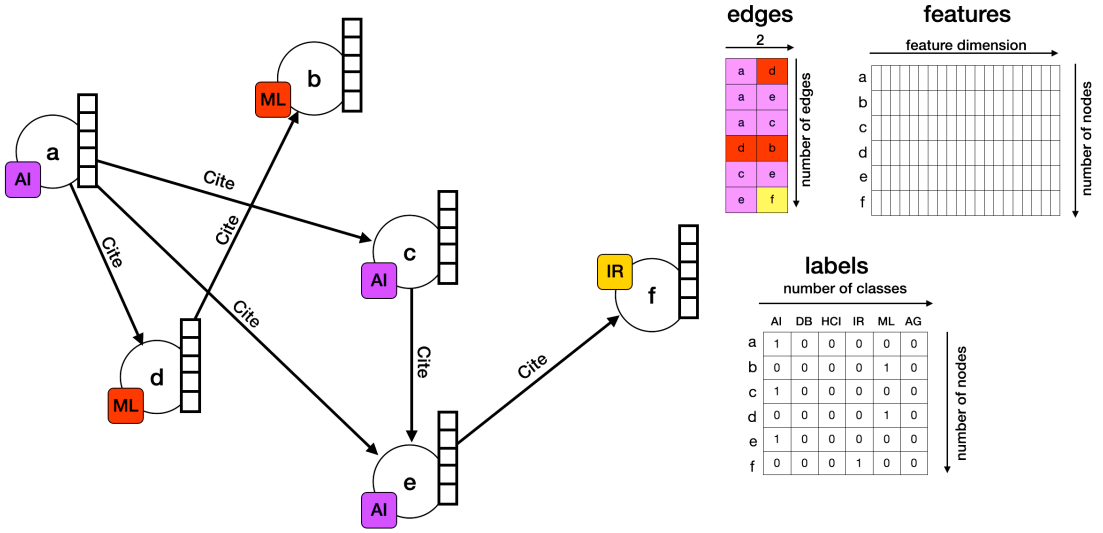


Figure 6.1.: An example extract of the Citeseer citation graph. Nodes represent scientific publications and labels represent their topics. Edges represent citations between them.

6.1. Method

KeGNN is a neuro-symbolic approach that can be applied to node classification tasks with the capacity of handling graph structure at the base neural network level. More precisely, at its core, KeGNN consists of two components that can both handle graph data and together form an end-to-end differentiable model. As base component, a graph neural network produces pre-activations based on node-level and edge-level numerical information. Knowledge enhancement layers refine these predictions guided by a set of prior knowledge with the goal of increasing the satisfaction of defined knowledge. The model takes two types of input: (1) real-valued graph data and (2) prior knowledge expressed in first-order logic.

6.1.1. Graph-structured Data

First, KeGNN takes as input an *node-attributed, labelled* graph $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{X}, \mathbf{Y})$ that consists of sets of n nodes $\mathbf{N}$, m edges $\mathbf{E}$, node features $\mathbf{X} \in \mathbb{R}^{n \times d}$ of dimension d and node labels $\mathbf{Y} \in \{0,1\}^{n \times c}$ with c classes. The label vector $\mathbf{y}$ contains one-hot encoded truth labels for c classes. In vector notation, the features and labels of a node v are described as $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in \{0,1\}^c$. The edges of the graph can be homogeneous or heterogeneous with multiple edge types.

Example 6.1.1 (Groundings of Unary and Binary Predicates). Let us define a *citation graph* $\mathbf{G}_{\text{Cit}}$ that consists of scientific publications and citations between them. Figure 6.1 shows an extract of a citation graph that is used as example to guide through this chapter. The publications are represented by a set of nodes $\mathbf{V}_{\text{Cit}}$ and citations by a set of edges $\mathbf{E}_{\text{Cit}}$. Publications are attributed with features $\mathbf{X}_{\text{Cit}}$ that describe their content as vectors [148]. Each node is labelled with one of the six topic categories {AI, DB, HCI, IR, ML, AG} that are encoded in $\mathbf{Y}_{\text{Cit}}$. The classes are abbreviations for the categories *Artificial Intelligence, Databases, Human-Computer Interaction, Information Retrieval, Machine Learning and Agents*. Since all nodes (publications) and edges (citations) have the same type, $\mathbf{G}_{\text{Cit}}$ is a homogeneous graph.

6.1.2. Prior Knowledge

A finite set of ℓ prior knowledge formulae $\mathcal{K}$ is provided to KeGNN. The knowlege is expressed in the logical language $\mathcal{L}$ that is defined over sets of constants C, variables $\mathcal{X}$ and predicates $\mathcal{P}$. Predicates have an arity r of one (unary) or two (binary): $\mathcal{P} = \mathcal{P}_U \cup \mathcal{P}_B$. Predicates of arity $r > 2$ are not considered in this work. Unary predicates describe nodes, and binary predicates describe relations. $\mathcal{L}$ supports negation ($\neg$) and disjunction ($\vee$). Each formula $\phi \in \mathcal{K}$ is defined as a clause $\bigvee_{j=1}^{q} o_j$ with q atoms $o_1 \vee \ldots \vee o_q$. Since the prior knowledge is general, all clauses are assumed to be universally quantified. A *grounded clause* is denoted as $\phi[x_1, x_2, ...|c_1, c_2, ...]$ with variables $x_i \in \mathcal{X}$ and constants $c_i \in C$. The set of all grounded clauses in a graph is $\mathcal{G}(\mathcal{K}, C)$.

Example 6.1.2 (Prior Knowledge on the Citation Graph). The graph $\mathbf{G}_{\text{Cit}}$ in Figure 6.1 is described with $\mathcal{L}$. Nodes are represented by a set of constants $C = \{a, b, \ldots, f\}$. The node labels are expressed as a set of unary predicates $\mathcal{P}_U = \{\text{AI}, \text{DB}, \ldots, \text{AG}\}$ and edges as a set of binary predicates $\mathcal{P}_B = \{\text{Cite}\}$. $\mathcal{L}$ has a set of variables $\mathcal{X} = \{x, y\}$. The atom AI(x), for example, expresses the membership of x to the class AI and Cite(x, y) denotes the existence of a citation between x and y. Some prior knowledge $\mathcal{K}$ is written as a set of $\ell = 6$ clauses in $\mathcal{L}$. Here, the assumption is denoted that two publications citing each other are member of the same class:

$$\begin{aligned}
&\phi_{\text{AI}} : \forall xy \neg\text{AI(x)} \vee \neg\text{Cite(x, y)} \vee \text{AI(y)} \\
&\phi_{\text{DB}} : \forall xy \neg\text{DB(x)} \vee \neg\text{Cite(x, y)} \vee \text{DB(y)} \\
&\ldots \\
&\phi_{\text{AG}} : \forall xy \neg\text{AG(x)} \vee \neg\text{Cite(x, y)} \vee \text{AG(y)}
\end{aligned}$$

The atoms are grounded by replacing the variables x and y with the constants $\{a, b, \ldots f\}$ to obtain the sets of unary groundings $\{\text{AI(a)}, \text{ML(b)}, \ldots, \text{IR(f)}\}$ and binary groundings $\{\text{Cite(a, d)}, \text{Cite(a, e)}, \ldots, \text{Cite(a, f)}\}$. Assuming a closed world and exclusive classes, negative facts are derived, such as $\{\neg\text{DB(a)}, \neg\text{IR(a)}, \ldots, \neg\text{Cite(a, b)}\}$.

Note that the knowledge encoded here is considered as *soft knowledge* that does not need to fully be satisfied in a logical sense and encodes assumptions that may be violated by

some instances. For example, there may be two publications that cite each other belong to different classes. The goal is not to fully satisfy the rules but rather refine predictions in order to be more consistent with the knowledge. Several ways to encode prior knowledge exist. Knowledge can be *manually handcrafted* from common sense assumptions, as in Example 6.1.2. Furthermore, knowledge might be automatically extracted with rule mining algorithms, as described in Section 2.3.

6.1.3. Fuzzy Semantics

Let us consider an attributed and labelled graph $\mathbf{G}$ and the prior knowledge $\mathcal{K}$. While $\mathcal{K}$ is defined in the logical language $\mathcal{L}$, the neural component in KeGNN relies on continuous and differentiable representations. To interpret Boolean logic in the real-valued domain, KeGNN uses fuzzy logic [220], which maps Boolean truth values to the continuous interval $[0, 1] \subset \mathbb{R}$. A constant in C is interpreted as a real-valued feature vector $\mathbf{h} \in \mathbb{R}^d$. A predicate $P \in \mathcal{P}$ with arity r is interpreted as a function

$$f_P : \mathbb{R}^{r \times d} \mapsto [0, 1] \tag{6.1}$$

that takes r feature vectors as input and returns a truth value.

Example 6.1.3 (Binary Predicates in KeGNN on the Citation Graph). In the example, a unary predicate $P_U \in \mathcal{P}_U = \{\text{AI}, \text{DB}, \ldots\}$ is interpreted as a function $f_{P_U} : \mathbb{R}^d \mapsto [0, 1]$ that takes a feature vector $\mathbf{h}$ and returns a truth value indicating whether the node belongs to the class encoded as P_U. The binary predicate Cite $\in \mathcal{P}_B$ is interpreted as the function

$$f_{\text{Cite}}(v, u) = \begin{cases} 1, & \text{if } (v, u) \in \mathbf{E}_{\text{Cit}} \\ 0, & \text{else.} \end{cases} \tag{6.2}$$

f_{Cite} returns the truth value one if there is an edge between two nodes v and u in $\mathbf{G}_{\text{Cit}}$ and zero otherwise.

T-conorm functions $\perp : [0, 1] \times [0, 1] \mapsto [0, 1]$ [116] take real-valued truth values of two literals and define the truth value of their disjunction. The Gödel t-conorm function for two truth values $\mathbf{t}_i, \mathbf{t}_j$ is

$$\perp(\mathbf{t}_i, \mathbf{t}_j) \mapsto \max(\mathbf{t}_i, \mathbf{t}_j). \tag{6.3}$$

To obtain the truth value of a formula $\phi : o_1 \vee \ldots \vee o_q$, the function $\perp$ is extended to a vector $\mathbf{t}$ of q truth values: $\perp(\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_q) = \perp(\mathbf{t}_1, \perp(\mathbf{t}_2 \ldots \perp(\mathbf{t}_{q-1}, \mathbf{t}_q)))$. Fuzzy negation over truth values is defined as $\mathbf{t} \mapsto 1 - \mathbf{t}$ [220].

Example 6.1.4 (A Grounded Clause). Given the clause $\phi_{\text{AI}} : \forall xy \neg\text{AI(x)} \vee \neg\text{Cite(x, y)} \vee \text{AI(y)}$ and its grounding $\phi_{\text{AI}}[x, y|a, b] : \text{AI(a)} \vee \neg\text{Cite(a, b)} \vee \text{AI(b)}$ to the constants a and b and truth values for the grounded predicates $\text{AI(a)} = \mathbf{t}_1$, $\text{AI(b)} = \mathbf{t}_2$ and $\text{Cite(a, b)} = \mathbf{t}_3$, the truth value of $\phi_{\text{AI}}[x, y|a, b]$ is $\max\{\max\{(1 - \mathbf{t}_1), (1 - \mathbf{t}_3)\}, \mathbf{t}_2\}$.

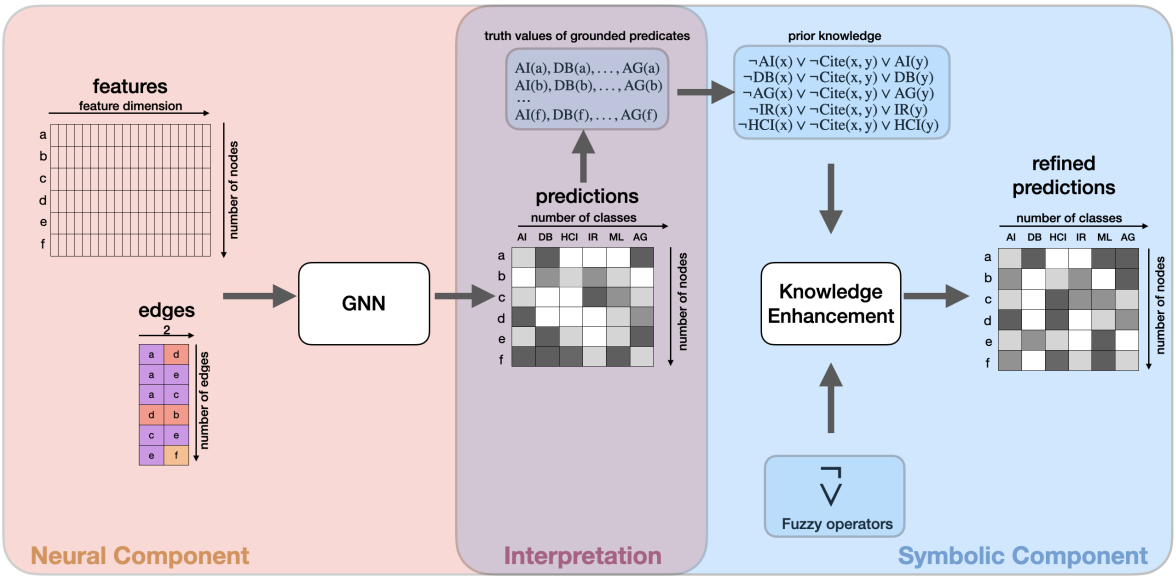


Figure 6.2.: Overview of the KeGNN architecture. A GNN outputs predictions which are interpreted as truth values for the unary predicates in the logical language and fed to the knowledge enhancement layers. The knowledge enhancement layers return refinements for the predictions.

6.1.4. Model Architecture

The way KeGNN computes the final predictions is divided in two stages. First, a GNN predicts the node classes given the features and the edges. Second, the knowledge enhancement layers use the predictions as truth values for the grounded unary predicates and update them with respect to the knowledge. An overview of KeGNN is illustrated in Figure 6.2.

6.1.4.1. Graph Neural Network Component

The role of the GNN is to exploit feature information in the graph structure. The key strength of a GNN is to enrich node representations with graph structure by nesting k message passing layers, as introduced in Section 3.2. Per layer, the representations of neighbouring nodes are aggregated and combined to obtain updated representations. The node representation $\mathbf{h}_v^{k+1}$ in the k-th message passing layer is

$$\mathbf{h}_v^{k+1} = \texttt{combine}\big(\mathbf{h}_v^k, \texttt{aggregate}\big(m_{v,u} | u \in \mathbf{N}_1(v)\big)\big). \tag{6.4}$$

The layers contain learnable parameters that are optimized with backpropagation. In this chapter, we consider two well-known GNNs as components for KeGNN: Graph Convolutional Networks (GCN) [114] and Graph Attention Networks (GAT) [196], see Section 3.2.1 and Section 3.2.2. While GCN considers the graph structure as known, GAT allows for assessing the importance of the neighbours with attention weights α_{vu}

between node v and node u. In case of multi-head attention, the attention weights are calculated multiple times and concatenated which allows for capturing different aspects of the input data. In KeGNN, the GNN implements the functions f_{P_U} for unary predicates, see Equation 6.1 in Section 6.1.3). In other words, the predictions of the GNN are used as truth values for the grounded unary predicates in the symbolic component

$$f_{P_U} := f_{\text{GCN}}(\mathbf{h}_0, \mathbf{E}, \theta_{\text{GCN}}).$$

In contrast KENN [46] does not consider graph structure at base neural network level:

$$f_{P_U} := f_{\text{MLP}}(\mathbf{h}_0, \theta_{\text{MLP}}).$$

6.1.4.2. Symbolic Component

To refine the predictions of the GNN, one or more knowledge enhancement layers are stacked onto the GNN to update its predictions $\mathbf{Y}$ to $\mathbf{Y}'$. The goal is to increase the satisfaction of the prior knowledge. The predictions $\mathbf{Y}$ of the GNN serve as input to the symbolic component where they are interpreted as fuzzy truth values for the unary grounded predicates $\mathbf{U} := \mathbf{Y}$ with $\mathbf{U} \in \mathbb{R}^{n \times c}$. Fuzzy truth values for the groundings of binary predicates are encoded as a matrix $\mathbf{B}$ where each row represents an edge (v, u) and each column represents an edge type e. In the context of node classification, the GNN returns only predictions for the node classes, while the edges are assumed to be given. A binary grounded predicate is therefore set to truth value 1 (true) if an edge between two nodes v and u exists:

$$\mathbf{B}_{[(v,u),e]} = \begin{cases} 1, & \text{if } (v,u) \text{ of type } e \in \mathbf{E} \\ 0, & \text{else.} \end{cases} \tag{6.5}$$

Example 6.1.5 (Unary and Binary Groundings on the Citation Graph). In case of the beforementioned citation graph of Figure 6.1, the unary and binary groundings $\mathbf{U}$ and $\mathbf{B}$ are defined as:

$$\mathbf{U} := \begin{bmatrix} \text{AI(a)} & \dots & \text{AG(a)} \\ \text{AI(b)} & \dots & \text{AG(b)} \\ \vdots & & \vdots \\ \text{AI(f)} & \dots & \text{AG(f)} \end{bmatrix} \quad \mathbf{B} := \begin{bmatrix} \text{Cite(a, d)} \\ \text{Cite(a, e)} \\ \text{Cite(a, c)} \\ \vdots \\ \text{Cite(c, e)} \\ \text{Cite(e, f)} \end{bmatrix}.$$

Since we only have one binary predicate Cite in this example, $\mathbf{B}$ has only one column.

To enhance the satisfaction of clauses that contain both unary and binary predicates, their groundings are joined into one matrix $\mathbf{M} \in \mathbb{R}^{m \times k}$ with $k = 2 \cdot |\mathcal{P}_U| + |\mathcal{P}_B|$. $\mathbf{M}$ is computed by joining $\mathbf{U}$ and $\mathbf{B}$ so that each row of $\mathbf{M}$ represents an edge (v, u). As a result, $\mathbf{M}$ contains all required grounded unary predicates for the edges and nodes in the graph.

Example 6.1.6 (Joined Groundings on the Citation Graph). For the example citation graph, we obtain $\mathbf{M}$ as follows:

$$\mathbf{M} = \left[\begin{array}{cccc|cccc|c} \overbrace{\mathrm{AI}^{\mathrm{x}}(\mathrm{a}) & \mathrm{DB}^{\mathrm{x}}(\mathrm{a}) & \ldots & \mathrm{AG}^{\mathrm{x}}(\mathrm{a})}^{\text{groundings unary predicates for x}} & \overbrace{\mathrm{AI}^{\mathrm{y}}(\mathrm{d}) & \mathrm{DB}^{\mathrm{y}}(\mathrm{d}) & \ldots & \mathrm{AG}^{\mathrm{y}}(\mathrm{d})}^{\text{groundings unary predicates for y}} & \overbrace{\mathrm{Cit}(\mathrm{a},\mathrm{d})}^{\text{groundings Cite(x,y)}} \\ \mathrm{AI}^{\mathrm{x}}(\mathrm{a}) & \mathrm{DB}^{\mathrm{x}}(\mathrm{a}) & \ldots & \mathrm{AG}^{\mathrm{x}}(\mathrm{a}) & \mathrm{AI}^{\mathrm{y}}(\mathrm{e}) & \mathrm{DB}^{\mathrm{y}}(\mathrm{e}) & \ldots & \mathrm{AG}^{\mathrm{y}}(\mathrm{e}) & \mathrm{Cit}(\mathrm{a},\mathrm{e}) \\ \mathrm{AI}^{\mathrm{x}}(\mathrm{a}) & \mathrm{DB}^{\mathrm{x}}(\mathrm{a}) & \ldots & \mathrm{AG}^{\mathrm{x}}(\mathrm{a}) & \mathrm{AI}^{\mathrm{y}}(\mathrm{c}) & \mathrm{DB}^{\mathrm{y}}(\mathrm{c}) & \ldots & \mathrm{AG}^{\mathrm{y}}(\mathrm{c}) & \mathrm{Cit}(\mathrm{a},\mathrm{c}) \\ \vdots & & & \vdots & \vdots & & & \vdots & \vdots \\ \mathrm{AI}^{\mathrm{x}}(\mathrm{c}) & \mathrm{DB}^{\mathrm{x}}(\mathrm{c}) & \ldots & \mathrm{AG}^{\mathrm{x}}(\mathrm{c}) & \mathrm{AI}^{\mathrm{y}}(\mathrm{e}) & \mathrm{DB}^{\mathrm{y}}(\mathrm{e}) & \ldots & \mathrm{AG}^{\mathrm{y}}(\mathrm{e}) & \mathrm{Cit}(\mathrm{c},\mathrm{e}) \\ \mathrm{AI}^{\mathrm{x}}(\mathrm{e}) & \mathrm{DB}^{\mathrm{x}}(\mathrm{e}) & \ldots & \mathrm{AG}^{\mathrm{x}}(\mathrm{e}) & \mathrm{AI}^{\mathrm{y}}(\mathrm{f}) & \mathrm{DB}^{\mathrm{y}}(\mathrm{f}) & \ldots & \mathrm{AG}^{\mathrm{y}}(\mathrm{f}) & \mathrm{Cit}(\mathrm{e},\mathrm{f}) \end{array}\right]$$

As mentioned previously, a knowledge enhancement layer consists of multiple *clause enhancers*. A clause enhancer is instantiated for each clause $\phi_i \in \mathcal{K}$. Its aim is to compute refinements $\delta\mathbf{M}_{\phi_i}$ for the groundings in $\mathbf{M}$ that increase the satisfaction of ϕ_i.

First, fuzzy negation is applied to the columns of $\mathbf{M}$ that correspond to negated atoms in ϕ. Then, the refinements are computed by a *t-conorm boost function* δ_ϕ [48], as introduced in Section 4.2.3. In this case, $\mathbf{M}$ represents the preactivations $\mathbf{Z}$.

$$\Delta^{\phi}_{ij} = \delta^{\phi}(\mathbf{Z})_{ij} = w_\phi \cdot \mathrm{softmax}(\mathbf{Z})_i = w_\phi \cdot \frac{e^{\mathbf{Z}_{ij}}}{\sum_{l=1}^{q} e^{\mathbf{Z}_{il}}}.$$

The function $\delta_\phi : [0,1]^q \mapsto [0,1]^q$ takes q truth values and returns refinements to those truth values such that the satisfaction is increased: $\bot(\mathbf{t}) \leq \bot(\mathbf{t} + \delta(\mathbf{t}))$. The boost function δ_{w_ϕ} employs a clause weight w_ϕ that is initialized in the beginning of the training and optimized during training as a learnable parameter. The refinements for the groundings calculated by δ_{w_ϕ} are proportional to w_ϕ. Therefore, w_ϕ determines the magnitude of the update and thus reflects the impact of a clause. The refinements to the atoms that do not occur in a clause are set to zero. The boost function is applied row-wise to $\mathbf{M}$ as illustrated in the following example.

Example 6.1.7 (Refinements on the Citation Graph). Given the clause $\phi_{AI} : \forall xy \neg \mathrm{AI}(\mathrm{x}) \vee \neg\mathrm{Cite}(\mathrm{x},\mathrm{y}) \vee \mathrm{AI}(\mathrm{y})$ and the clause weight w_{AI}, the refinements for this clause are $\delta\mathbf{M}_{\phi_{AI}} =$

$$w_{\mathrm{AI}} \cdot \begin{bmatrix} \delta_{\neg\mathrm{AI}^{\mathrm{x}}(\mathrm{a})} & 0 & \ldots & \delta_{\mathrm{AI}^{\mathrm{y}}(\mathrm{c})} & 0 & \ldots & \delta_{\neg\mathrm{Cit}(\mathrm{a},\mathrm{c})} \\ \delta_{\neg\mathrm{AI}^{\mathrm{x}}(\mathrm{a})} & 0 & \ldots & \delta_{\mathrm{AI}^{\mathrm{y}}(\mathrm{e})} & 0 & \ldots & \delta_{\neg\mathrm{Cit}(\mathrm{e},\mathrm{a})} \\ \delta_{\neg\mathrm{AI}^{\mathrm{x}}(\mathrm{a})} & 0 & \ldots & \delta_{\mathrm{AI}^{\mathrm{y}}(\mathrm{d})} & 0 & \ldots & \delta_{\neg\mathrm{Cit}(\mathrm{c},\mathrm{d})} \\ \vdots & & & \vdots & \vdots & & \\ \delta_{\neg\mathrm{AI}^{\mathrm{x}}(\mathrm{e})} & 0 & \ldots & \delta_{\mathrm{AI}^{\mathrm{y}}(\mathrm{f})} & 0 & \ldots & \delta_{\neg\mathrm{Cit}(\mathrm{e},\mathrm{f})} \end{bmatrix} \tag{6.6}$$

The values of $\delta\mathbf{M}_{\phi_{AI}}$ are calculated as

$$\delta_{\neg\mathrm{AI}^{\mathrm{x}}(\mathrm{a})} = \phi_{w_{\mathrm{AI}}}(\mathbf{z})_a = -\frac{e^{-\mathbf{z}_{AI(a)}}}{e^{-\mathbf{z}_{AI(a)}} + e^{-\mathbf{z}_{Cit(a,c)}} + e^{\mathbf{z}_{AI(c)}}} \tag{6.7}$$

Name	#nodes	#edges	#features	#Classes	train/valid/test split
Citeseer	3,327	9,104	3,703	6	1817/500/1000
Cora	2,708	10,556	1,433	7	1208/500/1000
PubMed	19,717	88,648	500	3	18217/500/1000
Flickr	89,250	899,756	500	7	44624/22312/22312

Table 6.1.: Overview of the Citeseer, Cora, PubMed and Flickr datasets

Each clause enhancer computes refinements $\delta\mathbf{M}_\phi$ to increase the satisfaction of a clause independently. The refinements of all clause enhancers are finally added, resulting in a matrix $\delta\mathbf{M} = \sum_{\phi\in\mathcal{K}} \delta\mathbf{M}_\phi$. To apply the refinements to the initial predictions, $\delta\mathbf{M}$ has to be added to $\mathbf{Y}$. The refinements in $\delta\mathbf{M}$ can not directly be applied to the predictions $\mathbf{Y}$ of the GNN. Since the unary groundings $\mathbf{U}$ were joined with the binary groundings $\mathbf{B}$, multiple refinements may be proposed for the same grounded unary atom. For example, for the grounded atom AI(c) the refinements $\delta_{\neg\mathrm{AI}^y(c)}$ and $\delta_{\neg\mathrm{AI}^x(c)}$ are computed, since c occurs in the grounded clauses $\phi_{\mathrm{AI}}[x, y|a, c]$ and $\phi_{\mathrm{AI}}[x, y|c, e]$. In citation graph in Example 6.1.1 the node v_c occurs in first place of edge (v_a, v_c) and in second place of edge (v_c, v_e). Therefore, all refinements for the same grounded atom are grouped and summed, which reduces the size of $\mathbf{M}$ to the size of $\mathbf{U}$.

To ensure that the updated predictions remain truth values in the range of $[0, 1]$, the knowledge enhancement layer refines the preactivations $\mathbf{Z}$ of the GNN and to $\mathbf{Z}'$ and then applies the activation function σ to $\mathbf{Z}'$ in order to obtain the final predictions: $\mathbf{Y}' = \sigma(\mathbf{Z}')$. The refinements by the knowledge enhancer are added to the preactivations $\mathbf{Z}$ of the GNN and passed to σ to obtain the updated predictions

$$\mathbf{Y}' = \sigma\left(\mathbf{Z} + \sum_{\phi\in\mathcal{K}} \delta\mathbf{U}_\phi\right) \tag{6.8}$$

where $\delta\mathbf{U}_\phi$ is the matrix obtained by aggregating the refinements to the unary predicates from $\delta\mathbf{M}_\phi$. Regarding the binary groundings, the values in $\mathbf{B}$ are set to a high positive value that results in one when σ is applied.

6.2. Experimental Evaluation

To evaluate the performance of KeGNN, node classification experiments are conducted. In the following, KeGNN is called *KeGCN* and *KeGAT* when instantiated to a GCN or a GAT, respectively. As additional baseline, we consider *KeMLP*, that stacks knowledge enhancement layers onto an MLP, as proposed in [46]. Furthermore, the standalone neural models MLP, GCN and GAT are used as baselines.

6.2.1. Datasets

The models are tested on the datasets Citeseer, Cora, PubMed and Flickr that are common benchmarks for node classification. Citeseer, Cora and PubMed are citation graphs that encode citations between scientific papers as in Example 6.1.2. Flickr contains images that are represented by nodes and shared properties between them that are represented by edges. All datasets are modelled as homogeneous, labelled and attributed graphs as defined in Section 6.1.1. Table 6.1 gives an overview of the datasets used in this chapter. The datasets are publicly available on the dataset collection of PyTorch Geometric [61]. For the split into train, valid and test set, we take the pre-defined splits in [35] for the citation graphs and in [221] for Flickr. Word2Vec vectors [148] are used as node features for the citation graphs and image data for Flickr. The models are trained and evaluated in a transductive setting.

6.2.2. Prior Knowledge

The set of prior logic for the knowledge enhancement layers is manually defined. In this work, we encode the assumption that the existence of an edge for a node pair points to their membership to the same class and hence provides added value to the node classification task. In the context of citation graphs, this implies that two documents that cite each other refer to the same topic, while for Flickr, connected images share the same properties. Following this pattern for all datasets, a clause ϕ: $\forall xy : \neg\text{Class}_i(x) \vee \neg\text{Link}(x, y) \vee \text{Class}_i(y)$ is instantiated for each node class Class_i with $i \in \{1, \ldots, c\}$.

6.2.3. Implementation

The source code of the experiments are publicly available[2]. It is based on PyTorch [161] and the graph learning library PyTorch Geometric [61]. The Weights & Biases tracking tool [22] is used to monitor the experiments. More experimental details can be found in Appendix A.1.

6.2.4. Results

Performance. To compare the performance of all models, we examine the average test accuracy over 50 runs (10 for Flickr) for the knowledge enhanced models KeMLP, KeGCN, KeGAT and the standalone base models MLP, GCN, GAT on the listed datasets. The results are presented in Table 6.2. For Cora and Citeseer, KeMLP leads to a significant improvement over MLP (p-value of one-sided t-test $\ll 0.05$). In contrast, no significant advantage of KeGCN or KeGAT in comparison to the standalone base model is observed. Nevertheless, all GNN-based models are significantly superior to KeMLP for Cora. This includes not

[2] `https://gitlab.inria.fr/tyrex/kegnn`. The hash is `ddb05f37d390fd06181bac8275aac45962b74ff0`

	MLP	KeMLP	GCN	KeGCN	GAT	KeGAT
Cora	0.7098 (0.0080)	0.8072 (0.0193)	0.8538 (0.0057)	**0.8587** (0.0057)	0.8517 (0.0068)	0.8498 (0.0066)
CiteSeer	0.7278 (0.0081)	0.7529 (0.0067)	0.748 (0.0102)	0.7506 (0.0096)	0.7718 (0.0072)	**0.7734** (0.0073)
PubMed	0.8844 (0.0057)	**0.8931** (0.0048)	0.8855 (0.0062)	0.8840 (0.0087)	0.8769 (0.0040)	0.8686 (0.0081)
Flickr	0.4656 (0.0018)	0.4659 (0.0012)	**0.5007** (0.0063)	0.4974 (0.0180)	0.4970 (0.0124)	0.4920 (0.0189)

Table 6.2.: Average test accuracy of 50 runs (10 for Flickr). The standard deviations are reported in brackets and the highest value per dataset is marked in bold.

Model	Avg Epoch Time
MLP	0.02684
GCN	0.03109
GAT	0.06228
KeMLP	0.04304
KeGCN	0.03747
KeGAT	0.08384

Table 6.3.: Comparison of the average epoch times on the Citeseer dataset.

only KeGCN and KeGAT, but also the GNN baselines. For Citeseer, KeGAT and GAT both outperform KeMLP. In the case of PubMed, only a significant improvement of KeMLP over MLP are observed, while the GNN-based models and their enhanced versions do not provide any positive effect. For Flickr, no significant improvement between the base model and the respective knowledge enhanced model are observed. Nevertheless, all GNN-based models outperform KeMLP, reporting significantly higher mean test accuracies for KeGAT, GAT, GCN and KeGCN.

Runtime. The average runtimes per epoch on the Citeseer dataset are compared for all models in Tab 6.3. The runtimes were reported for models with three hidden layers and three knowledge enhancement layers in full-batch training. It is noted that the knowledge enhancement layers lead to increased runtimes compared to the base models since the overall model complexity is higher.

6.2.5. Exploitation of the Graph Structure

It turns out that the performance gap between MLP and KeMLP is larger than for KeGNN in comparison to the standalone GNN. To explain this observation, we examine how the graph structure affects the prediction performance. In Figure 6.3 we analyse the accuracy grouped by the node degree for the entire graph for MLP vs. KeMLP and GCN vs. KeGCN. The findings for KeGAT are in line with those for KeGCN. It is observed that KeMLP performs better compared to MLP as the node degree increases. By contrast, when comparing GCN and KeGCN, for both models, the accuracy is superior for nodes with a higher degree.

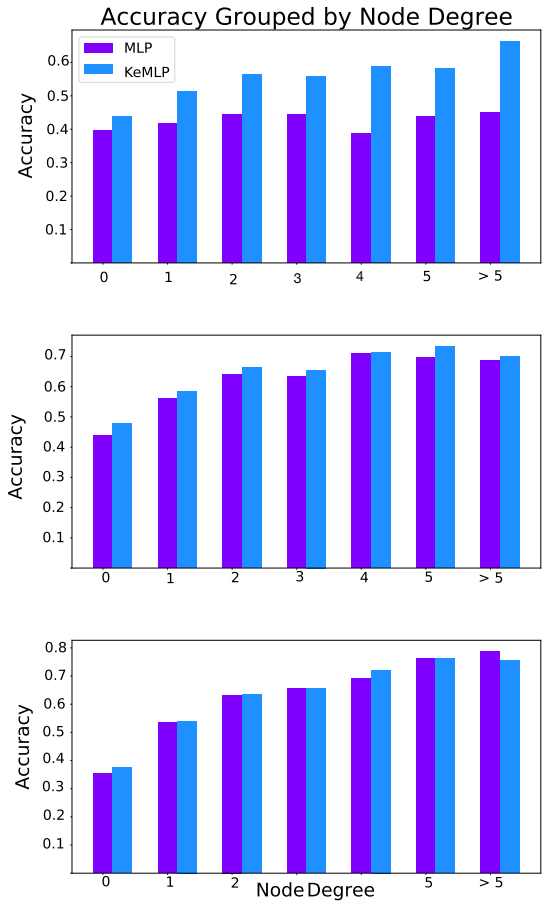


Figure 6.3.: The accuracy grouped by the node degree for MLP vs. KeMLP (above) and GCN vs. KeGCN (center) and GAT vs. KeGAT(below) on Citeseer.

This shows that rich graph structure is helpful for the node classification in general. Indeed, the MLP is a simple model that misses information on the graph structure and thus benefits from graph structure contributed by KeMLP in the form of binary predicates. On the contrary, standalone GNNs, even without knowledge enhancement layers, can handle graph structure by using message passing techniques to transfer learned node representations between neighbours. The prior knowledge introduced in the knowledge enhancer is simple. It encodes that two neighbours are likely to be of the same class. An explanation for the small difference in performance is that GNNs may be able to capture and propagate this simple knowledge across neighbours implicitly, using its message passing technique. In other words, we observe that, in this case, the introduced knowledge happens to be redundant with GNNs. However, the introduced knowledge significantly improves the accuracy of MLPs. In this context, we discuss perspectives for future work in Section 6.3.

6.2.6. Robustness to Incorrect Knowledge

Another question of interest is how the knowledge enhanced model finds a balance between knowledge and graph data when they are inconsistent. In other words, can the KeGNN successfully deal with nodes having mainly neighbours that belong to a different ground truth class and thus introduce misleading information to the node classification?

To analyse this question, we categorize the accuracy by the proportion of misleading nodes in the neighbourhood, see Figure 6.4. *Misleading nodes* are the nodes in the neighborhood of a node that have a different ground truth class than the node to be classified. It turns out that KeMLP is particularly helpful compared to MLP when the neighbourhood provides correct information. However, if the neighbourhood is misleading (if most or even all of the neighbours belong to a different class), an MLP that ignores the graph structure can lead to even better results. When comparing KeGCN and GCN, there is no clear difference. This is expected, since both models are equally affected by misleading nodes as they rely on the graph structure. Just as a GCN, the KeGCN is not necessarily robust to wrong prior knowledge since the GCN component uses the entire neighbourhood, including the misleading nodes.

When comparing GCN to KeMLP, see Figure 6.4 (below), KeMLP is more robust to misleading neighbours. While GCN takes the graph structure as given and includes all neighbours equally in the embeddings by graph convolution, the clause weights in the knowledge enhancement layers provide a way to reduce the importance of the prior knowledge. If the data frequently contradicts a clause, the model has the capacity to reduce the respective clause weight in the learning process and reduce its impact.

6.2.7. Clause Weight Learning

Furthermore, we want to examine whether the clause weights learned during training are aligned with the knowledge in the ground truth data. The clause weights provide insights on the magnitude of the refinements made by a clause. The *clause compliance* [48] measures how well the prior knowledge is satisfied in a graph.

Definition 6.2.1 (Clause Compliance). *Given a graph* $\mathbf{G} = (\mathbf{N}, \mathbf{E}, \mathbf{X}, \mathbf{Y})$ *and a clause* ϕ *in first-order logic, the clause compliance is*

$$\text{Compliance}(\mathbf{G}, \phi) = \frac{\sum_{v \in \mathbf{V_i}} \sum_{u \in \mathbf{N}(v)} \mathbb{I}[\ if\, u \in \mathbf{V_i}]}{\sum_{v \in \mathbf{V_i}} |\mathbf{N}(v)|},$$

where $\mathbf{N}(v)$ *is the first-order neighbourhood of a node* $v \in \mathbf{V}$*,* i *is a class in* $\{1, \ldots, c\}$ *and* $\mathbf{V}_i \subset \mathbf{V}$ *is the subset of nodes that have the ground truth label of class* i*:* $\mathbf{V_i} = \{v \in \mathbf{V} \mid \mathbf{y}(v) = i\}$.

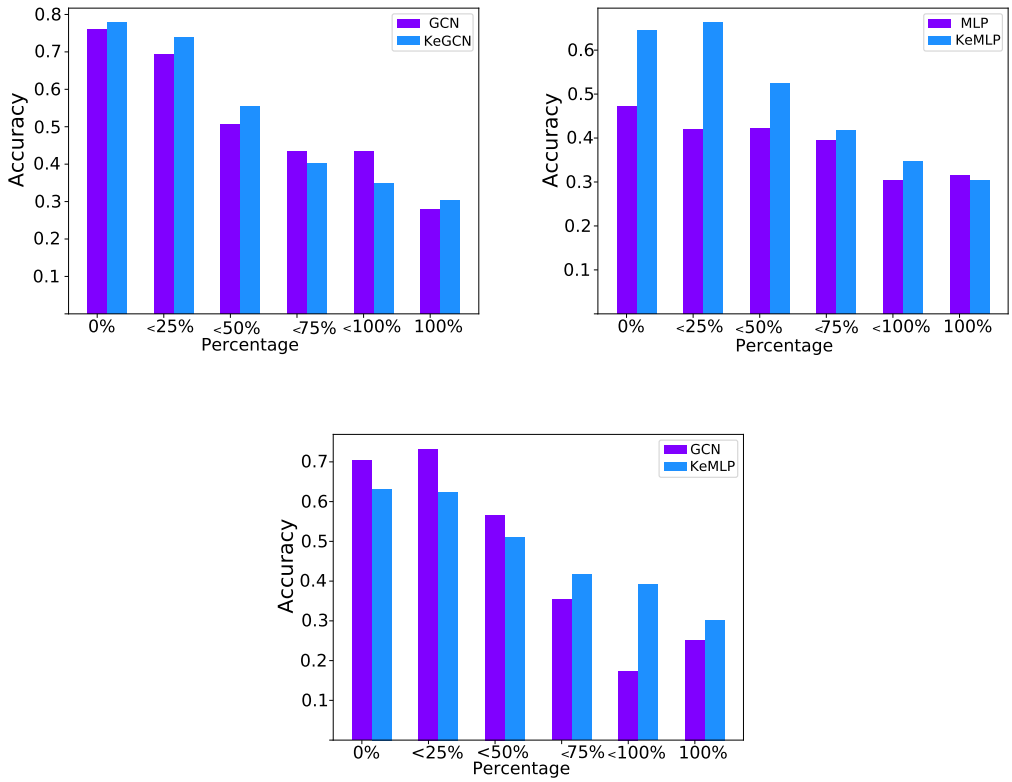


Figure 6.4.: The accuracy grouped by the ratio of misleading first-order neighbours for GCN vs. KeGCN (left), MLP vs. KeMLP (right), GCN vs. KeMLP (below) on the Citeseer dataset.

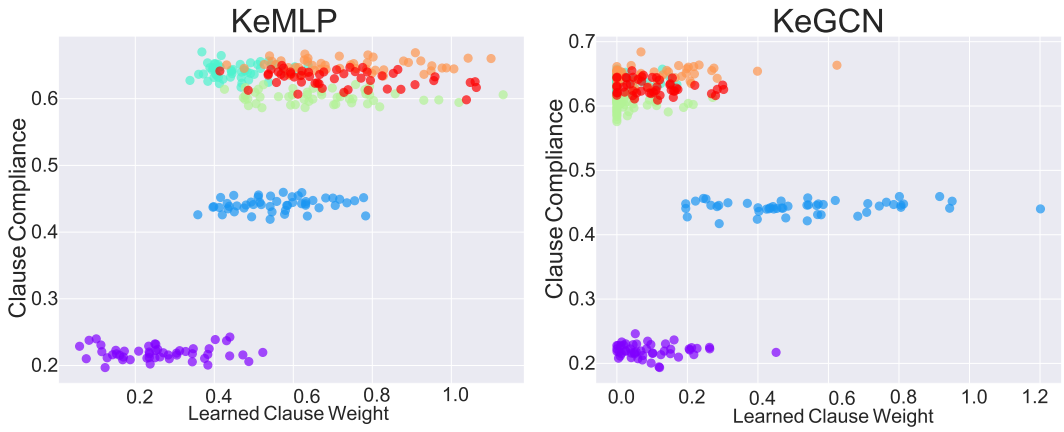


Figure 6.5.: The learned clause weights vs. clause compliance for KeMLP (left) and KeGCN (right) on the Citeseer dataset.

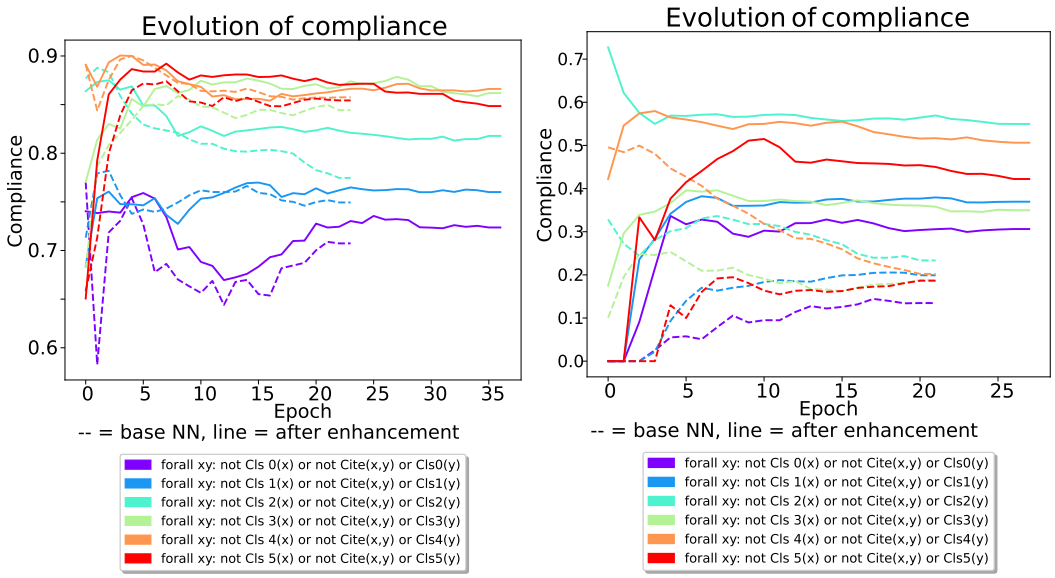


Figure 6.6.: The clause compliance during training for GCN vs. KeGCN (left) and MLP vs. KeMLP (right) on the Citeseer dataset.

In other words, the clause compliance counts how often among nodes of a class Cls_i the neighbouring nodes have the same class. The clause compliance is calculated on the ground truth classes of the training set or the predicted classes. As a reference, we measure the clause compliance based on the ground truth labels in the training set. Figure 6.5 displays the learned clause weights for KeGCN and KeMLP versus the clause compliance on the ground truth labels of the training set. For KeMLP, a positive correlation between the learned clause weights and the clause compliance on the training set is observed. This indicates that higher clause weights are learned for clauses that are satisfied in the training set. Consequently, these clauses have a higher impact on the refinements of the predictions. In addition, the clause weights corresponding to clauses with low compliance values make smaller refinements to the initial predictions. Accordingly, clauses that are rarely satisfied learn lower clause weights during the training process. In the case of KeGCN, the clause weights are predominantly set to values close to zero. This is in accordance with the absence of a significant performance gap between GCN and KeGCN. Since the GCN itself already leads to valid classifications, smaller refinements are required by the clause enhancers.

Furthermore, we analyse how the compliance evolves during training to investigate whether the models learn predictions that increase the satisfaction of the prior knowledge. Figure 6.6 illustrates the evolution of the clause compliance for the six clauses for GCN vs. KeGCN and MLP vs. KeMLP. It is observed that GCN and KeGCN yield similar results as the evolution of the compliance during training for both models is mostly aligned. For MLP vs. KeMLP the clause compliance of the prediction of the MLP converges to lower values for all classes than the clause compliance obtained with the KeMLP. This provides evidence that the knowledge enhancement layer actually improves the satisfiability of the

prior knowledge. As already observed, this indicates that the standalone GCN is able to implicitly learn to satisfy the prior knowledge even though it is not defined in a knowledge enhancement layer.

6.3. Limitations

The method of KeGNN is limited in some aspects.

Homogeneous graph structure. Here, KeGNN is only applied to homogeneous graphs. In reality, however, graphs are often heterogeneous with multiple node and edge types [217]. Adaptations are necessary on both the neural and the symbolic side to apply KeGNN to heterogeneous graphs. The restriction to homogeneous graphs also limits the scope of formulating complex prior knowledge. Eventually, the datasets used in this work and the set of prior knowledge are too simple for KeGNN to exploit its potential and lead to a significant improvement over the GNN. The experimental results show that the knowledge encoded in the symbolic component leads to significant improvement over an MLP which is not capable to capture and learn with that knowledge. This suggests that for more complex knowledge that is harder to encode in the message passing layers of a GNN, KeGNN has the potential to bring greater improvements.

Closed world assumption. Another open question is how to support negation, namely edges that do *not exist* between two nodes. The graph is assumed to be *complete* (closed world assumption). An open question is how the method scales with large multi-relational graphs and considering all negated links. In the specific case here, negated atoms $\neg\text{Cite}(x, y)$ are neglected since the clauses of this form in the experiments are always true if no edge between x and y exists. This situation is not obvious with any type of prior knowledge with several different binary predicates. Furthermore, the closed world assumption is not appropriate in many scenarios as graphs are incomplete and missing edges do not necessary mean that a relation does not hold true.

Link prediction. Furthermore, limitations occur in the context of link prediction with KeGNN. For link prediction, a neural component is required to predict fuzzy truth values for binary predicates. At present, KeGNN handles clauses containing binary predicates. However, their truth values are initialized with constant prediction values, where a high value encodes the presence of an edge. This limits the application of KeGNN to datasets for which the graph structure is complete and known a priori.

Small scale. So far, KeGNN and KENN [46] have only been applied to rather small graphs that are magnitudes smaller than graphs often found in the real-world. The scalability of the knowledge enhancement layers in KeGNN to large graphs remains an open question, which will be addressed in the next sections.

6.4. Conclusion and Outlook

In this work, we introduced KeGNN, a neuro-symbolic model that integrates GNNs with symbolic knowledge enhancement layers to create an end-to-end differentiable model. This allows the use of prior knowledge to improve node classification, while exploiting the strength of a GNN to learn expressive representations. Experimental studies show that incorporating prior knowledge has the potential to improve simple neural models such as MLPs. However, knowledge enhancement of GNNs is more difficult to achieve on the underlying and limited benchmarks, where the injection of simple knowledge about local neighbourhood is redundant with the representations that GNNs are able to learn.

With respect to the neuro-symbolic desiderata formulated in Section 4.1, KeGNN is knowledge-aware because it allows the incorporation of first-order logic clauses in the form of differentiable knowledge enhancement layers. The incorporation of more complex knowledge, such as existential quantification or conjunction, is still an open question. Thanks to the use of knowledge and the use of the base neural network, KeGNN is robust to noise in the data and also to prior knowledge that is not fully satisfiable in a logical sense. Regarding interpretability, clause weights provide a way to make the impact of knowledge on predictions more quantifiable. In terms of scalability, the application to large graphs with numerous negated atoms is an open question. There are also limitations with respect to the closed-world assumption and the focus only on node classification and homogeneous graphs. Nevertheless, KeGNN has the potential not only to improve graph completion tasks from a performance perspective, but also to improve interpretability through clause weights. This work is a step towards a holistic neuro-symbolic method on incomplete and noisy semantic data, such as knowledge graphs.

7. Knowledge Enhancement on Large Graphs

The KeGNN [207] method presented in Section 6 integrates prior knowledge in the form of logical clauses into a neural network by adding knowledge enhancement layers to the network architecture. Previous results show that the knowledge enhanced models outperform pure neural models on small graphs [136, 46]. However, the used benchmark graphs are unsatisfactory in terms of quality (homogeneous) and quantity (small size) [93, 182] and are therefore unsuitable for testing complex models. Consequently, the applicability of knowledge enhancement layers to large graphs with a high number of nodes and edges is still an open question.

This chapter focuses on the scalability of the concept of knowledge enhancement layers in the context of graphs and addresses two important obstacles. First, classic deep learning approaches commonly use mini-batch stochastic gradient descent (SGD) in the training phase. By splitting the dataset into smaller batches, memory utilisation can be reduced, which is particularly important on memory-constrained GPUs. However, the application of standard mini-batch SGD to is not well suited for graphs, because nodes are connected by edges and are consequently not independent. Therefore, the partitioning into batches must ensure that the relevant information in the form of node neighbourhood is available in every batch. Second, the problem of *neighbourhood explosion* [138, 59] can occur when stacking multiple knowledge enhancement layers with binary clauses. The number of nodes required for the computations in the knowledge enhancement layers grows exponentially with respect to the number of layers. This can lead to a drastic increase in memory usage during training on GPUs, to the point of infeasibility.

To address these issues, we propose a graph-specific mini-batching strategy called *Restrictive Neighbourhood Sampling* to make knowledge enhancement applicable to large graphs. The methods KENN [46] and KeGNN are tested on two datasets for node classification from the *Open Graph Benchmark (OGB)* [93], namely *ogbn-arxiv* and *ogbn-products*. OGB [93] is a collection of diverse datasets that provides large and informative graphs for benchmarking complex models. The experiments show that the proposed Restricted Neighbourhood Sampling technique makes the knowledge enhancement on large graphs feasible.

7.1. Problem Statement for Knowledge Enhancement on Large Graphs

This section elaborates on the problem of how the number of knowledge enhancement layers and the arity of the predicates in the logical language affect the memory requirements. Recall the concepts and notations of knowledge enhancement in Section 6.1.4.

7.1.1. Memory Requirements of a Knowledge Enhancement Layer

For a knowledge enhancement layer that enhances only unary clauses, the clause weights and the matrix with the unary groundings have to be stored. This results in memory requirements that increase linearly with the number of nodes n in the graph.

For graph data, the changes applied to a grounded predicate depend not only on the grounding of one variable, but also on the groundings of the *two* variables that are linked by binary predicates. In other words, not only the representation of a node itself from the previous layer is required, but also the representation of the nodes to which the node is connected by an edge, namely the first-order neighbourhood $\mathbf{N}_1(v)$ of a node v. To increase readibility, the notation is simplified to $\mathbf{N}_1$ from here on. Consequently, the unary predicates of the connected nodes must be encoded in a single representation. Therefore, the knowledge enhancement layers for binary clauses consist of a *join* operation that merges binary predicates and the binarised unary predicates into a single matrix $\mathbf{M}$, as shown in the Examples 6.1.6 and 6.1.7 of Chapter 6. After joining binarised unary and binary predicates, $\mathbf{M}$ looks as follows:

$$\mathbf{M} = \begin{array}{c} \\ (v_1,v_1) \\ (v_1,v_2) \\ \vdots \\ (v_n,v_n) \end{array} \overset{\begin{array}{ccc} \multicolumn{3}{c}{\text{unary groundings } \mathcal{P}_U^x} \end{array}}{\left[\begin{array}{ccc} P^x_{U_1}(v_1) & \dots & P^x_{U_p}(v_1) \\ P^x_{U_1}(v_1) & \dots & P^x_{U_p}(v_1) \\ & \ddots & \\ P^x_{U_1}(v_n) & \dots & P^x_{U_p}(v_n) \end{array}\right.} \overset{\begin{array}{ccc} \multicolumn{3}{c}{\text{unary groundings } \mathcal{P}_U^y} \end{array}}{\left|\begin{array}{ccc} P^y_{U_1}(v_1) & \dots & P^y_{U_p}(v_1) \\ P^y_{U_1}(v_2) & \dots & P^y_{U_p}(v_2) \\ & \ddots & \\ P^y_{U_1}(v_n) & \dots & P^y_{U_p}(v_n) \end{array}\right.} \overset{\begin{array}{ccc} \multicolumn{3}{c}{\text{binary groundings } \mathcal{P}_B^{xy}} \end{array}}{\left|\begin{array}{ccc} P_{B_1}(v_1,v_1) & \dots & P_{B_q}(v_1,v_1) \\ P_{B_1}(v_1,v_2) & \dots & P_{B_q}(v_1,v_2) \\ & \ddots & \\ P_{B_1}(v_n,v_n) & \dots & P_{B_q}(v_n,v_n) \end{array}\right]}. \tag{7.1}$$

The number of columns depends on the number of unary predicates p and binary predicates q and results in $2{\cdot}p{+}q$. The number of rows corresponds to the number of all possible binary combinations of nodes in the graph that equals n^2. This representation allows to encode all possible groundings of the binary predicates. Under the closed world assumption, nodes that are not connected by an edge are considered as negative grounded atoms. In consequence, the shape of $\mathbf{M}$ is $\mathbb{R}^{n^2\times(2p+q)}$. This leads to memory requirements that increase quadratically with respect to the number of nodes in the graph ($O(n^2)$).

Depending on the shape of the clauses in the prior knowledge, the number of rows can be reduced. The clauses used in Chapter 6, for example, have the form $\neg A(x) \vee \neg B(x,y) \vee C(y)$ with only negated binary grounded atoms. Since this example assumes that the groundings of the binary atoms are deterministic, any pair of two nodes between which no edge is

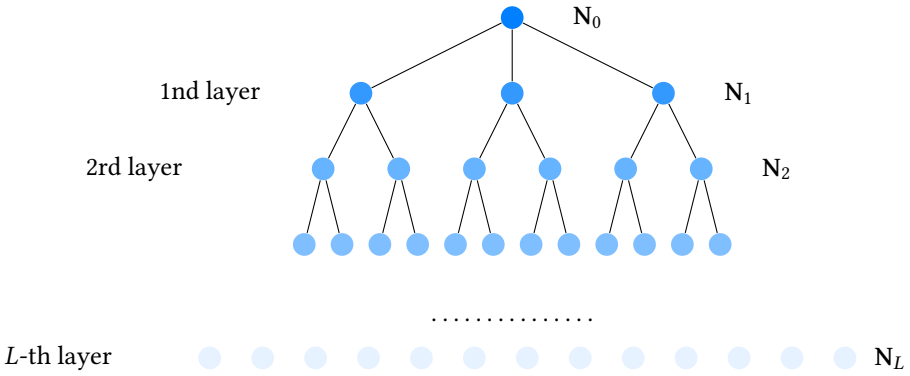


Figure 7.1.: Illustration of neighbourhood explosion.

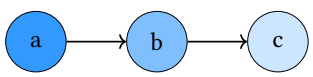
observed is interpreted as a negative grounded binary predicate. Thus, from a logic point of view, the clause is already satisfied for any grounding of the other predicates. Hence, the negative binary grounded atoms are be neglected in this case. This reduces the number of rows in $\mathbf{M}$ from n^2 to m, where m is the number of edges in the graph.

7.1.2. Multiple Knowledge Enhancement Layers

The number of knowledge enhancement layers stacked impacts the memory requirements of the whole knowledge enhanced model. When stacking multiple knowledge enhancement layers with binary clauses, the refinements recursively depend on the results of the previous layer. By stacking L binary knowledge enhancement layers, the L-th neighbourhood $\mathbf{N}_L$ is used to calculate the refinements of a grounded atom that occurs in the binary clause.

The memory requirements depend not only on the number of layers and nodes in the graph, but also on the connectivity of the graph, which is indicated by the node degree. The number of nodes to be stored in the memory increases exponentially with respect to L, see Figure 7.1). In the worst case, the required neighbourhood $\mathbf{N}_L$ results in the complete graph. The exponential growth of the relevant nodes with the number of layers is referred to as *neighbourhood explosion* [62]. It is a well-known problem in the graph neural network domain.

Example 7.1.1 (Multiple Knowledge Enhancement Layers with Binary Clauses). Here, two knowledge enhancement layers are stacked, supporting the binary clause $\phi : \forall xy : \neg AI(x) \vee \neg Cite(x, y) \vee AI(y)$. The following example graph is considered where nodes represent scientific publications edges denote citations.



We illustrate now the refinements that are applied to the atom $AI(a)$ by the clause ϕ. The refined prediction $\mathbf{Y}'_{[a,AI]}$ for the atom $AI(a)$ by the first knowledge enhancement layer with clause weight $w_\phi^{(1)}$ is calculated as follows:

$$\mathbf{Y}'_{[a,AI]} = \sigma\Bigg(\underbrace{\mathbf{Z}_{[a,AI]} + \underbrace{w_\phi^{(1)} \cdot \frac{e^{\mathbf{Z}_{[a,AI]}}}{e^{-\mathbf{Z}_{[a,AI]}} + e^{-\mathbf{Z}_{Cite[a,b]}} + e^{\mathbf{Z}_{[b,AI]}}}}_{\text{=refinements by 1st layer}}}_{\mathbf{Z}'_{[a,AI]}}\Bigg)$$

It is evident that the enhancement of a binary clause aggregates not only grounded predicates referring to the constant a, but also grounded predicates for node a's first-order neighbours in the graph, namely $\mathbf{Z}_{[b,AI]}$.

Stacking a second knowledge enhancement layer with clause weight $w_\phi^{(2)}$ results in the following computation:

$$\mathbf{Y}''_{[a,AI]} = \sigma\Bigg(\mathbf{Z}'_{[a,AI]} + w_\phi^{(2)} \cdot \underbrace{\frac{e^{\mathbf{Z}'_{[a,AI]}}}{e^{-\mathbf{Z}'_{[a,AI]}} + e^{-\mathbf{Z}'_{Cite[a,b]}} + e^{\mathbf{Z}'_{[b,AI]}}}}_{\text{=refinements by 2nd layer}}\Bigg) =$$

$$\sigma\Bigg(\mathbf{Z}'_{[a,AI]} + \underbrace{w_\phi^{(2)} \cdot \frac{e^{\left(\mathbf{Z}_{[a,AI]} + w_\phi^{(1)} \cdot \frac{e^{\mathbf{Z}_{[a,AI]}}}{e^{-\mathbf{Z}_{[a,AI]}} + e^{-\mathbf{Z}_{Cite[a,b]}} + e^{\mathbf{Z}_{[b,AI]}}}\right)}}{e^{-\mathbf{Z}'_{[a,AI]}} + e^{-\mathbf{Z}'_{Cite[a,b]}} + e^{\left(\mathbf{Z}_{[b,AI]} + w_\phi^{(1)} \cdot \frac{e^{\mathbf{Z}_{[b,AI]}}}{e^{-\mathbf{Z}_{[b,AI]}} + e^{-\mathbf{Z}_{Cite[b,c]}} + e^{\mathbf{Z}_{[c,AI]}}}\right)}}}_{\text{=refinements by 2nd Knowledge Enhancer}}\Bigg).$$

As we can see, the refinement of $\mathbf{Z}''_{[a,AI]}$ in the second layer depends on the grounded predicates of the second-order neighbours of node a, namely node c and node d.

7.2. Mini-batch Gradient Descent on Graphs

Traditional deep learning approaches often use *mini-batch gradient descent* to train neural networks on memory-constrained GPUs [127]. The dataset is divided into disjoint subsets, which are called *mini-batches* of size b. The network parameters are updated after each forward pass of a mini-batch through the neural network. This does not only lead to favorable properties such as more stable convergence and generalization, but also reduces the memory requirements, since batches can be processed independently of each other in parallel.

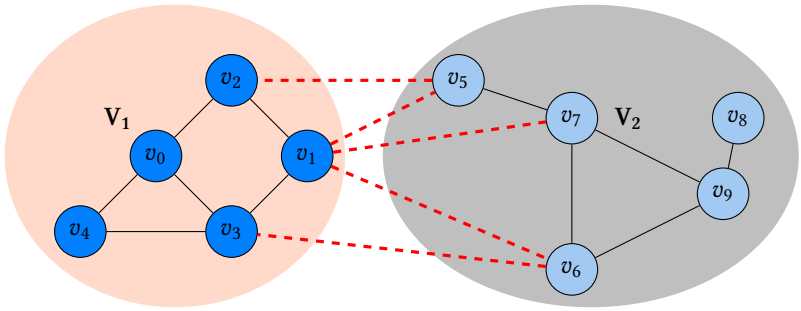


Figure 7.2.: The application of standard mini-batch gradient descent to graph data. The inter-batch edges (drawn in dotted and red) are not simultaneously available.

However, dividing a dataset into mini-batches is less obvious for graph data. This originates from the fact that the nodes in a graph are by design *not independent*. When splitting a set of nodes $\mathbf{V}$ of a graph $\mathbf{G}$ into batches, neighbouring nodes may appear in different batches and therefore not be available simultaneously. The following equation illustrates the effect on the adjacency matrix $\mathbf{A}$ when splitting a graph into mini-batches of size b, where $b \in \{1, 2, \ldots, n\}$.

$$\mathbf{A} = \begin{array}{c} \begin{matrix} & 0 & & b & & n-b & & n \end{matrix} \\ \begin{matrix} 0 \\ b \\ \\ n-b \\ n \end{matrix} \begin{bmatrix} (v_0,v_0) & \cdots & (v_0,v_b) & \cdots & (v_0,v_{n-b}) & \cdots & (v_0,v_n) \\ \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\ (v_b,v_0) & \cdots & (v_b,v_b) & & (v_b,v_{n-b}) & \cdots & (v_b,v_n) \\ & & & \ddots & & & \\ (v_{n-b},v_0) & \cdots & (v_{n-b},v_b) & & (v_{n-b},v_b) & \cdots & (v_{n-b},v_n) \\ \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\ (v_n,v_0) & \cdots & (v_n,v_b) & & (v_n,v_{n-b}) & \cdots & (v_n,v_n) \end{bmatrix} \end{array} \tag{7.2}$$

The adjacency matrix $\mathbf{A}$ of the entire graph is divided into adjacency matrices *per mini-batch* (marked in blue). The edges connecting nodes *within* a batch are called *intra-batch edges* and the edges connecting nodes in different batches are called *inter-batch edges*. The inter-batch edges are *not* simultaneously available and are consequently neglected during training.

Example 7.2.1 (Mini-batching on graphs). Consider the example graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ in Figure 7.2. The set of nodes $\mathbf{V} = \{v_0, \ldots, v_9\}$ is split into two disjoint subsets $\mathbf{V}_1 = \{v_0, \ldots, v_4\}$ and $\mathbf{V}_2 = \{v_5, \ldots, v_9\}$. The set of inter-batch edges $\{(v_1, v_5), (v_2, v_5), (v_1, v_7), (v_6, v_1), (v_3, v_6)\} \subset \mathbf{E}$ is neglected when applying standard mini-batching to the graph.

We define this *information loss* in form of inter-batch edges as a function of the batch size and the number of nodes.

Definition 7.2.1 (Information Loss). *Given a graph* $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ *with n nodes, adjacency matrix* $\mathbf{A}$ *and batch size b, the information loss* $\xi(b, n)$ *is defined as* $\xi(b, n) = n^2 - \frac{n}{b} \cdot b^2$.

Here, n^2 is the size of the adjacency matrix $\mathbf{A}$, b^2 is the size of each adjacency matrix per batch and $\frac{n}{b}$ is the number of batches. To simplify notations, we assume that $n \bmod b = 0$. If the batch size converges to the full size of the graph, an information loss of zero is obtained.

$$\lim_{b \to n} \xi(b, n) = n^2 - \frac{n}{n} \cdot n^2 = 0. \tag{7.3}$$

In contrast, the information loss increases when the batch size decreases.

$$\lim_{b \to 1} \xi(b, n) = n^2 - \frac{n}{1} \cdot 1 = n^2 - 1. \tag{7.4}$$

7.3. Restrictive Neighbourhood Sampling

The goal is to balance information loss and complexity when dividing the graph into batches. On the one hand, the batches must be small enough to fit into the memory. On the other hand, a sufficient number of nodes and edges are requisite to approximate the training on the full graph and minimize information loss.

To this end, we propose *Restrictive Neighbourhood Sampling (RNS)*. The aim is to find a set of batches in which many intra-batch edges are retained, while a few inter-batch edges are neglected. To this end, RNS creates so-called *batch graphs* at the pre-processing stage by sampling nodes from the neighbourhood. RNS proceeds by randomly sampling from the set of nodes without replacement until all nodes are taken. This way, disjoint sets of *target nodes* are obtained. Then, the samples of the τ-order neighbourhoods of the nodes are integrated. The number of neighbours to be sampled is constrained by the following hyperparameters that can be chosen in accordance with the available memory capacity and the topology of the graph:

- The *batch size b* is the number of target nodes in each batch graph.
- The *sampling depth* τ, $\tau \in \{1, \dots L\}$, is the depth of the neighbourhood taken into account.
- The *neighbour size* ρ, $\rho \in \{1, \dots n\}$ is the number of neighbours sampled per sampling depth level.

The pseudocode of RNS is introduced in Algorithm 1. A graph node-attributed, labeled graph $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{X}_V, \mathbf{Y})$ is considered as input. For the sake of readability, the index of the node features will be omitted in the following. In the first step $j = 1$ of $j \in \{1, \dots \tau\}$, initial batch graphs are drawn randomly from $\mathbf{V}$ without replacement, so that each node appears in exactly one batch together with its feature vector and label: $\{(\mathbf{V}_1^1, \mathbf{X}_1^1, \mathbf{Y}_1^1), \dots, (\mathbf{V}_S^1, \mathbf{X}_S^1, \mathbf{Y}_S^1)\}$. With batch size b, the number of batch graphs is denoted as $S = \lceil \frac{n}{b} \rceil$. These nodes $\mathbf{V}_i^1$ are called *target nodes* for the i-th batch graph. The last batch may contain less than b nodes if $n \bmod b > 0$. In the following iteration, ρ first-order neighbours are sampled from the

Algorithm 1 Restrictive Neighbourhood Sampling

Input

Graph $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{X}, \mathbf{Y})$

Parameters: batch size b, sampling depth τ, neighbour size ρ

Output

List of batch graphs: $\mathbf{G}_1(\mathbf{V}_1, \mathbf{E}_1, \mathbf{X}_1, \mathbf{Y}_1), \ldots, \mathbf{G}_S(\mathbf{V}_S, \mathbf{E}_S, \mathbf{X}_S, \mathbf{Y}_S)$

$S \leftarrow \lceil \frac{n}{b} \rceil$ ▷ *calculate the number of batches*

$\mathbf{V}_1^1, \mathbf{V}_2^1, \ldots, \mathbf{V}_S^1 \leftarrow$ randomly sample without replacement from $\mathbf{V}$ ▷ *create target node sets*

for $i \in \{1, \ldots, S\}$ **do**

  for $j \in \{1, \ldots, \tau\}$ **do**

    $\mathbf{N}_i^j \leftarrow$ randomly sample ρ nodes from $\mathbf{N}_1(\mathbf{V}_i^j)$

    $\mathbf{V}_i^{j+1} \leftarrow \mathbf{V}_i^j \cup \mathbf{N}_i^j$ ▷ *add to the set*

  end for

end for

target nodes' first-order neighbourhood $\mathbf{N}_1(\mathbf{V}_i^1)$ of the i-th batch graph and added to the node set of the batch graph. Together, they form the updated node set $\mathbf{V}_i^{j+1}$ for the i-th batch graph of the next iteration $j + 1$:

$$\mathbf{V}_i^{j+1} = \mathbf{V}_i^j \cup \mathbf{N}_1(\mathbf{V}_i^j) \tag{7.5}$$

To obtain a sample of the second-order neighbours, the set of first-order neighbours is traversed. For each node in the set of first-order neighbours, ρ neighbours are sampled and added. This procedure is repeated until the sampling depth $j = \tau$ is reached. The edges between the target nodes and all sampled neighbours are retained so that each batch graph corresponds to a subgraph $\mathbf{G}_i^j = (\mathbf{V}_i^j, \mathbf{E}_i^j, \mathbf{X}_i^j, \mathbf{Y}_i^j)$ with $i \in \{1, \ldots, S\}$ in iteration j. The hyperparameters b, τ and ρ have to be chosen carefully. If b and ρ are too large, the sampled graph might still exceed the available memory resources and result in out-of-memory errors during training.

In a forward pass of a knowledge enhanced neural network, the set of batch graphs is handled sequentially, see Algorithm 2. Some nodes might appear in several batch graphs as sampled neighbours. However, each node appears only in one batch graph as target node. If a node contributed to the loss more than once, a bias is introduced. For this reason, only the predictions of the target nodes contribute to the batch loss, while the predictions for the neighbouring nodes are only involved in the refinement calculations.

As mentioned in Section 7.1.1 and in particular in Equation 7.1, the memory requirements with full-batch training increase quadratically with the number of nodes in the graph. In contrast, applying standard mini-batching regardless of the graph structure leads to the loss of inter-batch edges, see Definition 7.2.1. RNS allows to constrain the problem with the parameters b, ρ and τ. RNS considers $S = \lceil \frac{n}{b} \rceil$ batch graphs of size $b + \tau \cdot \rho$, which constrains the memory requirements per batch with fixed parameters. The S batch graphs can be processed sequentially or in parallel.

Algorithm 2 Forward Pass with Batch Graphs Sampled with RNS

Input

Graph $\mathbf{G} = (\mathbf{V}, \mathbf{E}, \mathbf{X}, \mathbf{Y})$

Knowledge enhanced model $\mathcal{M}_\Theta$ with a set of trainable parameters Θ

Loss function $\mathbf{L}$

Output

Training loss per epoch l_{epoch}

$l\_\text{epoch} \leftarrow 0$

$\mathbf{G}_1, \mathbf{G}_2, \ldots, \mathbf{G}_S \leftarrow \text{RNS}(\mathbf{G}, \tau, b, \rho)$ ▷ *create batch graphs with RNS in Algorithm 1.*

for $i \in \{1, \ldots, S\}$ **do**

  $\hat{\mathbf{Y}}_i \leftarrow \mathcal{M}_\Theta(\mathbf{X}_i, \mathbf{E}_i)$ ▷ *Predict classes for all nodes in the batch graph.*

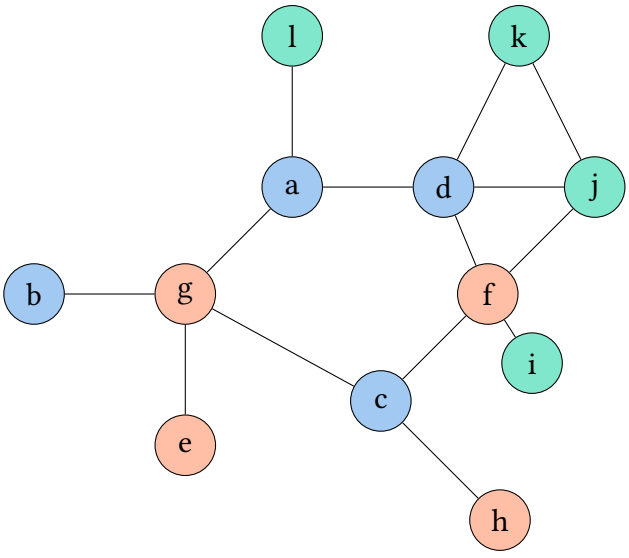
  $\hat{\mathbf{Y}} \leftarrow \hat{\mathbf{Y}}_{[1:b,\cdot]}$ ▷ *Take only predictions of target nodes*

  $l_i \leftarrow \mathbf{L}(\hat{\mathbf{Y}}, \mathbf{Y})$ ▷ *compute the batch loss*

  $l_{\text{epoch}} \leftarrow l_{\text{epoch}} + l_i$ ▷ *update the epoch loss*

end for

return l_{epoch}



a	$\vec{x}_a$	$\vec{y}_a$
b	$\vec{x}_b$	$\vec{y}_b$
c	$\vec{x}_c$	$\vec{y}_c$
d	$\vec{x}_d$	$\vec{y}_d$

$\mathbf{V}_1^1$

e	$\vec{x}_e$	$\vec{y}_e$
f	$\vec{x}_f$	$\vec{y}_f$
g	$\vec{x}_g$	$\vec{y}_g$
h	$\vec{x}_h$	$\vec{y}_h$

$\mathbf{V}_1^2$

i	$\vec{x}_i$	$\vec{y}_i$
j	$\vec{x}_j$	$\vec{y}_j$
k	$\vec{x}_k$	$\vec{y}_k$
l	$\vec{x}_l$	$\vec{y}_l$

$\mathbf{V}_1^3$

Figure 7.3.: Illustration of Restrictive Neighbourhood Sampling in Example 7.3.1. *Left:* The graph $\mathbf{G}$ with $\mathbf{V} = \{v_a, \ldots, v_l\}$. *Right:* the initial batch graphs of target nodes.

With RNS, the information loss from definition 7.2.1 is

$$\xi_{RNS}(n, b, \tau, \rho) = n^2 - \frac{n}{b} \cdot b^2 - \rho \cdot \tau \tag{7.6}$$

with $\rho, \tau \geq 0$ where $\xi_{RNS}(n, b, \tau, \rho) \leq \xi(n, b)$. Increasing the parameters batch size, sampling depth and neighbourhood size leads to a lower information loss, but to higher memory requirements. The hyperparameters can be selected tailored to the application depending on the available memory capacity, graph connectivity and size as well as the number of knowledge enhancement layers.

Example 7.3.1 (Restrictive Neighbourhood Sampling). We consider a graph $\mathbf{G}$ with $n = 12$ nodes $\mathbf{V} = \{v_a, v_b, v_c, \ldots v_l\}$ as illustrated in Figure 7.3. The parameters are set to $b = 4$, $\tau = 2$ and $\rho = 3$. In the first iteration, the set of nodes is split into sets of target nodes, which results in three batch graphs. Regarding the first batch graph for $j = 1$, we have $\mathbf{V}_1^1 = \{v_a, v_b, v_c, v_d\}$ (line 1 of Algorithm 1). Then, for $\rho = 3$ nodes are randomly sampled from $\mathbf{N}_1(V_1^1) = \{v_l, v_d, v_g, v_k, v_j, v_f, v_h, \}$, which results for example in $\mathbf{N}_1^1 = \{v_d, v_g, v_f\}$ (line 4 of Algorithm 1). This set is appended to set of target nodes: $\{v_a, v_b, v_c, v_d\} \cup \{v_d, v_g, v_f\} = \{v_a, v_b, v_c, v_d, v_g, v_f\} = \mathbf{V}_1^2$ (line 5). Then, in $j = 2$, samples are drawn from the first-order neighbourhood of the nodes in $\mathbf{V}_1^2$. This procedure is repeated for all three batch graphs, until $j = 2$ is reached. The final subgraphs are returned.

7.4. Experimental Evaluation

To test knowledge enhancement on large graphs, KeGNN is evaluated in the context of a multi-class node classification task on two benchmark datasets from the Open Graph Benchmark (OGB) [93]. OGB is a publicly available collection of various graph datasets for benchmarking. Further, we test whether the RNS technique is effective in mitigating the neighbourhood explosion problem in the context of binary clauses on large graphs.

7.4.1. Datasets

In the experiments, the OGB datasets *ogbn-arxiv* and *ogbn-products* are used [93, 61]. They represent homogeneous, node-attributed and labelled graphs for node classification. They exceed the size of the datasets used in the previous experiments with KeGNN, see Chapter 6. The datasets are further characterised in Table 7.1.

ogbn-arxiv. Ogbn-arxiv [200] is a citation graph extracted from the scientific platform Arxiv. Each node in the graph represents a research paper of the computer science domain. Directed edges between nodes indicate citations. Each paper has a 128-dimensional feature vector that is obtained with a word2vec [148] model from the text in the title and the

	ogbn-products	ogbn-arxiv
# nodes	2.449.020	169.343
# edges	61.859.140	1.166.243
# classes	47	40
Feature dimension	100	128
Avg. node degree	50.5	13.7

Table 7.1.: Overview of the ogbn-arxiv and ogbn-products datasets. The symbol # stands for the number of instances.

abstract. The documents in the graph belong to one of 40 classes. The dataset is split into training, validation and test set based on the publication dates with ratio of 54/18/28.

ogbn-products. Ogbn-products is a co-purchasing network that contains products sold on the platform Amazon [20]. The products are represented by nodes in the graph. Two nodes are connected by an edge if the respective products are purchased together. The dataset contains node features that are derived from the product descriptions and encoded as bag-of-word vectors. The dataset is split into training, validation and test set according to the sales rank with a ratio of 8/2/90. The task is to predict one of 47 product categories per node.

7.4.2. Prior Knowledge

The prior knowledge for the knowledge enhancement layers is manually derived, as in [46]. For ogbn-arxiv, the previous assumption that two documents belong to the same class when they cite each other is encoded, resulting in 40 clauses of the form

$$\forall x \forall y : \neg Class(x) \vee \neg Cite(x, y) \vee Class(y). \tag{7.7}$$

For ogbn-products, two products are supposed to belong to the same category if they are purchased together, which results in 47 clauses of the form

$$\forall x \forall y : \neg Class(x) \vee \neg CoPurchased(x, y) \vee Class(y). \tag{7.8}$$

As in [46] and as already described above, the edges and binary predicates are assumed to be known a priori and deterministic. For this reason, the preactivation of the binary predicate `Cite(x,y)` or `CoPurchased(x,y)` are set to a high positive value. As mentioned above, negative groundings of binary predicates, namely nodes that that are not connected by an edge, are neglected. The clause weights are initialized with a constant value of 0.5.

7.4.3. Hyperparameters and Experiment Setting

As in Chapter 6, knowledge enhancement layers are stacked onto an MLP and a GCN and the resulting models are designated KeMLP and KeGCN, respectively. The model architecture for the GCN and MLP proposed in [93] is adapted. The MLP and GCN consist

	ogbn-arxiv		ogbn-products	
	avg test accuracy	avg epoch time	avg test accuracy	avg epoch time
MLP	0.5403 (0.0061)	0.065	OOM	-
KeMLP	0.5713 (0.1063)	0.768	OOM	-
GCN	0.5273 (0.019)	0.182	OOM	-
KeGCN	0.4978 (0.0205)	0.888	OOM	-

Table 7.2.: Results with full-batch training on ogbn-arxiv and ogbn-products. The average test accuracies and average epoch times (in seconds) are shown. The standard deviation is noted in brackets.

of three hidden layers with hidden dimension of 256, batch normalisation layers [101] and ReLu activation after each hidden layer. For the MLP, the hidden layers are linear layers [161] and for GCN graph convolutional layers [61]. Regarding the knowledge enhanced models KeMLP and KeGCN, three knowledge enhancement layers with the clauses in Section 7.4.2 are stacked. For all models, the logarithmic softmax function [161] is employed as activation function in the last layer. The categorical cross entropy loss function [161] is optimized during training. The full set of hyperparameters is listed in the Appendix A.2.

In the experiments, the models MLP, GCN, KeMLP and KeGCN are compared. All four models are trained and evaluated with full-batch training and RNS. Multiple independent runs are conducted per experiment and the average mean accuracy μ is reported, as recommended in [93]. The following one-sided Student t-tests are performed to test whether the knowledge enhanced models outperform the baseline models MLP and GCN:

$$H_0 : \mu_{\text{MLP}} \geq \mu_{\text{KE}_{MLP}}, \quad H_1 : \mu_{\text{MLP}} < \mu_{\text{KE}_{MLP}}. \tag{7.9}$$

$$H_0 : \mu_{\text{GCN}} \geq \mu_{\text{KE}_{GCN}}, \quad H_1 : \mu_{\text{GCN}} < \mu_{\text{KE}_{GCN}}.$$

7.4.4. Implementation

The implementation of RNS and the experiments are publicly available on GitLab[1]. We use PyTorch [161] and modules from the graph learning library PyTorch Geometric [61]. The Weights and Biases application [22] is used to monitor the experiments.

7.4.5. Results

Full-batch Training. The results with full-batch training for ogbn-arxiv and ogbn-products are presented in Table 7.2. While the full-batch training on ogbn-arxiv is feasible for all models, full-batch training on ogbn-products results in an out-of-memory error for

[1] `https://gitlab.inria.fr/tyrex/scalable_ke`
at Hash `99c114a43ad625ada9e4cb5326588022579b1a53`.

	RNS on ogbn-arxiv		RNS on ogbn-products	
	avg test accuracy	avg epoch time	avg test accuracy	avg epoch time
MLP	0.5206 (0.0314)	0.09	0.5970 (0.0039)	4.17
KeMLP	0.5701 (0.0067)	2.77	0.6416 (0.0029)	6.50
GCN	0.5473 (0.0071)	1.02	0.7224 (0.0051)	4.13
KeGCN	0.5373 (0.0242)	2.94	0.7144 (0.0041)	6.78

Table 7.3.: Results with RNS training on ogbn-arxiv and ogbn-products. The average test accuracies and average epoch time are shown in seconds. The standard deviation is noted in brackets.

all experiments, as also reported in [62]. For ogbn-arxiv, KeMLP significantly outperforms the MLP, reporting a p-value of $7.21e^{-17}$ that is smaller than the significance threshold of 0.05. In case of KeGCN, no significant improvement is found with a p-value of 0.2277. It is shown that the knowledge enhanced models KeMLP and KeGCN have higher runtimes compared to the baselines. These observations are consistent with the results obtained in Chapter 6, where a significant improvement is achieved with KeMLP, but not with KeGCN.

RNS Training. The results with RNS training are displayed in Table 7.3. The models for ogbn-products can now be trained successfully without out-of-memory errors. For both datasets, KeMLP significantly outperforms the MLP. For KeGCN, no improvement compared to the GCN is significant. It can also be observed that the test accuracies reported for training with RNS on ogbn-arxiv are superior to the test accuracies for full-batch training. This comparison cannot be made for ogbn-products, as no results were obtained from full-batch training.

Overall, our results in this chapter confirm the results obtained by [46] on the Citeseer dataset with KeMLP and the results in Chapter 6. The KeMLP outperforms the MLP, but the KeGCN does not outperform the GCN. As already detailed in Section 6.2.4, several hypotheses support this observation. The GCN can handle relational information and is therefore a more complex model than the MLP, which relies only on node features. As a result, a GCN's knowledge gain is expected to be less than that of an MLP. Furthermore, each knowledge enhancement layer introduces clause weight parameters. In the case of ogbn-arxiv, there are 40 clauses to be satisfied. With three knowledge enhancement layers this leads to 120 additional training parameters. This might lead to overfitting. The prior knowledge in these experiments handcrafted based on an assumption concerning the relationship between document class and citations. If this relationship is not present in the dataset, the knowledge enhancement layer might introduce additional noise. In order to better investigate the conjunction of graph neural networks with knowledge enhancement layers, further experiments with different sets of logic formulae and other datasets are a future work.

Similar observations as in Chapter 6 are now obtained *at scale*. Although no significant prediction improvement has yet been observed, the RNS training technique is effective in making knowledge enhancement training feasible on memory-constrained GPUs. It

is therefore a step towards more extensive experimentation and advancement of neuro-symbolic techniques.

7.5. Limitations and Perspectives

While this chapter proposes solutions for knowledge enhancement at scale, the aforementioned limitations of KeGNN remain. These include the applicability to heterogeneous graphs, the consideration of links under the open-world assumption and the adaptability to link prediction. A line of future work is the application of knowledge enhanced neural networks to heterogeneous graphs. An appropriate benchmark compromising node features is the Wikilumni dataset [2]. Another line of future work is to improve the scalability of the implementation of knowledge-based neural networks, for example through parallelization techniques.

RNS is also limited in some aspects. Firstly, the reduction method of the neighbourhood is done once at pre-processing stage. An extension would be to re-sample the batch graphs at every epoch [80, 221]. Sampling per epoch would increase the coverage and is likely to introduce neighbours that were not sampled at the epoch before. Furthermore, clustering methods [37] may be useful to find a trade-off between batch size and information loss. The appropriate choice of RNS parameters based on experimental findings is still an open question.

7.6. Conclusion

This chapter investigated how knowledge enhancement with binary clauses can be applied to large graphs. First, the memory requirements of knowledge enhancement layers with binary clauses on graphs was analysed. It was shown that the problem of neighbourhood explosion can occur when multiple knowledge enhancement layers are used. To alleviate this problem, RNS was introduced, which makes it possible to control the space requirement using parameters.

To test whether RNS is effective, knowledge enhancement was applied to a GCN and an MLP, which were tested on the benchmark datasets ogbn-arxiv and obgn-products The KeMLP outperforms an MLP significantly, while no significant improvements is achieved for the GCN. These results are aligned with the results reported in Chapter 6.

Even though no significant prediction improvement has been observed yet, the RNS training technique is shown to be effective in rendering the training of knowledge enhancement with binary clauses feasible on memory-constrained GPUs. Furthermore, to the best of our knowledge, this is the first application of knowledge enhancement layers to a large-scale benchmark from the graph neural network domain. It is therefore an important step towards addressing scalability aspects in neuro-symbolic AI and enabling applications on real-world use cases.

Regarding the neuro-symbolic desiderata, the points discussed in Section 6.4 are still relevant. However, this chapter was dedicated to the topic of scalability in the context of knowledge enhanced neural networks on graphs. While full-batch KeGNN training is not applicable on memory-constrained GPUs, it can be applied to large graphs by using sampling methods such as RNS.

8. `RuleKGE`: Learning Rule-Injected Knowledge Graph Embeddings on Incomplete Knowledge Graphs

As explained in Chapter 1.1.2, knowledge graphs are a rich source of information that has recently received a lot of attention from research and industry. Knowledge graph embeddings represent the entities and relations of a knowledge graph in the vector space with the intention of capturing its regularities geometrically. However, their training and evaluation process is subject to some major limitations. First, the informative value of the embedding vectors depends on the one hand on the expressiveness and inductive capacity of the chosen knowledge graph embedding method, but also on the quality of the training data. Patterns that are not observed in the data during training may fail to be captured in the model and may not be retained during inference. Furthermore, many state-of-the-art knowledge graph embedding methods fail to capture common inference patterns [3].

Another challenge is the creation of negative examples. Knowledge graphs only store positive facts that are known to be true. However, in order to identify negative facts and avoid overgeneralisation to the positive facts, negative facts need to be included in the training. At the same time, knowledge graphs are often incomplete. This can happen unintentionally due to flaws in graph extraction. For performance reasons, many implicit facts are not explicitly stored because they can be inferred from other facts and would therefore be redundant. Hence, knowledge graph embeddings are typically trained under the local closed world assumption. In the state-of-the-art, negative facts are randomly generated by corrupting the head or tail of facts. Therefore, the entity is replaced by another entity in the knowledge graph [26]. On the one hand, this procedure likely generates facts that obviously negative but meaningless, e.g. `(Paris, capitalOf, AngelaMerkel)`. This runs the risk of overfitting to trivial cases. On the other hand, a risk of sampling false negatives remains. In other words, negative facts may be generated that are actually true but are not included in the set of positive facts due to the incompleteness of the graph.

Most knowledge graph embedding methods focus solely on the facts. Moreover, the general knowledge about the facts from the ontology is often neglected by knowledge graph embedding methods. However, ontologies can be of great benefit to make embedding training more efficient and qualitatively better, especially with incomplete and noisy data. Neuro-symbolic methods are intended to unify the symbolic information in the ontology with the numerical information in knowledge graph embeddings.

Neuro-symbolic methods such as KeGNN is not well suited for knowledge graphs. Firstly, the dense representation of groundings is based on the closed world assumption. Since knowledge graphs are often incomplete and large, this representation is inappropriate. In dense adjacency matrix representations, every combination of entities has to be encoded, which is not feasible for large graphs. In addition, interpreting unobserved facts as incorrect facts can result in false negative facts for incomplete graphs. Furthermore, KeGNN addresses the task of node classification. Its application to the task of link prediction has not yet been studied. However, in knowledge graph research, link prediction is a task of high importance, as it has many applications, e.g. in recommender systems, biological networks or information retrieval.

In this chapter, the neuro-symbolic method called *RuleKGE* is presented for training knowledge graph embeddings with the support of rules. RuleKGE belongs to the category of knowledge-driven graph augmentation approaches and the objective is to learn meaningful embeddings on incomplete knowledge graphs while exploiting ontological knowledge. To this end, a knowledge graph embedding model and a Datalog reasoner are combined. The reasoner is used for *positive reasoning* and *negative reasoning*. In positive reasoning, implicit facts are made explicit. In negative reasoning, rules are used to generate more reliable negative facts. The aim is to improve the generation of negative facts through reasoning, with two objectives: Firstly, to generate more meaningful negative facts, and secondly, to reduce the risk of false negatives. The reasoning steps are performed per batch graph in the training set, and the inferred positives and facts are iteratively added to augment the explicit facts in the batch graph. In this way, knowledge in the form of additional facts is introduced at the training stage. This allows the model to learn patterns that can be useful at inference.

RuleKGE is evaluated through experiments on the *Family dataset* [120], which encodes kinship relationships between entities. Experiments are conducted with different sets of prior logic rules that are typically found in an ontology. These rule sets include for example symmetry, inversion, composition and antisymmetry patterns. The experiments show that RuleKGE achieves the goal of learning more meaningful knowledge graph embeddings despite incomplete datasets and unseen relations.

8.1. Incomplete Knowledge Graphs

First, we formally define the incompleteness of a knowledge graph. Given a knowledge graph $\mathbf{K} = (\mathcal{E}, \mathcal{R}, \mathcal{W})$ with finite sets of entities $\mathcal{E}$ and relations $\mathcal{R}$, consider the closed world $\mathcal{W} \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ with facts $f \in \mathcal{W}$. The total number of facts in the world is $n = |\mathcal{W}| = |\mathcal{R}| \cdot |\mathcal{E}|^2$. This is shown schematically in Figure 8.1. $\mathcal{W}$ consists of two disjoint sets of positive $\mathcal{W}^+$ and negative $\mathcal{W}^-$ facts, so that $\mathcal{W} = \mathcal{W}^+ \cup \mathcal{W}^-$ and $\mathcal{W}^+ \cap \mathcal{W}^- = \emptyset$. Consider that we observe a set of positive facts $\mathcal{F}^+ \subset \mathcal{W}$. We call them *explicit positive facts*. Knowledge graphs typically do not explicitly denote negative facts. Various assumptions can be made to categorise the remaining facts $\mathcal{W} \backslash \mathcal{F}^+$ into positive and negative facts.

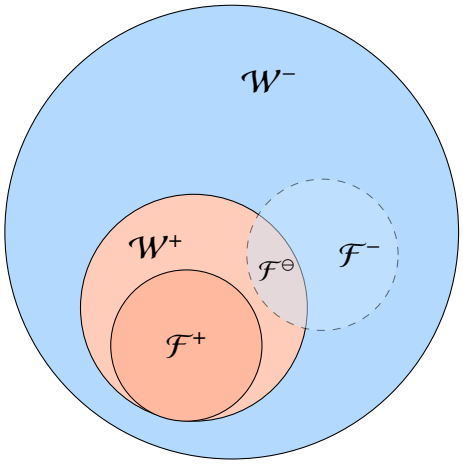


Figure 8.1.: Illustration of incomplete knowledge graphs.

The *Closed World Assumption (CWA)* assumes that $\mathcal{W}^+ = \mathcal{F}^+$. Consequently, all facts that are not in $\mathcal{F}^+$ are assumed to be negative: $\mathcal{W}^- = \mathcal{E} \times \mathcal{R} \times \mathcal{E} \backslash \mathcal{F}^+$. Under the *Local Closed World Assumption (LCWA)*, only a subset of negative facts $\mathcal{F}^- \subset \mathcal{W}^-$ is considered, since the number of facts in $\mathcal{W}^-$ is usually orders of magnitude larger than in $\mathcal{W}^+$: $|\mathcal{W}^-| \gg |\mathcal{W}^+|$. Under the *Stochastic Local Closed World Assumption (SCLWA)*, the set of negative facts $\mathcal{T}^-$ is randomly sampled from $\mathcal{W}^- = \mathcal{E} \times \mathcal{R} \times \mathcal{E} \backslash \mathcal{F}^+$. This is achieved by randomly replacing the head or tail for a positive fact $(h, r, t) \in \mathcal{F}^+$ and replacing it with a random entity $h', t' \in \mathcal{E}$ to generate a negative fact (h', r, t) or (h, r, t'). This procedure is called *uniform negative sampling* [9, 26].

In contrast, under the *Open World Assumption (OWA)*, we assume that the remaining facts are positive or negative, but we do not know to which set each fact belongs to: $\mathcal{W} \backslash \mathcal{F}^+ = \mathcal{W}^+ \backslash \mathcal{F}^+ \cup \mathcal{W}^-$, with disjointness $\mathcal{W}^+ \backslash \mathcal{F}^+ \cup \mathcal{W}^- = \emptyset$. The set of positive facts that are not contained in the set of explicit positive facts $\mathcal{W}^+ \backslash \mathcal{F}^+$ is referred to as *implicit positive facts*. Under the *Stochastic Open World Assumption* the set of negative facts $\mathcal{F}^-$ is sampled whereby the facts might be *false negative facts*. In other words, the set of negative facts $\mathcal{F}^-$ consists of true negatives $\mathcal{F}^- \cup \mathcal{W}^-$ and false negatives: $\mathcal{F}^\ominus = \mathcal{F}^- \cup \mathcal{W}^+$.

8.2. Method

We propose *RuleKGE*, a neuro-symbolic method that combines a reasoning engine with logical rules and a knowledge graph embedding method to learn embeddings for an incomplete graph under the OWA. The aim is to reduce the bias introduced by false negatives and incomplete positive facts, thereby improving the quality of the resulting knowledge graph embedding and link predictions.

RuleKGE takes as input a set of positive facts and a set of rules specified in Datalog, see Section 2.1. The rules are user-defined or originate from an ontology. They describe patterns that the facts in the knowledge graph should respect. During training, a Datalog

program is instantiated given a subset of the explicit facts and the set of rules. By forward chaining, a set of positive and negative facts are inferred with the rules forming an *augmented graph* of positive and negative facts. This augmented graph is used in the training of the knowledge graph embedding method and is the input to the loss function.

8.2.1. Reasoning Engine

The reasoning engine is based on a set of rules Σ and a set of facts $\mathcal{F}$, which together form a Datalog program. $\mathcal{F}[r] = [(e_h, e_t)^0, \ldots, (e_h, e_t)^n]$ describes a set of n facts under the relation $r \in \mathcal{R}$ with head and tail entities $e_h, e_t \in \mathcal{E}$. We consider the sets of *positive relations* $\mathcal{R}^+$ and *negative relations* $\mathcal{R}^-$.

The set of *positive facts* $\mathcal{F}^+[r]$ is denoted as a joint set of facts under all positive relations $r \in \mathcal{R}^+$:

$$\mathcal{F}^+[r] = \bigcup_{r \in \mathcal{R}^+} \mathcal{F}[r]. \tag{8.1}$$

The set of *negative facts* $\mathcal{F}^-[r]$ is denoted as a joint set of facts under all negative relations $r \in \mathcal{R}^-$:

$$\mathcal{F}^-[r] = \bigcup_{r \in \mathcal{R}^-} \mathcal{F}[r]. \tag{8.2}$$

Rules encode semantics about the facts in a graph. While the positive facts in the graph represent the grounded atoms in the program, the rules are predefined and formulate knowledge about the facts in the domain.

Rules. The rules are formulated in the logical language $\mathcal{L} = \{\mathcal{C}, \mathcal{P}, \mathcal{V}\}$, which consists of finite sets of constants $\mathcal{C}$, variables $\mathcal{V}$ and predicates $\mathcal{P}$ with arity in $\{1, 2\}$, namely unary and binary predicates. Binary predicates $P(x, y)$ indicate relations between two variables x and y. Unary predicates are denoted as $P(\_, y)$ and $P(x, \_)$ and refer to the variable x or y. The operators negation $\neg$, conjunction $\wedge$, disjunction $\vee$ and implication $\leftarrow$ allow to form logical expressions of predicates. The rules are formulated as Horn rules in $\mathcal{L}$, consisting of a head η and a body B. The body B contains a predicate or a conjunction of predicates, while the head is a single atom. The rules are read from the right to the left as "If-Then" rules. The variables that occur in the head must occur at in at least one atom in the body. Further, we consider rules where the atoms in the body of the rule must not contain a negative relation.

The rules are distinguished in positive and negative rules. A rule ϕ^+ with a head atom with a positive relation $r \in \mathcal{R}^+$ is called a *positive rule*:

$$\phi^+ : \eta \leftarrow \beta. \tag{8.3}$$

A finite set of positive rules is denoted as $\Sigma^+ = \{\phi_0^+, \ldots, \phi_n^+\}$.

Example 8.2.1 (Positive Rules). Here, the set of positive rules Σ^+ describes the patterns composition and hierarchy between the relations `parent, mother, father` and `sibling`.

```
% define relations
rel parent, mother, father, sibling

% define rules
parent(x,y) ⟵ mother(x,y) or father(x,y)
sibling(y,z) ⟵ parent(x,z) and parent(x,y)
```

In constrast, a rule with a head atom with a negative relation $r \in \mathcal{R}^-$ is called a *negative rule*:

$$\phi^- : \neg\eta \leftarrow \beta. \tag{8.4}$$

A finite set of negative rules is denoted as $\Sigma^- = \{\phi_0^-, \ldots, \phi_n^-\}$.

Example 8.2.2 (Negative Rules). Here, the set of rules Σ^- describes the patterns anti-symmetry and mutual exclusion between the relations `mother` and `father`.

```
% define relations
rel mother, father
rel not_mother, not_father

% define rules
not_mother(x,y) ⟵ mother(y,x)
not_father(x,y) ⟵ father(y,x)
not_father(x,y) ⟵ mother(y,x)
```

Facts. While rules are assumed to be provided, the explicit positive facts contained in the graph are interpreted as grounded atoms in the logic program. The set of facts $\mathcal{F}^+$ must be consistent with the rules in Σ^+. In other words, the facts added to the program have to be a *model* of the rules. A *batch graph* $\mathcal{B}^+$ is a subset of explicit positive facts $\mathcal{B}^+ \subseteq \mathcal{F}^+$ of size b. Given $\mathcal{L}$, the facts in $\mathcal{B}^+$ represent predicates grounded to constants. The set of binary predicates in $\mathcal{L}$ is a subset of the relations $\mathcal{R}$ in the graph.

Given a set of rules, the facts are divided into *extensional* and *intensional* facts, depending on the relation $\mathcal{F}^+[r]$ associated with them. An extensional relation is a relation that occurs only in the body of the rules, while an intensional relation is a relation that occurs in the head of a rule.

Example 8.2.3 (Positive Rules and Facts). The previous example on positive rules is extended with the facts $\mathcal{F}^+[\texttt{mother}]$ and $\mathcal{F}^+[\texttt{father}]$ for the relations $\{\texttt{mother}, \texttt{father}\} \subset \mathcal{R}$:

```
% define relations
rel parent, mother, father, sibling

% define rules
parent(x,y) ⟵ mother(x,y) or father(x,y)
```

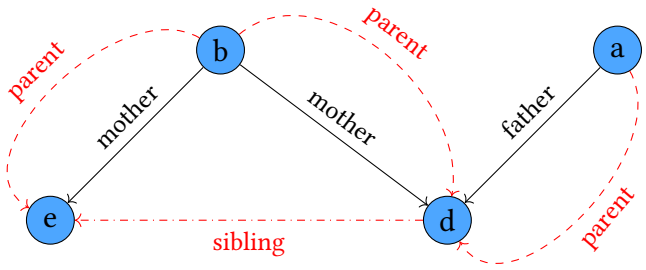


Figure 8.2.: Illustration of Example 8.2.4. The black arrows indicate the set of explicit positive facts $\mathcal{B}^+$. In the first reasoning step, the implicit positive facts under the relation parent (dashed, red) are inferred. In the second reasoning step, the fact under the relation sibling (dashed, red) is inferred . All facts in the graph form the set $\mathcal{B}^{+*}$ when the fixpoint is reached.

```
6     sibling(y,z) ⟵ parent(x,z) and parent(x,y)

8     % define facts
9     mother = [("b", "e"), ("b", "d")]
10    father = [("a", "d")]
```

8.2.2. Reasoning with Positive Rules

The evaluation of a rule $\phi^+ \in \Sigma^+$ on a set of facts $\mathcal{B}^+$ involves the following steps. First, the body expression B of the rule is evaluated. Second, the corresponding head atom η is inferred. Third, the inferred positive fact is concatenated with the explicit positive facts. Thus, the reasoner implements a function from a set of explicit positive facts and the rules to an updated set of positive facts containing explicit and implicit positive facts.

A fact f' is an *immediate consequence* of Σ^+ and $\mathcal{B}^+$ if f' is in $F^+[r]$ for some extensional relation $r \in \mathcal{R}$ or if $f' \leftarrow f_1, \ldots, f_n$ is an instantiation of a rule $\phi \in \Sigma^+$ and $f_1, \ldots f_n \in \mathcal{B}^+$. The function T defines a set of immediate consequences $\mathcal{B}^+_{i+1}$ for the positive rules Σ^+ and explicit positive facts $\mathcal{B}^+_i$:

$$T : \Sigma^+, \mathcal{B}^+_i \mapsto \mathcal{B}^+_{i+1} \tag{8.5}$$

For a batch graph of explicit positive facts $\mathcal{B}^+$, the output of $T(\Sigma^+, \mathcal{B}^+)$ consists of all facts $f \in \mathcal{W}^+$ that are immediate consequences of $\mathcal{B}^+$ and Σ^+. The operator T is monotone: $\mathcal{B}^+ \subseteq T(\mathcal{B}^+)$. While new facts can be added, once inferred facts cannot be removed. The function T can be applied several times to the set of facts: $T^2(\Sigma^+, \mathcal{B}^+) = T(T(\Sigma^+, \mathcal{B}^+))$, $T^3(\Sigma^+, \mathcal{B}^+) = T(T(T(\Sigma^+, \mathcal{B}^+)))$ and $T^n(\Sigma^+, \mathcal{B}^+) = T(\ldots T(T(\Sigma^+, \mathcal{B}^+)))$. It follows

$$\mathcal{B}^+ \subseteq T(\Sigma^+, \mathcal{B}^+) \subseteq T^2(\Sigma^+, \mathcal{B}^+) \subseteq \ldots \subseteq T^n(\Sigma^+, \mathcal{B}^+). \tag{8.6}$$

Starting from the explicit positive facts $\mathcal{B}^+$, this process is repeated until a *fixpoint* is reached. A fixpoint is reached at step N for all $J > N$ if

$$T^N(\Sigma^+, \mathcal{B}^+) = T^J(\Sigma^+, \mathcal{B}^+). \tag{8.7}$$

The *least fixpoint operator* $\text{lfp}^\circ(T)$ is defined as

$$\text{lfp}^\circ(T) = T \circ \cdots \circ T = T^N \tag{8.8}$$

if there exists a minimum $N > 0$ such that $T^N(\mathcal{B}^+) \overset{\circ}{=} T^{N+1}(\mathcal{B}^+)$.

Overall, the function implemented by the positive reasoner is a least fixpoint operator:

$$\text{lfp}^\oplus(T) : \mathcal{B}^+, \Sigma^+ \mapsto \mathcal{B}^\oplus. \tag{8.9}$$

The set of facts generated at the fixpoint is $\mathcal{B}^\oplus = T^N(\Sigma^+, \mathcal{B}^+)$. The final fact set after reasoning $\mathcal{B}^\oplus$ is defined as the disjunction of the fact sets for all relations in the graph at the fixpoint.

$$\mathcal{B}^\oplus = \bigcup_{\forall r \in \mathcal{R}} \mathcal{F}[r]^\oplus. \tag{8.10}$$

The number of inferred implicit positive facts is $\Delta^+ = |\mathcal{B}^\oplus| - |\mathcal{B}^+|$.

Example 8.2.4 (Reasoning with Positive Rules and Facts). As illustrated in Figure 8.2, the rules Σ^+ and the explicit positive facts $\mathcal{B}^+$ are used to infer implicit positive facts with the least fixpoint operator in Equation 8.12. The explicit positive facts $\mathcal{B}^+ = \{$ `mother = [("b", "e"), ("b", "d")], father = [("a", "d")]` $\}$ are extensional facts for rule 1. Therefore, in step $n = 1$, the inferred facts $\{$ `parent = [("b", "e"), ("b", "d"), ("a", "d")]` $\}$ are added to the set of positive facts. In step $n = 2$, the facts {`parent=[("b","e"), ("b", "d")]`} are extensional facts for rule 2 and allow to infer {`sibling = [("d", "e")]`}. A fixpoint is reached for $n = 2$, since no more implicit facts are inferred by reapplying the rules to the fact set. The set of facts $\mathcal{B}^\oplus$ returned by the reasoner consists of all facts derived at $n = 2$.

```
% define relations
rel parent, mother, father, sibling

% define rules
parent(x,y) ← mother(x,y) or father(x,y) % rule 1
sibling(y,z) ← parent(x,z) and parent(x,y) % rule 2

% Initial facts
mother = [("b", "e"), ("b", "d")]
father = [("a", "d")]
sibling = [] % no facts for these relations
parent = []

% facts after step 1
mother = [("b", "e"), ("b", "d")]
```

```
father = [("a", "d")]
parent = [("b", "e"), ("b", "d"), ("a", "d")] % new facts
sibling = []

% facts after step 2
mother = [("b", "e"), ("b", "d")]
father = [("a", "d")]
parent = [("b", "e"), ("b", "d"), ("a", "d")]
sibling = [("d", "e")] % new facts
```

8.2.3. Reasoning with Negative Rules

While positive reasoning aims to augment the set of explicit positive facts by inferring implicit positive facts, negative reasoning seeks to generate *negative facts* $\mathcal{B}^- \subseteq \mathcal{F}^-$ given a set of negative and positive rules Σ^+, Σ^- with $\Sigma^- \neq \emptyset$. Note that we restrict the set negative facts $\mathcal{B}^-$ to be intensional facts. This signifies that additional positive facts can never be derived from negative facts and positive rules. First, a set of *negative relations* $\mathcal{R}^-$ is instantiated for each relation that occurs as the negated head in a negative rule:

$$\mathcal{R}^- = \{r \in \mathcal{R} | \exists \phi^- \in \Sigma^- : \neg r = \neg \eta\}$$

The set of facts under a negative relation $r \in \mathcal{R}^-$ are $\mathcal{F}^-[r]$. In the program, these negated relations are initialised in addition to the positive relations, as shown in the following example.

Example 8.2.5 (Negative Rules). The set of rules Σ^- describe the patterns antisymmetry and mutual exclusion for the relation `mother`.

```
% define relations
rel mother, father
rel not_mother, not_father % negative relations

% define rules
not_mother(x,y) ⟵ mother(y,x)
not_father(x,y) ⟵ mother(y,x)
```

As for positive reasoning, positive facts can serve as extensional facts to derive intensional facts. For negative rules, however, the intensional facts are negative, which modifies function T in Equation 8.5 to

$$T : \Sigma^+, \Sigma^-, \mathcal{B}_i^+ \mapsto \mathcal{B}_{i+1}^-. \tag{8.11}$$

As in the case of positive reasoning, the operator T is repeatedly applied until a fixpoint N is reached and no more negative facts can be derived. This results in the set of negative facts $\mathcal{B}^\ominus$.

$$\text{lfp}^\ominus(T) : \mathcal{B}^+, \Sigma^+, \Sigma^- \mapsto \mathcal{B}^\ominus. \tag{8.12}$$

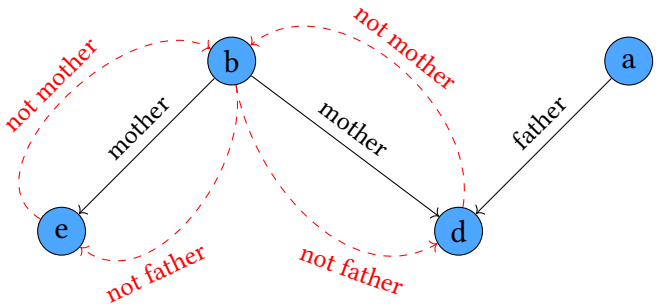


Figure 8.3.: Illustration of Example 8.2.6. Starting from the explicit positive facts, which are indicated by black arrows, the facts with red dashed lines are inferred in step $n = 1$, where a fixpoint is reached. The negative facts $\mathcal{B}^{\ominus}$ (red, dashed) are returned.

The resulting set of negative facts $\mathcal{B}^{\ominus}$ is defined as the disjunction of all sets of fact under negative relations:

$$\mathcal{B}^{\ominus} = \bigcup_{\forall r \in \mathcal{R}^-} \mathcal{F}^-[r]^{\ominus}. \tag{8.13}$$

The number of inferred negative facts is $\Delta^- = |\mathcal{B}^{\ominus}|$.

Example 8.2.6 (Negative Rules and Facts). The negative reasoning process is illustrated in the following example, see Figure 8.3. The set of explicit positive facts $\mathcal{B}^+ =\{$ `mother=[("b", "e"), ("b", "d")], father=[("a", "d")]` $\}$ and the rules 1 and 2 are provided. They model the antisymmetry of the mother relation and the mutual exclusion of the mother and father relation. The negative facts $\mathcal{B}^{\ominus} =\{$ `not mother = [("b", "e"), ("d", "b")], not father = [("b", "e"), ("b", "d")]` $\}$ are inferred.

```
% define relations
rel mother, father, aunt
rel not_mother, not_father % negative relations

% define rules
not_mother(x,y) ⟵ mother(y,x) %rule 1
not_father(x,y) ⟵ mother(y,x) %rule 2

% define facts
mother=[("b", "e"), ("b", "d")]
father=[("a", "d")]

% facts after step one
not_mother = [("b", "e"), ("d", "b")]
not_father = [("b", "e"), ("b", "d")]
```

8.2.4. Training and Reasoning

First, recall how knowledge graph embeddings are trained without any reasoning, as described in Section 3.1. Given a set of facts $\mathcal{F}^+$, the goal is to find representations of entities and relations in the graph. They are usually parametrized with learnable parameters Θ. The dimension of Θ and the exact way in which relations and entities are modelled depend on the knowledge graph embedding method. In any case, the score function calculates a plausibility score for a fact $(e_h, r, e_t) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, which depends on the set of learnable parameters Θ:

$$f_{score} : \{(\mathcal{E} \times \mathcal{R} \times \mathcal{E}), \Theta\} \mapsto \mathbb{R}. \tag{8.14}$$

Negative facts are usually generated by uniform negative sampling, which randomly replaces the head or tail of the fact by another entity, see Equation 3.13. The ratio of the number of positive facts to the number of negative facts is a hyperparameter $\rho \in \mathbb{R}^+ \setminus \{0\}$. It is usually set to one to produce an equal number of positive and negative facts. Then the label one is assigned to positive facts and zero to negative facts. The goal is to learn Θ to distinguish between positive and negative facts, which is a binary classification task. Therefore, a differentiable loss function $\mathbf{L}$ is minimized. It measures the distance of the score for the facts in the training set. The parameters Θ receive gradient updates $\frac{\delta \mathbf{L}}{\delta \Theta}$ and represent the resulting embedding vectors for entities and relations. The training is performed with mini-batch gradient descent. Therefore, $\mathcal{F}^+$ is divided into S batches $\{\mathcal{B}_0^+, \ldots, \mathcal{B}_S^+\}$ of size b with $S = \left\lfloor \frac{|\mathcal{F}^+|}{b} \right\rfloor$. In this notation, the last batch with less than b facts is discarded. The loss and the parameters are updated per batch. The positive and negative reasoning process in RuleKGE modifies the training loop. The architecture of RuleKGE is shown in Figure 8.4. When a set of rules is provided, reasoning is applied to the initial set of positive facts before the loss is calculated. The following sections explain how positive and negative reasoning are incorporated into the process of training knowledge graph embeddings, and how the two components are combined.

Positive Reasoning. The positive reasoning process is illustrated in the lines 4-8 of Algorithm 3. Consider a batch of explicit positive facts $\mathcal{B}_i^+$ of size b, where $i \in [1, S]$. For the sake of readability, the index of the batch is omitted. The least fixpoint operator of Equation 8.12 is used to derive $\mathcal{B}^\oplus$, given the batch of explicit positive facts $\mathcal{B}^+$ and the set of positive rules Σ^+. Depending on the coverage of the rule heads in the batch, implicit positive facts are inferred, increasing the batch size to $|\mathcal{B}^\oplus| = b + \Delta_+$. Regarding the labels of the facts, labels with value one are derived for the inferred implicit positive facts, since they are logical consequences of $\mathcal{B}^+$ and Σ^+. When only positive rules are provided, the generation of the negative facts uses uniform negative sampling, see Section 3.13. However, the number of negative facts to be generated is adjusted to $\frac{b+\Delta^+}{\rho}$.

Negative Reasoning. The negative reasoning process is described in the lines 9-17 of Algorithm 3. Given a set of negative rules Σ^- and a batch $\mathcal{B}^+$ of explicit positive facts, negative reasoning aims to infer negative facts. The reasoner returns a set of negative facts $\mathcal{B}^\ominus$ of size Δ^-. To keep the ratio of positive to negative facts ρ, the size of $\mathcal{B}^\ominus$

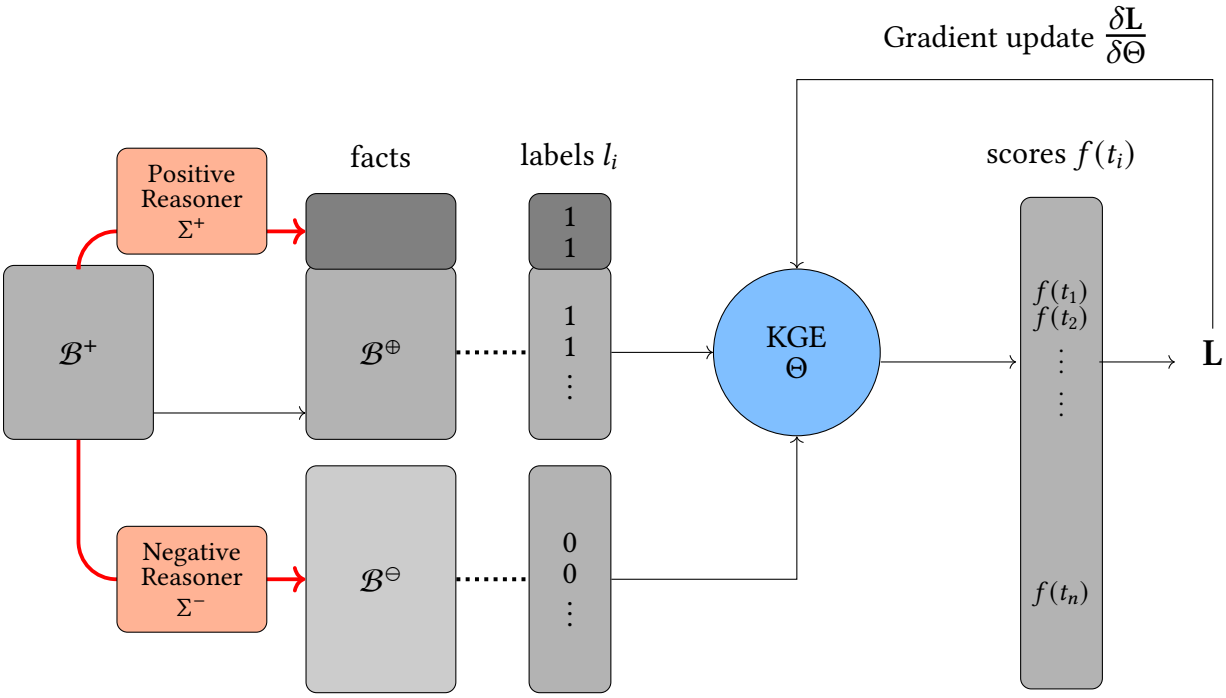


Figure 8.4.: Overview of the training loop of RuleKGE with positive and negative reasoning.

may be too small or too large compared to the number of facts actually needed. If the number of inferred negative facts is too small $\Delta^- < \frac{|\mathcal{B}^+|}{\rho}$ additional negative facts are generated with uniform negative sampling. The set of remaining facts $\{f_1, \ldots, f_k\}$ with size $k = \left\lfloor \frac{|\mathcal{B}^+|}{\rho} - \Delta^- \right\rfloor$ are randomly sampled from the set of negative facts:

$$f_1, \ldots f_k \sim \text{Uniform}(\mathcal{N}). \tag{8.15}$$

Then, the sampled negative facts are combined with the inferred negative facts: $\mathcal{B}^\ominus \cup \{f_1, \ldots f_k\}$. If the negative reasoner generates too many negative facts ($\Delta^- > \frac{|\mathcal{B}^+|}{\rho}$), facts are sampled *from* the negative facts, where $k = \left\lfloor \frac{|\mathcal{B}^+|}{\rho} \right\rfloor$:

$$f_1, \ldots f_k \sim \text{Uniform}(\mathcal{B}^\ominus). \tag{8.16}$$

Positive and Negative Reasoning. Given a rule set of positive *and* negative rules $\Sigma = \Sigma^+ \cup \Sigma^-$, both positive and negative reasoning are used to infer implicit positive facts and negative facts for training. The functions of positive and negative reasoning are composed as follows:

$$\text{lfp}^\ominus \circ \text{lfp}^\oplus(T) = \text{lft}^\ominus(\Sigma^-, \text{lfp}^\oplus(\Sigma^+, \mathcal{B}^+)). \tag{8.17}$$

The first step is the positive reasoning. Then the negative sampling takes as input the set of explicit and implicitly inferred positive facts $\mathcal{B}^\oplus$. The number of negative facts required is $\left\lfloor \frac{|\mathcal{B}^\oplus|}{\rho} \right\rfloor$.

Algorithm 3 RuleKGE. Forward Pass with Reasoning

Input
Training facts $\mathcal{F}^+$, Positive Rules Σ^+, Negative Rules Σ^-
Parameters batch size b, ratio ρ, knowledge graph embedding score function $f^{\Theta}_{\text{score}}$, set of parameters Θ

Split $\mathcal{F}^+$ into n batches $\mathbf{B} = \{\mathcal{B}^+_1, \dots \mathcal{B}^+_n\}$ of size b.
for $\mathcal{B}^+ \in \mathbf{B}$ **do**
  Initialize DatalogReasoner(Σ^+, Σ^-)
  if len(Σ^+) > 0 **then**
    $\mathcal{B}^{\oplus} \leftarrow$ positive reasoning ($\mathcal{B}^+, \Sigma^+$)
  else
    $\mathcal{B}^{\oplus} \leftarrow \mathcal{B}^+$
  end if
  if len(Σ^-) > 0 **then**
    $\mathcal{B}^{\ominus} \leftarrow$ negative reasoning ($\mathcal{B}^+, \Sigma^-$)
    if len($\mathcal{B}^{\ominus}$) < len($\mathcal{B}^{\oplus}$)/ρ **then**
      $k \leftarrow \left\lfloor \frac{len(\mathcal{B}^{\oplus})}{\rho} - \Delta^- \right\rfloor$
      $\{f_1, \dots f_k\} \leftarrow$ uniform sampling k from $\mathcal{N}$
      $\mathcal{B}^{\ominus} \leftarrow \mathcal{B}^{\ominus} \cup \{f_1, \dots f_k\}$
    else if len($\mathcal{B}^{\ominus}$) > len($\mathcal{B}^{\oplus}$)/ρ **then**
      $k \leftarrow \left\lfloor \frac{len(\mathcal{B}^{\oplus})}{\rho} \right\rfloor$
      $\mathcal{B}^{\ominus} \leftarrow$ uniform sampling k from $\mathcal{B}^{\ominus}$
    end if
  else
    $\mathcal{B}^{\ominus} \leftarrow \mathcal{N}$ ▷ *uniform negative sampling*
  end if
  positive labels $\leftarrow \{1\}^{len(\mathcal{B}^{\oplus})}$
  negative labels $\leftarrow \{0\}^{len(\mathcal{B}^{\ominus})}$
  labels $\leftarrow$ concat[positive labels, negative labels]
  facts $\leftarrow$ concat[$\mathcal{B}^{\oplus}, \mathcal{B}^{\ominus}$]
  loss $f^{\Theta}_{\text{score}}$(facts, labels)
  $\Theta \leftarrow \frac{\delta \text{Loss}}{\delta \Theta}$ gradient update
end for

Positive reasoning

Negative reasoning

8.3. Experimental Evaluation

Experiments are conducted to address the following research questions:

R#1 How does the *reasoning process* in RuleKGE affect the batches of inferred facts?

R#2 Given *incomplete data*, does positive reasoning contribute to learning meaningful embeddings of knowledge graphs?

R#3 Does *negative reasoning* contribute to learning meaningful embeddings of knowledge graphs?

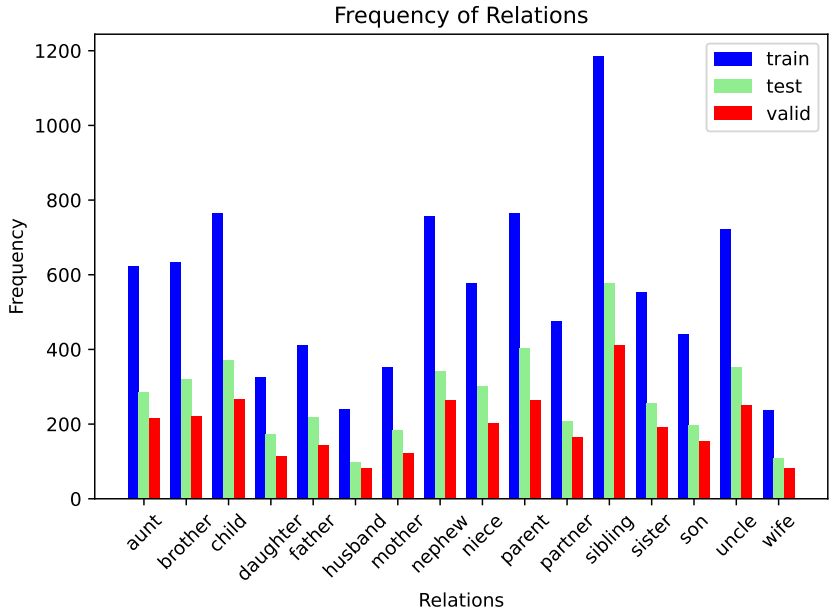


Figure 8.5.: The counts of facts in family dataset grouped by relation for train, valid and test.

R#4 Does reasoning with *intermediate concepts* lead to more meaningful embeddings of knowledge graphs?

R#5 Does RuleKGE allow to learn unseen *zero-shot relations*?

8.3.1. Dataset

In this chapter, we use the publicly available Family Dataset[1] [120]. The Family dataset is a multi-relational, non-attributed graph. The facts encode kinship relationships between individuals. The individuals represent the entities in the graph and the relations indicate kinship relationships between them. For example, the fact "Person x is the aunt of person y" is written as `(x, aunt, y)` or `Aunt(x,y)`. The initial version of the dataset in [120] consists of 12 different relations. We extend the dataset using the DLV tool [129] in order to have the following set of 16 relations: `aunt`, `uncle`, `mother`, `father`, `son`, `daughter`, `wife`, `husband`, `niece`, `nephew`, `sister`, `brother`, `partner`, `parent`, `sibling`, `child`. The frequency of the relations in the train, valid and test set is shown in Figure 8.5 and Table 8.2. The dataset contains the many-to-many relations `aunt`, `brother`, `husband`, `nephew`, `niece`, `partner`, `sibling`, `sister`, `uncle`, `wife`, the one-to-many relations (`father`, `mother`), the many-to-two relations `child`, `daughter`, `son` and the two-to-many relations (`parent`), see Table 8.2. Overall, the graph contains 2,745 unique entities. The facts are divided into disjoint sets of

[1]The version of the Family dataset used in this chapter is available here `https://github.com/Glaciohound/LERP`.

Relation	Train	Valid	Test
aunt	622	216	286
brother	633	220	321
child	765	266	370
daughter	325	113	173
father	412	143	219
husband	239	83	99
mother	352	122	184
nephew	757	263	341
niece	578	201	301
parent	764	265	403
partner	476	165	207
sibling	1.185	412	576
sister	552	192	255
son	440	153	197
uncle	721	250	351
wife	237	82	108
total	9,058	3,146	4,391

	# different tails		# different heads		characteristic
	median	max	median	max	
aunt	2	15	1	6	many-to-many
brother	2	9	1	7	many-to-many
child	1	2	1	7	many-to-two
daughter	1	2	1	6	many-to-two
father	1	11	1	1	one-to-many
husband	1	4	1	3	many-to-many
mother	1	7	1	1	one-to-many
nephew	2	9	2	8	many-to-many
niece	2	8	2	10	many-to-many
parent	1	11	1	2	two-to-many
partner	1	4	1	4	many-to-many
sibling	2	9	2	10	many-to-many
sister	2	8	1	6	many-to-many
son	1	2	1	5	many-to-two
uncle	2	17	1	16	many-to-many
wife	1	3	1	4	many-to-many

Table 8.2.: *Left*: Frequency of facts per relation in the train, valid and test set. *Right:* Overview of the relations in the Family dataset.

train, valid and test facts as suggested in [81]. The training dataset contains 9,058 facts, the validation set contains 3,246 facts and the test set contains 4,391 facts.

8.3.2. Rules

The rule sets used in the experiments are based on the inference patterns symmetry, antisymmetry, inversion, composition, mutual exclusion and hierarchy as defined in the Definitions 3.1.2 to 3.1.7 in Section 3.1.5. They are formulated in Datalog. Figure 8.6 shows the programs for positive reasoning and Figure 8.7 shows the programs for negative reasoning.

For the experiments on the composition rule set, the original Family dataset with 12 relations is used. For all other experiments, the augmented family dataset with 16 relations is used.

8.3.3. Implementation

The RuleKGE implementation is based on Python and PyTorch [161] and is publicly available on GitLab[2]. The library *PyKEEN* [9] is used to implement various knowledge graph embedding methods. For the positive and negative reasoner, we make use of the Python library *scallopy*[3] based on SCLRam [131]. *Wandb* [22] is used as experiment tracking tool.

[2] `https://gitlab.inria.fr/tyrex-public/ruleKGE`

[3] Scallopy: `https://pypi.org/project/scallopy/`

```
# symmetry
sibling(a,b)←sibling(b,a)
partner(a,b)←partner(b,a)
```

```
# inversion
parent(x,y)←child(y,x)
child(x,y)←parent(y,x)
wife(x,y)←husband(y,x)
husband(x,y)←wife(y,x)
```

```
# hierarchy
parent(a,b)←father(a,b)
parent(a,b)←mother(a,b)
parent(a,b)←son(b,a)
parent(a,b)←daughter(b,a)
sibling(a,b)←sister(a,b)
sibling(a,b)←brother(a,b)
partner(a,b)←wife(a,b)
partner(a,b)←husband(a,b)
child(a,b)←daughter(a,b)
child(a,b)←son(a,b)
```

```
# composition
aunt(x,z)←sister(x,y) and mother(y,z)
aunt(x,z)←sister(x,y) and father(y,z)
uncle(x,z)←brother(x,y) and mother(y,z)
uncle(x,z)←brother(x,y) and father(y,z)
mother(x,z)←wife(x,y) and father(y,z)
father(x,z)←husband(x,y) and mother(y,z)
```

Figure 8.6.: The programs with positive rules for the inference patterns symmetry, inversion, hierarchy, composition on the Family dataset.

```
# mutual exclusion
not father(x,y)←mother(y,x)
not aunt(x,y)←mother(y,x)
not child(x,y)←mother(y,x)
not sister(x,y)←mother(y,x)
not brother(x,y)←mother(y,x)
not wife(x,y)←mother(y,x)
not husband(x,y)←mother(y,x)
not son(x,y)←mother(y,x)
not daughter(x,y)←mother(y,x)
not nephew(x,y)←mother(y,x)
not niece(x,y)←mother(y,x)
not uncle(x,y)←mother(y,x)
...
not brother(x,y)←wife(y,x)
not father(x,y)←wife(y,x)
```

```
# antisymmetry
not mother(x,y)←mother(y,x)
not father(x,y)←father(y,x)
not son(x,y)←son(y,x)
not daughter(x,y)←daughter(y,x)
not parent(x,y)←parent(y,x)
not child(x,y)←child(y,x)
not aunt(x,y)←aunt(y,x)
not uncle(x,y)←uncle(y,x)
not wife(x,y)←wife(y,x)
not husband(x,y)←husband(y,x)
not niece(x,y)←niece(y,x)
not nephew(x,y)←nephew(y,x)
```

Figure 8.7.: The programs for neative reasoning for the inference patterns antisymmetry and mutual exclusion.

8.3.4. Analysis of the reasoner (R#1)

First, the reasoning process in RuleKGE on batch graphs is analysed in isolation from the knowledge graph embedding training.

Reasoning Time. As described in Figure 8.8 for the inversion set, the reasoning time increases linearly with the number of facts in the input batch. In line with this observation, the number of inferred facts increases linearly with the batch size.

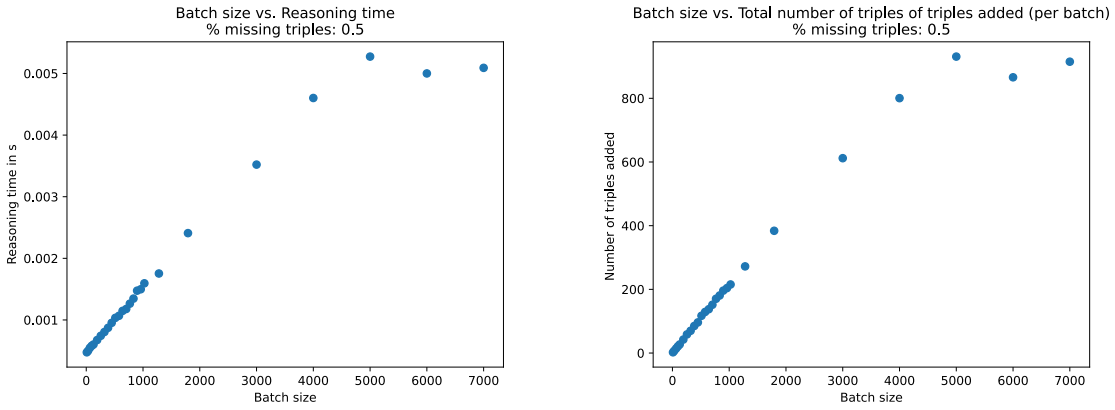


Figure 8.8.: *Left:* Reasoning Time vs. batch size. *Right:* Number of inferred facts vs. batch size for the inversion rule set.

Relation Frequency. Furthermore, the relation of the facts to be inferred depends on the predicates that occur in the head atom of the rules. Therefore, the reasoner increases the frequency of facts corresponding to the head atoms. This is shown in Figures 8.9 for the hierarchy, inversion, symmetry and composition rule sets.

Fact duplication and leakage. It may happen that the reasoner infers a fact that is already contained in another batch of the training set. In this case, the inferred fact would not be an implicit fact, but an explicit fact in another batch. Across all batches, this fact would contribute to the loss calculation several times, thus increasing the importance of the fact for the gradient updates.

Potentially, the reasoner could infer facts that are in the test set and violate the disjointness of the training, validation, and test sets. In this case, the reasoner would *leak* test data at the training stage. The magnitude of this problem is shown in Figure 8.10. It can be seen that the number of redundant facts in the train set decreases with increasing batch size, which is expected. If the reasoner considers the entire training graph, it cannot reason about facts in other batches. The number of leaked facts of the train and the test set increases with the batch size, because more facts are inferred for a larger batch size. However, in this observed case the problem is minor(~ 50 leaked test facts per 2,000 facts).

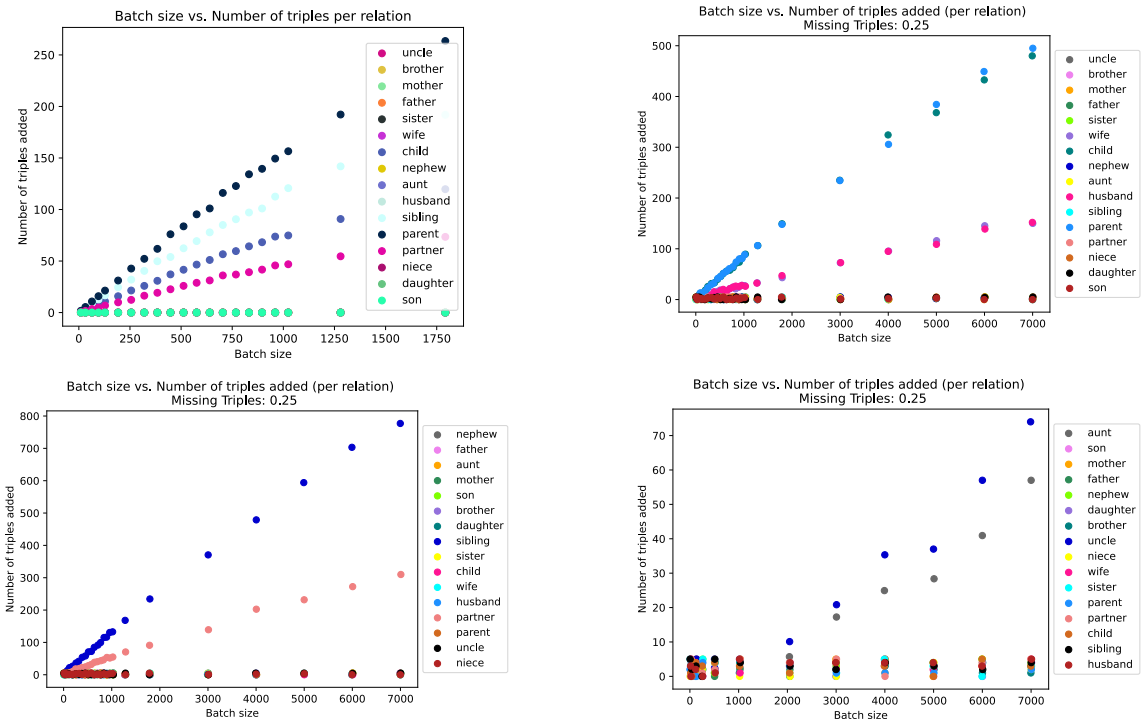


Figure 8.9.: Number of facts added per relation type vs. batch size for the hierarchy (top left), inversion (top right), symmetry (bottom left) and composition (bottom right) rule sets.

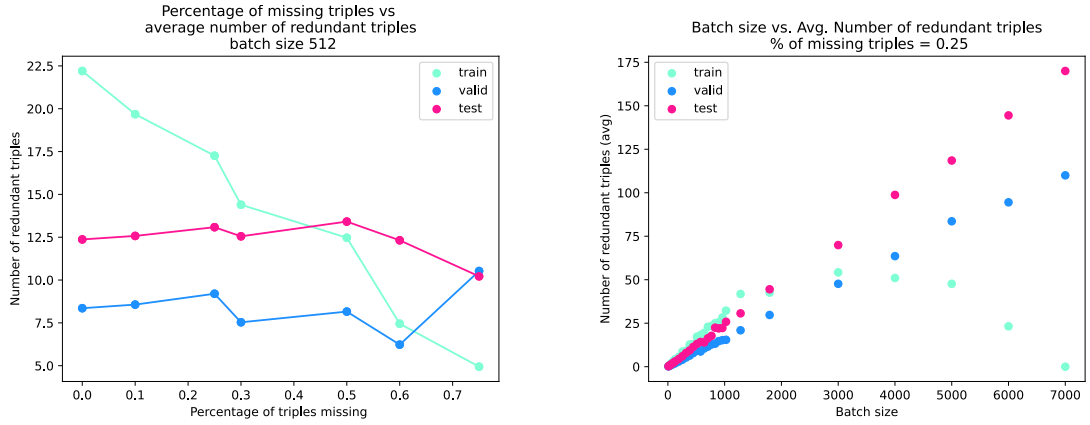


Figure 8.10.: Number of redundant facts vs. percentage of missing facts for batch size 512 for the train, valid and test set (left) and number of redundant facts vs. batch size for the train, valid and test set (right) for the inversion rule set.

Parameter	Value
Batch Size	1024
Early Stopping Delta	0
Early Stopping Patience	50
Epochs	1000
Learning Rate	0.001
Loss	BCE Loss Margin Ranking Loss
Optimizer	Adam
Ratio ρ	1.0
Reasoner Semiring	top-k-proofs
Reasoner k	5

Name	Embedding Dimension	Number of Parameters
BoxE	256	1,551,896
DistMult	50	150,950
QuatE	100	1,207,600
RGCN	500	6,010,692
RotatE	200	1,207,600
TransE	50	138,050
TransH	50	151,550

Table 8.3.: The parameters (left), embedding dimensions and number of parameters (right) per knowledge graph embedding method.

8.3.5. Positive Reasoning (R#2)

The next step is to evaluate whether positive reasoning is useful for training knowledge graph embeddings. The reasoner can make implicit facts explicit by reasoning using the rules. The inferred facts add valuable information to each training batch. The idea is that they will help the model to learn hidden patterns in the dataset.

In the experiments, the knowledge graph embedding methods TransE, TransR, TransH, RotatE, BoxE, DistMult, RGCN and QuatE are used. The set of hyperparameters is shown in Table 8.3 and is based on the set of hyperparameters proposed in [9]. The RGCN has two hidden layers, a dropout rate of 0.4 and the DistMult score function. The Scallop tool [95] is used as implementation of the reasoning engine in RuleKGE.

The initial training set is assumed to represent a closed world where explicit facts represent all positive facts and unknown facts are true negative facts. A percentage of the facts $\gamma \in \{0.1, 0.25, 0.5\}$ are randomly deleted to simulate the incompleteness of the graph. Then, the respective knowledge graph embeddings with and without reasoning are trained on the incomplete training set. The Margin Ranking Loss is used for the models TransE, TransH, RotatE and BoxE and the binary cross entropy loss (BCE Loss) for DistMult, RGCN and QuatE [9]. The rank-based metric Hits@k with $k = 10$ is used to evaluate the performance of the models. The experiments are run 30 times, and it is reported whether the Hits@10 on the test set is significantly higher for runs *with reasoning*. A significance level of $\alpha = 0.05$ is chosen. The hypotheses are formulated as

$$\begin{aligned} H0 &: \text{Hits@10}_{reason} \leq \text{Hits@10}_{noreason} \\ H1 &: \text{Hits@10}_{reason} > \text{Hits@10}_{noreason}. \end{aligned} \tag{8.18}$$

Results. The results for RuleKGE with the symmetry, inversion, hierarchy and composition rule sets are shown in the Tables 8.5 to 8.11.

Before evaluating the effectiveness of the reasoner, it is evident that all knowledge graph embedding models perform worse as the amount of missing data increases. It is observed

Model	missing facts	RuleKGE	no reasoning	delta	significance
TransE	10%	0.1249	0.0777	0.0472	✓
	25%	0.0833	0.0479	0.0354	✓
	50%	0.0300	0.0178	0.0122	✓
TransH	10%	0.5399	0.4984	0.0415	✓
	25%	0.5246	0.5039	0.0208	✓
	50%	0.4730	0.4653	0.0077	✓
RotatE	10%	0.4202	0.1954	0.2248	✓
	25%	0.2204	0.0663	0.1542	✓
	50%	0.0535	0.0221	0.0314	✓
BoxE	10%	0.4542	0.4449	0.0092	✓
	25%	0.4090	0.3974	0.0117	✓
	50%	0.2844	0.2731	0.0113	✓
DistMult	10%	0.1612	0.1520	0.0092	✗
	25%	0.1283	0.1160	0.0123	✓
	50%	0.0661	0.0537	0.0124	✓
RGCN	10%	0.1602	0.1457	0.0145	✓
	25%	0.0923	0.0733	0.0191	✓
	50%	0.0275	0.0256	0.0019	✓
QuatE	10%	0.6153	0.6142	0.0010	✗
	25%	0.5512	0.5415	0.0097	✓
	50%	0.3694	0.3316	0.0378	✓

Table 8.5.: Link Hits@10 for knowledge graph embeddings trained with positive reasoning on the *symmetry* rule set.

that the Hits@10 for 10% missing data is higher than for 25% missing data and higher than for 50% missing data. The general performance of the models regarding $Hits@K$ also differs. While TransE achieves an average $Hits@K$ of 0.1035 on the test set when no data is missing, QuatE achieves an average $Hits@K$ of 0.6508 on the test set, see Table 8.13.

Regarding the positive reasoning, the results for the *symmetry* rule set is presented in Table 8.5. Training with reasoning significantly improves the performance across all levels of missing data for the TransE, TransH, RotatE, BoxE, and RGCN models. For QuatE and DistMult, reasoning is only effective when 25% and 50% of the training data are missing. One reason for this may be that when a higher percentage of data is missing, also more potential extentional facts are missing, which decreases the impact of the rules. This may result in fewer inferred facts that augment the graph. For the *inversion* rule set, as shown in Table 8.9, significant improvements are observed across all percentages of missing data and all models, except for TransH. The results for the *hierarchy* rule set are presented in Table 8.7. Again, significant improvements are observed for all percentages of missing data and all models, with the exception of TransH. For the *composition* rule set in Table 8.11, significant improvements are found for all experiments with RotatE, BoxE and QuatE. However, the RuleKGE does not lead to significant improvements for RGCN, DistMult and TransH. One reason for this may be that DistMult is unable to capture the composition

Model	missing facts	RuleKGE	no reasoning	delta	significance
TransE	10%	0.1058	0.0777	0.0282	✓
	25%	0.0733	0.0479	0.0254	✓
	50%	0.0313	0.0178	0.0135	✓
TransH	10%	0.4792	0.4984	-0.0192	✗
	25%	0.4546	0.5039	-0.0492	✗
	50%	0.3608	0.4653	-0.1045	✗
RotatE	10%	0.4014	0.1954	0.2060	✓
	25%	0.2003	0.0663	0.1340	✓
	50%	0.0454	0.0221	0.0233	✓
BoxE	10%	0.4533	0.4449	0.0084	✓
	25%	0.4081	0.3974	0.0108	✓
	50%	0.2951	0.2731	0.0220	✓
DistMult	10%	0.2247	0.1520	0.0727	✓
	25%	0.1899	0.1160	0.0739	✓
	50%	0.1278	0.0537	0.0741	✓
RGCN	10%	0.1747	0.1457	0.0290	✓
	25%	0.1125	0.0733	0.0392	✓
	50%	0.0335	0.0256	0.0079	✓
QuatE	10%	0.6302	0.6198	0.0104	✓
	25%	0.5667	0.5459	0.0208	✓
	50%	0.3777	0.3339	0.0438	✓

Table 8.7.: Hits@10 for knowledge graph embeddings trained with positive reasoning on the *hierarchy* rule set.

pattern. RGCN uses DistMult as a decoder and is also affected by this limitation. For TransE, significant improvements can only be achieved when 10% of the data is missing.

8.3.6. Negative Reasoning (R#3)

Instead of generating negative facts by uniform negative sampling [26], *negative reasoning* is applied to generate negative facts from negative rules. To evaluate the effectiveness of negative reasoning in the training loop, the antisymmetry and mutual exclusion rule sets in Figure 8.7 are considered. The evaluation is implemented as in the experiments for R#2. In contrast, no positive facts are deleted from the training set, since the negative rules are only used for obtaining negative facts.

The results for the antisymmetry rule set are shown in Table 8.13. The antisymmetry rule set contains 12 rules. The number of negative facts generated by the negative reasoner is not sufficient to meet the required ratio $\rho = 1$ of positive and negative facts. For this reason, additional negative facts are generated with uniform negative sampling, see lines 12 to 14 of Algorithm 3. No significant advantage is found for any of the knowledge graph embeddings tested by generating negative facts with the antisymmetry rule set.

Model	missing facts	RuleKGE	no reasoning	delta	significance
TransE	10%	0.1140	0.0775	0.0365	✓
	25%	0.0808	0.0478	0.0330	✓
	50%	0.0338	0.0175	0.0164	✓
TransH	10%	0.4294	0.4984	-0.0690	✗
	25%	0.4032	0.5039	-0.1007	✗
	50%	0.3439	0.4653	-0.1214	✗
RotatE	10%	0.4381	0.1954	0.2426	✓
	25%	0.1864	0.0663	0.1202	✓
	50%	0.0517	0.0221	0.0296	✓
BoxE	10%	0.4558	0.4449	0.0109	✓
	25%	0.4077	0.3974	0.0103	✓
	50%	0.2793	0.2731	0.0062	✓
DistMult	10%	0.2170	0.1520	0.0650	✓
	25%	0.1770	0.1160	0.0610	✓
	50%	0.1008	0.0537	0.0472	✓
RGCN	10%	0.1698	0.1457	0.0242	✓
	25%	0.0909	0.0733	0.0176	✓
	50%	0.0331	0.0256	0.0075	✓
QuatE	10%	0.6428	0.6198	0.0231	✓
	25%	0.5737	0.5459	0.0279	✓
	50%	0.3575	0.3339	0.0236	✓

Table 8.9.: Hits@10 for knowledge graph embeddings trained with positive reasoning on the *inversion* rule set.

The results for negative reasoning on the mutual exclusion programme are presented in Table 8.15. The mutual exclusion rule set contains 144 rules. In this case, the number of generated negative facts exceeds the number of facts required, so only remaining number of facts is sampled from the set of inferred negative facts, as shown in lines 16 and 17 of the Algorithm 3. As before, there is no significant advantage for any of the model with RuleKGE. On the contrary, negative reasoning tends to degrade performance compared to random head or tail corruption.

The negative reasoning process used in these experiments might introduce an excessive number of negative facts under certain relations apparent in the rules. As a result, there may be far fewer negative facts that are unrelated for *any* relation. This might result in a model that did not learn to distinguish these obviously negative facts from positive facts. As the evaluation procedure for knowledge graph embeddings only considers the ranking of the scores of positive and negative facts, this may affect the performance evaluation. This performance evaluation does not account for the severity of false ranks, whether they result from predicting an incorrect relation between genuinely related entities or predicting a relation between unrelated entities.

Model	missing facts	RuleKGE	no reasoning	delta	significance
TransE	10%	0.0409	0.0359	0.0050	✓
	25%	0.0233	0.0223	0.0010	✗
	50%	0.0096	0.0094	0.0002	✗
TransH	10%	0.5362	0.5388	-0.0026	✗
	25%	0.5005	0.4964	0.0041	✗
	50%	0.3717	0.3662	0.0055	✗
RotatE	10%	0.0938	0.0809	0.0129	✓
	25%	0.0477	0.0426	0.0050	✓
	50%	0.0200	0.0193	0.0007	✓
BoxE	10%	0.4532	0.4369	0.0163	✓
	25%	0.3701	0.3549	0.0151	✓
	50%	0.1309	0.1243	0.0066	✓
DistMult	10%	0.0797	0.0747	0.0050	✗
	25%	0.0504	0.0479	0.0025	✗
	50%	0.0077	0.0079	-0.0001	✗
RGCN	10%	0.0399	0.0380	0.0018	✗
	25%	0.0236	0.0231	0.0005	✗
	50%	0.0170	0.0167	0.0004	✗
QuatE	10%	0.6129	0.5873	0.0256	✓
	25%	0.4778	0.4469	0.0310	✓
	50%	0.2247	0.2119	0.0129	✓

Table 8.11.: Link Prediction results (Hits@10) with knowledge graph embeddings trained with positive reasoning on the *composition* rule set.

Model	RuleKGE	no reasoning	delta	significance
TransE	0.0186	0.1035	-0.0849	✗
TransH	0.0253	0.4928	-0.4675	✗
RotatE	0.3796	0.4005	-0.0209	✗
BoxE	0.2461	0.4696	-0.2235	✗
DistMult	0.1116	0.1734	-0.0618	✗
RGCN	0.1047	0.2084	-0.1038	✗
QuatE	0.6556	0.6508	0.0048	✗

Table 8.13.: Link prediction results (Hits@10) for knowledge graph embedding training with negative reasoning on the antisymmetry rule set.

Model	RuleKGE	no reasoning	delta	significance
TransE	0.0039	0.1035	-0.0995	✗
TransH	0.0113	0.4928	-0.4815	✗
RotatE	0.0031	0.4005	-0.3974	✗
BoxE	0.0097	0.4696	-0.4599	✗
DistMult	0.0950	0.1734	-0.0784	✗
RGCN	0.0151	0.2084	-0.1933	✗
QuatE	0.1140	0.6508	-0.5368	✗

Table 8.15.: Link prediction results (Hits@10) of knowledge graph embedding training with negative reasoning on the mutual exclusion rule set.

Model	RuleKGE	no reasoning	delta	significance
TransE	0.0949	0.0208	0.0741	✓
TransH	0.4814	0.4688	0.0126	✗
RotatE	0.4091	0.0630	0.3461	✓
BoxE	0.4650	0.3935	0.0714	✓
DistMult	0.2289	0.0604	0.1684	✓
RGCN	0.0510	0.0295	0.0215	✓
QuatE	0.6684	0.6508	0.0175	✓

Table 8.17.: Link prediction results (Hits@10) with *zero-shot learning*.

8.3.7. Zero-shot Learning (R#4)

It was shown that the positive reasoning of RuleKGE can be used to infer implicit facts and improve the training of knowledge graph embedding models. Here, we investigate whether RuleKGE is effective in *zero-shot learning* scenarios. Zero-shot learning is defined as the task of predicting classes or relations that do not occur in the training set [34]. These relations are known to exist, but no single fact under these relations is observed during training. Such relations are called *unseen relations*:

$$\{(\mathcal{E} \times \mathcal{R} \times \mathcal{R}) | \mathcal{R} = r\} \cup \mathcal{F}^{+}_{train} = \emptyset. \tag{8.19}$$

In contrast, the relations that occur in the training set are called *seen relations*.

In the experiments in for R#4, all facts for unseen relations are deleted from the training set to simulate the zero-shot learning scenario. Here, the unseen relations are {`parent`, `sibling`, `partner`, `child`}. All facts under these relations are removed from the training set.

However, the reasoner contains knowledge that can be useful in inferring the missing information. The hierarchy rule set shown in Figure 8.6 is taken into account. The results

for the zero-shot learning experiments are shown in Table 8.17. The positive reasoning in the zero-shot scenario leads to significant improvements for all models, except TransH.

8.3.8. Reasoning via an intermediate concept (R#5)

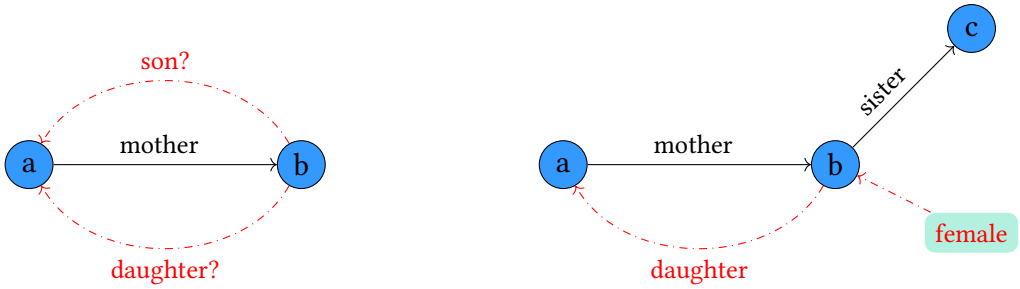


Figure 8.11.: Reasoning with an intermediate concept.

We also test whether reasoning with *intermediate concepts* helps to learn more meaningful knowledge graph embeddings for link prediction. The relations of an intermediate concept do not exist explicitly in the training or test graph. An intermediate concept is a relation $r \notin \mathcal{R}$ that does not belong to the initial set of relations, but can be extracted through reasoning and help to infer facts under known relations in $\mathcal{R}$. Intermediate concepts are visualised in Figure 8.11.

Example 8.3.1 (Intermediate Concepts). Given a rule `mother(x,y)`←`child(x,y)`, the fact `mother(a,b)` is observed. From this, it can only be inferred that `b` must be the child of `a`, but it is unclear whether `b` is the son or the daughter of `a`. However, the intermediate concept of gender `female` and `male` is introduced. The gender is inferred with rules like `female(x)`← `sister(x,_)`. If `x` is the sister of any other entity, then `x` must be female.

This section explores the use of intermediate concepts in RuleKGE. A set of rules is defined in Figure 8.12. The concepts `female` and `male` are intermediate concepts, while the other relations are apparent in the graph. As in the previous dimensions, a proportion of the training facts $\gamma \in \{10\%, 25\%, 50\%\}$ are randomly dropped to simulate the incompleteness of the training graph. Figure 8.13 shows that the reasoner infers unary facts for the genders as intermediate concepts. However, these facts are only relevant at the reasoning stage to infer other facts under relations in $\mathcal{R}$. The larger the batch size, the more entities receive gender information. The results for RuleKGE with intermediate concepts are shown in Table 8.19. It is observed that reasoning via the intermediate gender concepts on the Family dataset leads to significant improvements for all rates of missing data for the TransE, RotatE, BoxE, DistMult and QuatE models. For TransH, significant improvements are observed only when 10% of the facts are missing. For RGCN, significant improvements are evident for 10% and 50% of facts missing, but not for 25% of facts missing.

```
# intermediate concept
female(x)←daughter(x, _) or sister(x, _) or mother(x, _) or aunt(x, _) or wife(x, _)
male(x)←son(x, _) or brother(x, _) or father(x, _) or uncle(x, _) or husband(x, _)
.
aunt(y,x)←niece(x,y) and female(y)
aunt(y,x)←nephew(x,y) and female(y)
uncle(y,x)←niece(x,y) and male(y)
uncle(y,x)←nephew(x,y) and male(y)
mother(y,x)←child(x,y) and female(y)
father(y,x)←child(x,y) and male(y)
daughter(y,x)←parent(x,y) and female(y)
son(y,x)←parent(x,y) and male(y)
sister(y,x)←sibling(x,y) and female(y)
brother(y,x)←sibling(x,y) and male(y)
```

Figure 8.12.: Program Intermediate concept

8.4. Limitations

Despite these promising results, RuleKGE has a number of limitations.

Predictable inference. As RuleKGE belongs to the category of knowledge-driven graph augmentation described in Section 4.3.2, the reasoner contributes knowledge in the form of additional training data. While this allows the model to more successfully learn patterns from this data, it does not necessarily lead to predictable and reliable inference with respect to the rules. Furthermore, even if facts from complex patterns is introduced at the reasoning stage, the inductive capacity and expressiveness of the chosen knowledge graph embedding method may limit the extent to which the information is captured and made relevant at the inference stage.

Scalability. The data volume increases due to the inferred facts by RuleKGE. This can make training more resource intensive and lead to scalability problems. The complexity of Datalog reasoning can be exponential. However, the number of facts given to the reasoner in RuleKGE is controlled by the batch size since reasoning is applied only to the facts in a batch and not the entire graph. An interesting idea is to apply reasoning in the neighbourhood of the nodes contained in the batch. This may further reduce the increase of data volume by the reasoning process.

Noise and Incompleteness. RuleKGE is able to deal with incomplete data, but not with noise. A basic assumption of the reasoning process is that the facts must be a model of the rules. Furthermore, the reasoning process is monotone, i.e. new facts can be added, but once inferred facts cannot be removed. These are strong assumptions, given that real-world knowledge graphs may contain noisy information. If the explicit facts contain

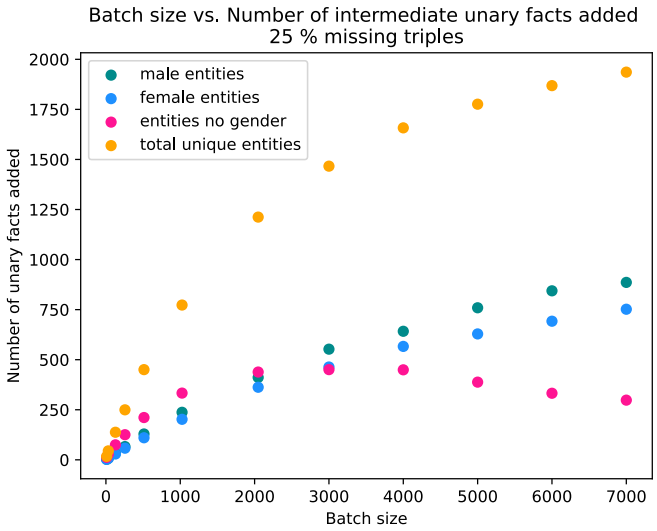


Figure 8.13.: Implicit unary facts inferred for the intermediate concept male and female vs. batch size.

Model	missing facts	RuleKGE	no reasoning	delta	significance
TransE	10%	0.1184	0.0777	0.0407	✓
	25%	0.0737	0.0479	0.0258	✓
	50%	0.0242	0.0178	0.0064	✓
TransH	10%	0.5313	0.4984	0.0329	✓
	25%	0.5249	0.5039	0.0210	✗
	50%	0.4722	0.4653	0.0069	✗
RotatE	10%	0.5274	0.1954	0.3320	✓
	25%	0.3894	0.0663	0.3231	✓
	50%	0.0495	0.0221	0.0274	✓
BoxE	10%	0.4705	0.4449	0.0256	✓
	25%	0.4232	0.3974	0.0259	✓
	50%	0.3033	0.2731	0.0302	✓
DistMult	10%	0.2296	0.1520	0.0777	✓
	25%	0.1812	0.1160	0.0652	✓
	50%	0.0982	0.0537	0.0445	✓
RGCN	10%	0.1590	0.1457	0.0133	✓
	25%	0.0772	0.0733	0.0039	✗
	50%	0.0276	0.0256	0.0020	✓
QuatE	10%	0.6733	0.6198	0.0535	✓
	25%	0.6197	0.5459	0.0739	✓
	50%	0.4481	0.3339	0.1142	✓

Table 8.19.: Link prediction results (Hits@10) for reasoning with intermediate concepts.

errors, the reasoner may propagate incorrect information because there is no way to detect and correct errors.

Furthermore, if the extent to which RuleKGE can help with incomplete data is also limited to the amount of missing data tolerated. If too many facts are missing, there might be not enough extensional facts that allow the reasoner to infer new facts.

Unwarranted Bias. The reasoner in RuleKGE introduces a bias that allows knowledge graph embedding models to better capture certain inference patterns. However, there is a risk that the reasoner may lead to overly biased data. This is particularly relevant for the generation of negative facts where a bias for the relations in the rules is imposed. For example, if inference rules generate a large amount of rules under a certain relation, this relation will be overrepresented. As a future work, normalization methods could provide a solution.

8.5. Conclusion and Outlook

This chapter introduced the neuro-symbolic method RuleKGE, which combines knowledge graph embedding training with a symbolic reasoning component. Given a knowledge graph affected by incompleteness, RuleKGE allows to dynamically infer new positive facts during training and to incorporate them into the loss computation and gradient updates. In this way, implicit positive facts of a rule pattern in the knowledge graph are made explicit and injected into the learned knowledge graph embeddings. Furthermore, RuleKGE supports negative reasoning, which aims at generating negative facts in a more coherent way based on prior knowledge. With these features, RuleKGE belongs to the category of knowledge-aware graph augmentation methods.

The experimental evaluation on the Family dataset with different Datalog programs and knowledge graph embedding methods shows that RuleKGE can be used as a model-agnostic framework to help learn better knowledge graph embeddings in the context of incomplete graphs and especially in zero-shot learning scenarios. However, the use of RuleKGE for negative reasoning was not shown to be effective for learning more meaningful knowledge graph embeddings for link prediction. More extensive experiments on other state-of-the-art benchmarks such as Freebase [25] and Wordnet [149] are an important line of future work. Some future directions with RuleKGE include improving the negative sampling technique to find a good balance between generating true negative facts and exposing the model to different patterns of negative facts. Also, a more integrated combination of the reasoner with probabilistic reasoning and the knowledge graph embedding to be more robust to noise is a future area of work.

With respect to the neuro-symbolic desiderata, RuleKGE is knowledge-aware as it processes prior knowledge in the form of Horn rules in first-order logic. In terms of robustness, the reasoning process of RuleKGE is not robust to noise, as the reasoning process is monotone and the facts are assumed to be a model of the rules, while the knowledge graph embedding process remains robust to noise. However, if the reasoner propagates false

information to incorrectly inferred facts, this could lead to harmful bias in the knowledge graph embedding, ultimately leading to performance degradation.

In terms of interpretability, RuleKGE depends on how many new (and correct) facts can be generated by the reasoner and added to the training set. In consequence, there are still no guarantees at inference.

Regarding scalability, the limiting factor of RuleKGE is the reasoner, which can become a bottleneck for large graphs and complex rules.

9. Conclusion

In this thesis, the application of neuro-symbolic AI techniques to graphs was studied.

9.1. Summary of Contribution

The following aspects were explored. First, in the state-of-the-art part, different neuro-symbolic techniques were studied, focusing on general frameworks as well as their ability to support knowledge graphs. At first sight, these general frameworks seemed to be theoretically capable of capturing binary predicates and applying them to graphs. It turned out they were only be applied to toy examples and very small graphs. These examples often hide the profound limitations of these frameworks when applied to real-world scenario with reasonably large graphs.

The contribution part started with a reproducibility study of Knowledge Enhanced Neural Networks. The method was tested in a graph-specific framework. A step-by-step method of reproducing, replicating and re-evaluating was used to ensure that subsequent extensions are built on a solid ground. General lessons learned were drawn and crucial aspects were proposed on how to improve reproducibility in the field of machine learning.

Then, the method *KeGNN* was proposed that integrates knowledge enhancement layers and graph neural networks. Their usefulness in graph neural networks was investigated on the task of node classification on homogeneous graphs. The experiments provided no evidence for the effectiveness of the knowledge enhancement. One explanation for this observation is that the information in the rules was often redundant with the inductive bias of the GNN. Another issue was related to the benchmarks. State-of-the-art benchmarks were used that turned out to be not representative and expressive enough (homogeneous, limited opportunities for inference).

Subsequently, KeGNN was applied to large graphs, focussing on node classification in homogeneous graphs. Some serious representation issues related to memory requirements were encountered, particularly when training on large graphs. To mitigate this, a sampling mechanism called *restrictive neighbourhood sampling* was proposed to split data into mini-batches based on the graph structure. In this way, the sparse nature of the graphs is better taken into account and the information loss of the batching process is considered. This sampling technique makes knowledge enhancement layers applicable to large graphs.

Moreover, the method *RuleKGE* was introduced, which addresses the task of link prediction on unattributed knowledge graphs. Horn rules were incorporated into the training loop,

along with a positive and negative reasoning step. The goal was to train knowledge graph embeddings in an informed way by using rules from ontologies. RuleKGE was shown to be effective on incomplete graphs in various experimental settings. It turned out to be particularly effective in zero-shot learning scenarios where some relations were unseen in the training graph. However, negative reasoning did not show the expected benefits. One reason for this may be that the negative facts generated by the negative reasoner are too biased towards a particular pattern of negative facts. A future direction with RuleKGE involves improving the negative reasoning technique to allow the model to learn different patterns of negative facts.

9.2. Perspectives and Future Directions

Looking ahead, several open questions and perspectives for future work emerge from this thesis, highlighting both the progress made and the challenges ahead.

Benchmarking. Studies that experimentally compare existing approaches on the same data are lacking. Many works in the state-of-the-art use different datasets or versions. This prevents meaningful comparisons and is a barrier to successful progress in neuro-symbolic AI on graphs. Prior knowledge is often not included in the data and every work comes with its own set of rules. To ensure that evaluations are both realistic and informative, neuro-symbolic benchmarks should provide the data and the prior knowledge.

The provided level of knowledge. A significant difference exists in the expressivity of knowledge supported by several neuro-symbolic methods. Most of the methods deal only with simple rules. Capturing complex rules within the embedding space or within neural methods remains an open question. This includes, for example, multi-hop rules, recursive rules, n-ary predicates, or quantification. While most of the existing approaches focus on the use of such rules in training, their reliability in inference often remains unclear.

Graph symbol grounding problem. General neuro-symbolic frameworks such as LTN, DeepProblog and KENN require significant redesign to be usable for large graphs. They have been built under the closed-world assumption. As a consequence, they rely on dense representations that not only do not scale, particularly with negated facts, and fail to capture unknown relations. A typical example is the Logic Tensor Networks that rely on adjacency matrices to represent relations. It would be more appropriate to rely only on the explicit facts (relation tuples).

Bibliography

[1] AAAI. *Reproducibility Checklist*. https://aaai.org/conference/aaai/aaai-23/reproducibility-checklist/. Accessed: 2023-01-04. 2023.

[2] Ralph Abboud and İsmail İlkan Ceylan. *Node Classification Meets Link Prediction on Knowledge Graphs*. 2021. DOI: 10.48550/ARXIV.2106.07297. URL: https://arxiv.org/abs/2106.07297.

[3] Ralph Abboud et al. "BoxE: A Box Embedding Model for Knowledge Base Completion". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 9649–9661. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/6dbbe6abe5f14af882ff977fc3f35501-Paper.pdf.

[4] Ralph Abboud et al. "The Surprising Power of Graph Neural Networks with Random Node Initialization". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Zhi-Hua Zhou. Main Track. International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 2112–2118. DOI: 10.24963/ijcai.2021/291. URL: https://doi.org/10.24963/ijcai.2021/291.

[5] Serge Abiteboul, Richard Hull, and Victor Vianu. "Foundations of Databases". In: Jan. 1995. ISBN: 0-201-53771-0.

[6] ACM. *Association for Computing Machinery. Artifact review and badging, 1.1*. https://www.acm.org/publications/policies/artifact-review-badging. Accessed: 2023-01-04. 2016.

[7] Kian Ahrabian et al. "Structure Aware Negative Sampling in Knowledge Graphs". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Bonnie Webber et al. Online: Association for Computational Linguistics, Nov. 2020, pp. 6093–6101. DOI: 10.18653/v1/2020.emnlp-main.492. URL: https://aclanthology.org/2020.emnlp-main.492.

[8] Saeed S. Alahmari et al. "Challenges for the Repeatability of Deep Learning Models". In: *IEEE Access* 8 (2020), pp. 211860–211868. DOI: 10.1109/ACCESS.2020.3039833.

[9] Mehdi Ali et al. "Bringing Light Into the Dark: A Large-Scale Evaluation of Knowledge Graph Embedding Models Under a Unified Framework". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.12 (Dec. 2022), pp. 8825–8845. ISSN: 1939-3539. DOI: 10.1109/tpami.2021.3124805. URL: http://dx.doi.org/10.1109/TPAMI.2021.3124805.

[10] Ankit Ankit et al. "[Re] counterfactual generative networks". In: *ReScience C* 8.#2 (2 2022). DOI: 10.5281/zenodo.6574625.

[11] A. Artale et al. "The DL-Lite Family and Relations". In: *Journal of Artificial Intelligence Research* 36 (Oct. 2009), pp. 1–69. ISSN: 1076-9757. DOI: 10.1613/jair.2820. URL: http://dx.doi.org/10.1613/jair.2820.

[12] Franz Baader et al. *An Introduction to Description Logic*. Cambridge University Press, 2017. DOI: 10.1017/9781139025355.

[13] Sebastian Bader and Pascal Hitzler. *Dimensions of Neural-symbolic Integration - A Structured Survey*. 2005. arXiv: cs/0511042 [cs.AI].

[14] Samy Badreddine et al. "Logic Tensor Networks". In: *Artificial Intelligence* 303 (Feb. 2022), p. 103649. DOI: 10.1016/j.artint.2021.103649. URL: https://doi.org/10.1016%2Fj.artint.2021.103649.

[15] Pradeep Kr. Banerjee et al. *Oversquashing in GNNs through the lens of information contraction and graph expansion*. 2022. arXiv: 2208.03471 [cs.LG].

[16] Chitta Baral and Michael Gelfond. "Logic programming and knowledge representation". In: *The Journal of Logic Programming* 19-20 (1994). Special Issue: Ten Years of Logic Programming, pp. 73–148. ISSN: 0743-1066. DOI: https://doi.org/10.1016/0743-1066(94)90025-6. URL: https://www.sciencedirect.com/science/article/pii/0743106694900256.

[17] Yoshua Bengia and Gary Marcus. *Ai debate: the best way forward for AI*. https://montrealartificialintelligence.com/aidebate/. Accessed on 13 December 2021. 2021.

[18] Yoshua Bengio et al. *A Meta-Transfer Objective for Learning to Disentangle Causal Mechanisms*. 2019. arXiv: 1901.10912 [cs.LG].

[19] Tarek R. Besold et al. *Neural-Symbolic Learning and Reasoning: A Survey and Interpretation*. 2017. arXiv: 1711.03902 [cs.AI].

[20] K. Bhatia et al. *The extreme classification repository: Multi-label datasets and code*. 2016. URL: http://manikvarma.org/downloads/XC/XMLRepository.html.

[21] Joanna Biega, Erdal Kuzey, and Fabian Suchanek. "Inside YAGO2s: a transparent information extraction architecture". In: May 2013, pp. 325–328.

[22] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[23] Russa Biswas et al. "Knowledge Graph Embeddings: Open Challenges and Opportunities". In: *Transactions on Graph Data and Knowledge* 1.1 (2023), 4:1–4:32. DOI: 10.4230/TGDK.1.1.4. URL: https://drops.dagstuhl.de/entities/document/10.4230/TGDK.1.1.4.

[24] Wilfredo Blanco et al. "Non-replicability circumstances in a neural network model with Hodgkin-Huxley-type neurons". In: *Journal of computational neuroscience* 48.3 (2020), pp. 357–363. DOI: 10.1007/s10827-020-00748-3.

[25] Kurt Bollacker et al. “Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge”. In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’08. Vancouver, Canada: Association for Computing Machinery, 2008, pp. 1247–1250. ISBN: 9781605581026. DOI: 10.1145/1376616.1376746. URL: https://doi.org/10.1145/1376616.1376746.

[26] Antoine Bordes et al. “Translating Embeddings for Modeling Multi-relational Data”. In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper_files/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf.

[27] Michael M. Bronstein et al. “Geometric Deep Learning: Going beyond Euclidean data”. In: *IEEE Signal Processing Magazine* 34.4 (July 2017), pp. 18–42. ISSN: 1558-0792. DOI: 10.1109/msp.2017.2693418. URL: http://dx.doi.org/10.1109/MSP.2017.2693418.

[28] Andrea Cali, Georg Gottlob, and Thomas Lukasiewicz. “A general datalog-based framework for tractable query answering over ontologies”. In: *Proceedings of the Twenty-Eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. PODS ’09. <conf-loc>, <city>Providence</city>, <state>Rhode Island</state>, <country>USA</country>, </conf-loc>: Association for Computing Machinery, 2009, pp. 77–86. ISBN: 9781605585536. DOI: 10.1145/1559795.1559809. URL: https://doi.org/10.1145/1559795.1559809.

[29] Alison Callahan et al. “Bio2RDF Release 2: Improved Coverage, Interoperability and Provenance of Life Science Linked Data”. In: vol. 7882. May 2013, pp. 200–212. ISBN: 978-3-642-38287-1. DOI: 10.1007/978-3-642-38288-8_14.

[30] Jiahang Cao et al. *Knowledge Graph Embedding: A Survey from the Perspective of Representation Spaces*. 2023. arXiv: 2211.03536 [cs.LG].

[31] Andrew Carlson et al. “Toward an Architecture for Never-Ending Language Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 24.1 (July 2010), pp. 1306–1313. DOI: 10.1609/aaai.v24i1.7519. URL: https://ojs.aaai.org/index.php/AAAI/article/view/7519.

[32] Andrea Cavallo et al. “GCNH: A Simple Method For Representation Learning On Heterophilous Graphs”. In: *2023 International Joint Conference on Neural Networks (IJCNN)*. IEEE, June 2023. DOI: 10.1109/ijcnn54540.2023.10191196. URL: http://dx.doi.org/10.1109/IJCNN54540.2023.10191196.

[33] Boyuan Chen et al. “Towards Training Reproducible Deep Learning Models”. In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 2202–2214. ISBN: 9781450392211. DOI: 10.1145/3510003.3510163. URL: https://doi.org/10.1145/3510003.3510163.

[34] Jiaoyan Chen et al. “Knowledge-aware Zero-Shot Learning: Survey and Perspective”. In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Zhi-Hua Zhou. Survey Track. International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 4366–4373. DOI: 10.24963/ijcai.2021/597. URL: https://doi.org/10.24963/ijcai.2021/597.

[35] Jie Chen, Tengfei Ma, and Cao Xiao. “FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling.” In: *ICLR (Poster)*. OpenReview.net, 2018. URL: http://dblp.uni-trier.de/db/conf/iclr/iclr2018.html#ChenMX18.

[36] Kewei Cheng et al. “UniKER: A Unified Framework for Combining Embedding and Definite Horn Rule Reasoning for Knowledge Graph Inference”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Ed. by Marie-Francine Moens et al. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 9753–9771. DOI: 10.18653/v1/2021.emnlp-main.769. URL: https://aclanthology.org/2021.emnlp-main.769.

[37] Wei-Lin Chiang et al. “Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 257–266. ISBN: 9781450362016. DOI: 10.1145/3292500.3330925. URL: https://doi.org/10.1145/3292500.3330925.

[38] Francois Chollet et al. *Keras*. https://keras.io. Accessed: 2023-01-04. 2015.

[39] Philipp Cimiano and Heiko Paulheim. “Knowledge graph refinement: A survey of approaches and evaluation methods”. In: *Semant. Web* 8.3 (Jan. 2017), pp. 489–508. ISSN: 1570-0844. DOI: 10.3233/SW-160218. URL: https://doi.org/10.3233/SW-160218.

[40] Andrew Cropper, Sebastijan Dumančić, and Stephen H. Muggleton. *Turning 30: New Ideas in Inductive Logic Programming*. 2020. arXiv: 2002.11002 [cs.AI].

[41] J. Cullen and A. Bryman. “The Knowledge Acquisition Bottleneck: Time for Reassessment?” In: *Expert Systems* 5.3 (1988), pp. 216–225. DOI: https://doi.org/10.1111/j.1468-0394.1988.tb00065.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1468-0394.1988.tb00065.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1468-0394.1988.tb00065.x.

[42] Claudia d’Amato, Nicola Quatraro, and Nicola Fanizzi. “Injecting Background Knowledge into Embedding Models for Predictive Tasks on Knowledge Graphs”. In: May 2021, pp. 441–457. ISBN: 978-3-030-77384-7. DOI: 10.1007/978-3-030-77385-4_26.

[43] Yuanfei Dai et al. “A Survey on Knowledge Graph Embedding: Approaches, Applications and Benchmarks”. In: *Electronics* 9.5 (2020). ISSN: 2079-9292. DOI: 10.3390/electronics9050750. URL: https://www.mdpi.com/2079-9292/9/5/750.

[44] Zihang Dai, Lei Li, and Wei Xu. "CFO: Conditional Focused Neural Question Answering with Large-scale Knowledge Bases". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Katrin Erk and Noah A. Smith. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 800–810. DOI: 10.18653/v1/P16-1076. URL: https://aclanthology.org/P16-1076.

[45] Alessandro Daniele and Luciano Serafini. "Knowledge Enhanced Neural Networks". In: *PRICAI 2019: Trends in Artificial Intelligence*. Ed. by Abhaya C. Nayak and Alok Sharma. Cham: Springer International Publishing, 2019, pp. 542–554. ISBN: 978-3-030-29908-8.

[46] Alessandro Daniele and Luciano Serafini. "Knowledge Enhanced Neural Networks for Relational Domains". In: *AIxIA 2022 – Advances in Artificial Intelligence*. Ed. by Agostino Dovier, Angelo Montanari, and Andrea Orlandini. Cham: Springer International Publishing, 2023, pp. 91–109. ISBN: 978-3-031-27181-6.

[47] Alessandro Daniele and Luciano Serafini. "Neural Networks Enhancement through Prior Logical Knowledge". In: *ArXiv* abs/2009.06087 (2020).

[48] Alessandro Daniele and Luciano Serafini. *Neural Networks Enhancement with Logical Knowledge*. https://arxiv.org/abs/2009.06087. 2020. DOI: 10.48550/ARXIV.2009.06087.

[49] Adnan Darwiche. "SDD: a new canonical representation of propositional knowledge bases". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*. IJCAI'11. Barcelona, Catalonia, Spain: AAAI Press, 2011, pp. 819–826. ISBN: 9781577355144.

[50] Luc De Raedt et al. "From statistical relational to neural-symbolic artificial intelligence". In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*. IJCAI'20. Yokohama, Yokohama, Japan, 2021. ISBN: 9780999241165.

[51] Craig DeLancey. *A Concise Introduction to Logic*. Geneseo, NY: Open SUNY Textbooks, 2017.

[52] Lauren Nicole DeLong et al. *Neurosymbolic AI for Reasoning on Graph Structures: A Survey*. https://arxiv.org/abs/2302.07200. 2023. arXiv: 2302.07200 [cs.AI].

[53] Caglar Demir and Axel-Cyrille Ngonga Ngomo. "Neuro-Symbolic Class Expression Learning". In: *The 32nd International Joint Conference on Artificial Intelligence, IJCAI 2023*. 2023. URL: https://papers.dice-research.org/2023/IJCAI_DRILL/public.pdf.

[54] Wenqing Deng et al. "Choice-Driven Contextual Reasoning for Commonsense Question Answering". In: *PRICAI 2022: Trends in Artificial Intelligence: 19th Pacific Rim International Conference on Artificial Intelligence, PRICAI 2022, Shanghai, China, November 10–13, 2022, Proceedings, Part II*. Shangai, China: Springer-Verlag, 2022, pp. 335–346. ISBN: 978-3-031-20864-5. DOI: 10.1007/978-3-031-20865-2_25. URL: https://doi.org/10.1007/978-3-031-20865-2_25.

[55] Tim Dettmers et al. "Convolutional 2D Knowledge Graph Embeddings". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'18/IAAI'18/EAAI'18. New Orleans, Louisiana, USA: AAAI Press, 2018. ISBN: 978-1-57735-800-8.

[56] Michelangelo Diligenti, Marco Gori, and Claudio Saccà. "Semantic-based regularization for learning and inference". In: *Artificial Intelligence* 244 (2017). Combining Constraint Solving with Mining and Learning, pp. 143–165. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2015.08.011. URL: https://www.sciencedirect.com/science/article/pii/S0004370215001344.

[57] Ivan Donadello, Luciano Serafini, and Artur d'Avila Garcez. "Logic Tensor Networks for Semantic Image Interpretation". In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017, pp. 1596–1602. DOI: 10.24963/ijcai.2017/221. URL: https://doi.org/10.24963/ijcai.2017/221.

[58] Christopher Drummond. *Replicability is not reproducibility: nor is it good science*. http://cogprints.org/7691/. CogPrints. Accessed: 2023-01-04. 2009.

[59] Keyu Duan et al. "A Comprehensive Study on Large-Scale Graph Training: Benchmarking and Rethinking". In: *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. 2022.

[60] Stefano Faralli et al. "The ContrastMedium Algorithm: Taxonomy Induction From Noisy Knowledge Graphs With Just A Few Links". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Ed. by Mirella Lapata, Phil Blunsom, and Alexander Koller. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 590–600. URL: https://aclanthology.org/E17-1056.

[61] Matthias Fey and Jan E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*. New Orleans, USA, 2019. URL: https://arxiv.org/abs/1903.02428.

[62] Matthias Fey et al. *GNNAutoScale: Scalable and Expressive Graph Neural Networks via Historical Embeddings*. 2021. DOI: 10.48550/ARXIV.2106.05609. URL: https://arxiv.org/abs/2106.05609.

[63] Daan Fierens et al. "Inference and learning in probabilistic logic programs using weighted Boolean formulas". In: *Theory and Practice of Logic Programming* 15.3 (Apr. 2014), pp. 358–401. ISSN: 1475-3081. DOI: 10.1017/s1471068414000076. URL: http://dx.doi.org/10.1017/S1471068414000076.

[64] Jessica Zosa Forde and Michela Paganini. "The Scientific Method in the Science of Machine Learning". In: *ICLR 2019 workshop, May 6, New Orleans* (2019).

[65] Luis Antonio Galarraga et al. "AMIE: association rule mining under incomplete evidence in ontological knowledge bases". In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW '13. Rio de Janeiro, Brazil: Association for Computing Machinery, 2013, pp. 413–422. ISBN: 9781450320351. DOI: 10.1145/2488388.2488425. URL: https://doi.org/10.1145/2488388.2488425.

[66] Luis Galárraga et al. “Fast rule mining in ontological knowledge bases with AMIE+”. In: *The VLDB Journal* 24.6 (Dec. 2015), pp. 707–730. ISSN: 1066-8888. DOI: 10.1007/s00778-015-0394-1. URL: https://doi.org/10.1007/s00778-015-0394-1.

[67] Artur d’Avila Garcez and Luis C. Lamb. *Neurosymbolic AI: The 3rd Wave*. 2020. arXiv: 2012.05876 [cs.AI].

[68] Artur d’Avila Garcez et al. *Neural-Symbolic Computing: An Effective Methodology for Principled Integration of Machine Learning and Reasoning*. 2019. arXiv: 1905.06088 [cs.AI].

[69] Marta Garnelo and Murray Shanahan. “Reconciling deep learning with symbolic artificial intelligence: representing objects and relations”. In: *Current Opinion in Behavioral Sciences* 29 (2019). Artificial Intelligence, pp. 17–23. ISSN: 2352-1546. DOI: https://doi.org/10.1016/j.cobeha.2018.12.010. URL: https://www.sciencedirect.com/science/article/pii/S2352154618301943.

[70] Aryo Pradipta Gema et al. *Knowledge Graph Embeddings in the Biomedical Domain: Are They Useful? A Look at Link Prediction, Rule Learning, and Downstream Polypharmacy Tasks*. 2023. arXiv: 2305.19979 [cs.LG].

[71] Genet Asefa Gesese, Russa Biswas, and Harald Sack. “A Comprehensive Survey of Knowledge Graph Embeddings with Literals: Techniques and Applications”. In: *DL4KG@ESWC*. 2019. URL: https://api.semanticscholar.org/CorpusID:186206127.

[72] Steven Goodman, Daniele Fanelli, and John Ioannidis. “What does research reproducibility mean?” In: *Science translational medicine* 8.341 (2016). DOI: 10.1126/scitranslmed.aaf5027.

[73] Eleonora Grilli et al. “Knowledge Enhanced Neural Networks for Point Cloud Semantic Segmentation”. In: vol. 15. 10. 2023. DOI: 10.3390/rs15102590. URL: https://www.mdpi.com/2072-4292/15/10/2590.

[74] Odd Erik Gundersen. “The Reproducibility Crisis Is Real”. In: *AI Magazine* 41.3 (Sept. 2020), pp. 103–106. DOI: 10.1609/aimag.v41i3.5318. URL: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/5318.

[75] Odd Erik Gundersen, Yolanda Gil, and David W. Aha. “On Reproducible AI: Towards Reproducible Research, Open Science, and Digital Scholarship in AI Publications”. In: *AI Magazine* 39.3 (Sept. 2018), pp. 56–68. DOI: 10.1609/aimag.v39i3.2816. URL: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2816.

[76] Odd Erik Gundersen and Sigbjørn Kjensmo. “State of the Art: Reproducibility in Artificial Intelligence”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 2018). DOI: 10.1609/aaai.v32i1.11503. URL: https://ojs.aaai.org/index.php/AAAI/article/view/11503.

[77] Shu Guo et al. "Jointly Embedding Knowledge Graphs and Logical Rules". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 192–202. DOI: 10.18653/v1/D16-1019. URL: https://aclanthology.org/D16-1019.

[78] Shu Guo et al. "Knowledge Graph Embedding With Iterative Guidance From Soft Rules". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1 (Apr. 2018). DOI: 10.1609/aaai.v32i1.11918. URL: https://ojs.aaai.org/index.php/AAAI/article/view/11918.

[79] Victor Gutierrez-Basulto and Steven Schockaert. *From Knowledge Graph Embedding to Ontology Embedding? An Analysis of the Compatibility between Vector Space Representations and Rules*. 2018. arXiv: 1805.10461 [cs.AI].

[80] William L. Hamilton, Rex Ying, and Jure Leskovec. *Inductive Representation Learning on Large Graphs*. 2017. DOI: 10.48550/ARXIV.1706.02216. URL: https://arxiv.org/abs/1706.02216.

[81] Chi Han et al. *Logical Entity Representation in Knowledge-Graphs for Differentiable Rule Learning*. 2023. arXiv: 2305.12738 [cs.AI].

[82] Junheng Hao et al. "Universal Representation Learning of Knowledge Bases by Jointly Embedding Instances and Ontological Concepts". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 1709–1719. ISBN: 9781450362016. DOI: 10.1145/3292500.3330838. URL: https://doi.org/10.1145/3292500.3330838.

[83] Frank van Harmelen. *The K in neuro-symbolic stands for knowledge*. https://de.slideshare.net/slideshow/the-k-in-neurosymbolic-stands-for-knowledge/258584872. June 2023.

[84] Stevan Harnad. "The symbol grounding problem". In: *Physica D: Nonlinear Phenomena* 42.1–3 (June 1990), pp. 335–346. ISSN: 0167-2789. DOI: 10.1016/0167-2789(90)90087-6. URL: http://dx.doi.org/10.1016/0167-2789(90)90087-6.

[85] L Vivek Harsha Vardhan, Guo Jia, and Stanley Kok. "Probabilistic Logic Graph Attention Networks for Reasoning". In: *Companion Proceedings of the Web Conference 2020*. WWW '20. Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 669–673. ISBN: 9781450370240. DOI: 10.1145/3366424.3391265. URL: https://doi.org/10.1145/3366424.3391265.

[86] Qi He, Bee-Chung Chen, and Deepak Agarwal. *Building The LinkedIn Knowledge Graph. LinkedIn Blog*. Slides at https://speakerdeck.com/emeij/understanding-news-using-the-bloomberg-knowledge-graph. 2016.

[87] Peter Henderson et al. "Deep Reinforcement Learning That Matters". In: *Proceedings of AAAI'18/IAAI'18/EAAI'18 conferences in Artificial Intelligence*. New Orleans, Louisiana, USA: AAAI Press, 2018. ISBN: 978-1-57735-800-8.

[88] P. Hitzler and M.K. Sarker. *Neuro-symbolic Artificial Intelligence: The State of the Art*. Frontiers in artificial intelligence and applications. IOS Press, 2022. ISBN: 9781643682440. URL: https://books.google.de/books?id=jnL0zgEACAAJ.

[89] Vinh Thinh Ho et al. "Rule Learning from Knowledge Graphs Guided by Embedding Models". In: *The Semantic Web – ISWC 2018*. Ed. by Denny Vrandecic et al. Cham: Springer International Publishing, 2018, pp. 72–90. ISBN: 978-3-030-00671-6.

[90] Sepp Hochreiter. "Toward a broad AI". In: *Commun. ACM* 65.4 (Mar. 2022), pp. 56–57. ISSN: 0001-0782. DOI: 10.1145/3512715. URL: https://doi.org/10.1145/3512715.

[91] Aidan Hogan et al. "Knowledge Graphs". In: *ACM Computing Surveys* 54.4 (July 2021), pp. 1–37. ISSN: 1557-7341. DOI: 10.1145/3447772. URL: http://dx.doi.org/10.1145/3447772.

[92] Alfred Horn. "On Sentences Which Are True of Direct Unions of Algebras". In: *Journal of Symbolic Logic* 16.3 (1951), pp. 216–217. DOI: 10.2307/2266412.

[93] Weihua Hu et al. "Open Graph Benchmark: Datasets for Machine Learning on Graphs". In: *CoRR* abs/2005.00687 (2020). arXiv: 2005.00687. URL: https://arxiv.org/abs/2005.00687.

[94] Yuwei Hu et al. "FeatGraph: A Flexible and Efficient Backend for Graph Neural Network Systems". In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2020, pp. 1–13. DOI: 10.1109/SC41405.2020.00075.

[95] Jiani Huang et al. "Scallop: From Probabilistic Deductive Databases to Scalable Differentiable Reasoning". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 25134–25145. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/d367eef13f90793bd8121e2f675f0dc2-Paper.pdf.

[96] Cuiying Huo et al. "T2-GNN: graph neural networks for graphs with incomplete features and structure via teacher-student distillation". In: *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*. AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. ISBN: 978-1-57735-880-0. DOI: 10.1609/aaai.v37i4.25553. URL: https://doi.org/10.1609/aaai.v37i4.25553.

[97] ICLR. *ICLR Reproducibility Challenge. Second Edition, 2019*. https://www.cs.mcgill.ca/~jpineau/ICLR2019-ReproducibilityChallenge.html. Accessed: 2023-01-04. 2019.

[98] ICML. *Paper Guidelines*. https://icml.cc/Conferences/2023/PaperGuidelines. Accessed: 2023-01-04. 2023.

[99] IJCAI. *Reproducibility guideline*. https://ijcai-22.org/reproducibility/. Accessed: 2023-01-04. 2022.

[100] Eleni Ilkou and Maria Koutraki. "Symbolic Vs Sub-symbolic AI Methods: Friends or Enemies?" In: *CIKM (Workshops)*. Ed. by Stefan Conrad and Ilaria Tiddi. Vol. 2699. CEUR Workshop Proceedings. CEUR-WS.org, 2020. URL: http://dblp.uni-trier.de/db/conf/cikm/cikm2020w.html#IlkouK20.

[101] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. DOI: 10.48550/ARXIV.1502.03167. URL: https://arxiv.org/abs/1502.03167.

[102] Nitisha Jain et al. "Improving Knowledge Graph Embeddings with Ontological Reasoning". In: *The Semantic Web – ISWC 2021: 20th International Semantic Web Conference, ISWC 2021, Virtual Event, October 24–28, 2021, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2021, pp. 410–426. ISBN: 978-3-030-88360-7. DOI: 10.1007/978-3-030-88361-4_24. URL: https://doi.org/10.1007/978-3-030-88361-4_24.

[103] Apache Jena. *A free and open source Java framework for building Semantic Web and Linked Data applications*. https://jena.apache.org/. The Apache Software Foundation, Licensed under the Apache Livence, Version 2.0. 2024.

[104] Shaoxiong Ji et al. "A Survey on Knowledge Graphs: Representation, Acquisition, and Applications". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.2 (Feb. 2022), pp. 494–514. ISSN: 2162-2388. DOI: 10.1109/tnnls.2021.3070843. URL: http://dx.doi.org/10.1109/TNNLS.2021.3070843.

[105] Daniel Kahneman. *Thinking, fast and slow*. New York: Farrar, Straus and Giroux, 2011. ISBN: 9780374275631 0374275637. URL: https://www.amazon.de/Thinking-Fast-Slow-Daniel-Kahneman/dp/0374275637/ref=wl_it_dp_o_pdT1_nS_nC?ie=UTF8&colid=151193SNGKJT9&coliid=I3OCESLZCVDFL7.

[106] Zoi Kaoudi, Abelardo Carlos Martinez Lorenzo, and Volker Markl. *Towards Loosely-Coupling Knowledge Graph Embeddings and Ontology-based Reasoning*. 2022. arXiv: 2202.03173 [cs.AI].

[107] Henry A. Kautz. *The third AI summer: AAAI Robert S. Engelmore Memorial Lecture*. https://onlinelibrary.wiley.com/doi/10.1002/aaai.12036. 2022. DOI: https://doi.org/10.1002/aaai.12036.

[108] Seyed Mehran Kazemi and David Poole. "SimplE Embedding for Link Prediction in Knowledge Graphs". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/b2ab001909a8a6f04b51920306046ce5-Paper.pdf.

[109] Seyed Mehran Kazemi et al. "Representation learning for dynamic graphs: a survey". In: *J. Mach. Learn. Res.* 21.1 (Jan. 2020). ISSN: 1532-4435.

[110] Mishal Kazmi, Peter Schüller, and Yücel Saygın. "Improving scalability of inductive logic programming via pruning and best-effort optimisation". In: *Expert Systems with Applications* 87 (Nov. 2017), pp. 291–303. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2017.06.013. URL: http://dx.doi.org/10.1016/j.eswa.2017.06.013.

[111] Henry J Kelley. “Gradient theory of optimal flight paths”. In: *Ars Journal* 30.10 (1960), pp. 947–954.

[112] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. “An Algebraic Prolog for Reasoning about Possible Worlds”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 25.1 (Aug. 2011), pp. 209–214. DOI: 10.1609/aaai.v25i1.7852. URL: https://ojs.aaai.org/index.php/AAAI/article/view/7852.

[113] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[114] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *Proceedings of the 5th International Conference on Learning Representations* (ICLR). ICLR ’17. Palais des Congrès Neptune, Toulon, France, 2017. URL: https://openreview.net/forum?id=SJU4ayYgl.

[115] Thomas N. Kipf and Max Welling. “Semi-Supervised Classification with Graph Convolutional Networks”. In: *ICLR*. 2017.

[116] E.P. Klement, R. Mesiar, and E. Pap. *Triangular Norms*. Trends in Logic. Springer Netherlands, 2013. ISBN: 9789401595407. URL: https://books.google.fr/books?id=HXzvCAAAQBAJ.

[117] Erich Peter Klement, Radko Mesiar, and Endre Pap. “Triangular norms. Position paper III: Continuous t-norms”. In: *Fuzzy Sets and Systems* 145 (Aug. 2004), pp. 439–454. DOI: 10.1016/S0165-0114(03)00304-X.

[118] Graham Klyne and Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C. 2004. URL: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (visited on 03/15/2015).

[119] Stanley Kok and Pedro Domingos. “Learning the structure of Markov logic networks”. In: *Proceedings of the 22nd International Conference on Machine Learning*. ICML ’05. Bonn, Germany: Association for Computing Machinery, 2005, pp. 441–448. ISBN: 1595931805. DOI: 10.1145/1102351.1102407. URL: https://doi.org/10.1145/1102351.1102407.

[120] Stanley Kok and Pedro Domingos. “Statistical Predicate Invention”. In: *Proceedings of the 24th International Conference on Machine Learning*. ICML ’07. Corvalis, Oregon, USA: Association for Computing Machinery, 2007, pp. 433–440. ISBN: 9781595937933. DOI: 10.1145/1273496.1273551. URL: https://doi.org/10.1145/1273496.1273551.

[121] Bhushan Kotnis and Vivi Nastase. *Analysis of the Impact of Negative Sampling on Link Prediction in Knowledge Graphs*. 2018. arXiv: 1708.06816 [cs.AI].

[122] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet classification with deep convolutional neural networks”. In: *Commun. ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: https://doi.org/10.1145/3065386.

[123] Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. "A Description Logic Primer". In: *Perspectives on Ontology Learning*. Ed. by Jens Lehmann and Johanna Völker. IOS Press, 2014.

[124] Jonathan Lajus, Luis Galárraga, and Fabian M. Suchanek. "Fast and Exact Rule Mining with AMIE 3". In: *The Semantic Web* 12123 (2020), pp. 36–52. URL: https://api.semanticscholar.org/CorpusID:211559724.

[125] Jonathan Lajus and Fabian M. Suchanek. "Are All People Married? Determining Obligatory Attributes in Knowledge Bases". In: *Proceedings of the 2018 World Wide Web Conference*. WWW '18. Lyon, France: International World Wide Web Conferences Steering Committee, 2018, pp. 1115–1124. ISBN: 9781450356398. DOI: 10.1145/3178876.3186010. URL: https://doi.org/10.1145/3178876.3186010.

[126] Luis C. Lamb et al. *Graph Neural Networks Meet Neural-Symbolic Computing: A Survey and Perspective*. 2021. arXiv: 2003.00330 [cs.AI].

[127] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[128] Douglas B. Lenat, Mayank Prakash, and Mary Shepherd. "CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks". In: *AI Magazine* 6.4 (Mar. 1985), p. 65. DOI: 10.1609/aimag.v6i4.510. URL: https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/510.

[129] Nicola Leone et al. "The DLV system for knowledge representation and reasoning". In: *ACM Transactions on Computational Logic* 7.3 (July 2006), pp. 499–562. DOI: 10.1145/1149114.1149117. URL: https://doi.org/10.1145%2F1149114.1149117.

[130] Juanhui Li et al. "Are Message Passing Neural Networks Really Helpful for Knowledge Graph Completion?" In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Ed. by Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki. Toronto, Canada: Association for Computational Linguistics, July 2023, pp. 10696–10711. DOI: 10.18653/v1/2023.acl-long.597. URL: https://aclanthology.org/2023.acl-long.597.

[131] Ziyang Li, Jiani Huang, and Mayur Naik. "Scallop: A Language for Neurosymbolic Programming". In: *Proc. ACM Program. Lang.* 7.PLDI (June 2023). DOI: 10.1145/3591280. URL: https://doi.org/10.1145/3591280.

[132] Yankai Lin et al. "Learning Entity and Relation Embeddings for Knowledge Graph Completion". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 29.1 (Feb. 2015). DOI: 10.1609/aaai.v29i1.9491. URL: https://ojs.aaai.org/index.php/AAAI/article/view/9491.

[133] Weijie Liu et al. "K-BERT: Enabling Language Representation with Knowledge Graph". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.03 (Apr. 2020), pp. 2901–2908. DOI: 10.1609/aaai.v34i03.5681. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5681.

[134] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692.

[135] Michael A. Lones. *How to avoid machine learning pitfalls: a guide for academic researchers*. https://arxiv.org/abs/2108.02497. Accessed: 2023-01-04. 2023. arXiv: 2108.02497 [cs.LG].

[136] Qing Lu and Lise Getoor. "Link-Based Classification". In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML'03. Washington, DC, USA: AAAI Press, 2003, pp. 496–503. ISBN: 1577351894.

[137] Mario Lucic et al. "Are GANs Created Equal? A Large-Scale Study". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/e46de7e1bcaaced9a54f1e9d0d2f800d-Paper.pdf.

[138] Y. Ma and J. Tang. *Graph Neural Networks. In Deep Learning on Graphs*. Cambridge: Cambridge University Press., 2021.

[139] Robin Manhaeve et al. *DeepProbLog: Neural Probabilistic Logic Programming*. 2018. DOI: 10.48550/ARXIV.1805.10872. URL: https://arxiv.org/abs/1805.10872.

[140] Tiina Manninen et al. "Challenges in Reproducibility, Replicability, and Comparability of Computational Models and Tools for Neuronal and Glial Networks, Cells, and Subcellular Structures". In: *Frontiers in Neuroinformatics* 12 (2018). ISSN: 1662-5196. DOI: 10.3389/fninf.2018.00020. URL: https://www.frontiersin.org/article/10.3389/fninf.2018.00020.

[141] Gary Marcus. *The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence*. 2020. arXiv: 2002.06177 [cs.AI].

[142] Giuseppe Marra et al. "Relational Neural Machines". In: *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*. Ed. by Giuseppe De Giacomo et al. Vol. 325. Frontiers in Artificial Intelligence and Applications. IOS Press, 2020, pp. 1340–1347. DOI: 10.3233/FAIA200237. URL: https://doi.org/10.3233/FAIA200237.

[143] Frank Massey. "The Kolmogorov-Smirnov test for goodness of fit". English. In: *Journal of the American Statistical Association* 46.253 (1951), pp. 68–78.

[144] A.K. McCallum. "Automating the construction of internet portals with machine learning". In: *Information Retrieval* 3 (Jan. 2000), pp. 127–163.

[145] Edgar Meij. *Understanding News using the Bloomberg Knowledge Graph. Invited talk at the Big Data Innovators Gathering (TheWebConf)*. https://www.linkedin.com/blog/engineering/knowledge/building-the-linkedin-knowledge-graph. 2019.

[146] Christian Meilicke et al. "Anytime Bottom-up Rule Learning for Knowledge Graph Completion". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. IJCAI'19. Macao, China: AAAI Press, 2019, pp. 3137–3143. ISBN: 9780999241141.

[147] Christian Meilicke et al. *Reinforced Anytime Bottom Up Rule Learning for Knowledge Graph Completion*. 2020. arXiv: 2004.04412 [cs.AI].

[148] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

[149] George A. Miller. "WordNet: A Lexical Database for English". In: *Commun. ACM* 38.11 (Nov. 1995), pp. 39–41. ISSN: 0001-0782. DOI: 10.1145/219717.219748. URL: https://doi.org/10.1145/219717.219748.

[150] Tim Miller. *Explanation in Artificial Intelligence: Insights from the Social Sciences*. 2018. arXiv: 1706.07269 [cs.AI].

[151] Sameh K. Mohamed et al. "Loss Functions in Knowledge Graph Embedding Models". In: *DL4KG@ESWC*. 2019. URL: https://api.semanticscholar.org/CorpusID:186206474.

[152] Salman Mohammed, Peng Shi, and Jimmy Lin. "Strong Baselines for Simple Question Answering over Knowledge Graphs with and without Neural Networks". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. Ed. by Marilyn Walker, Heng Ji, and Amanda Stent. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 291–296. DOI: 10.18653/v1/N18-2047. URL: https://aclanthology.org/N18-2047.

[153] Stephen Muggleton and Luc de Raedt. "Inductive Logic Programming: Theory and methods". In: *The Journal of Logic Programming* 19-20 (1994). Special Issue: Ten Years of Logic Programming, pp. 629–679. ISSN: 0743-1066. DOI: https://doi.org/10.1016/0743-1066(94)90035-3. URL: https://www.sciencedirect.com/science/article/pii/0743106694900353.

[154] NeurIPS. *Reproducibility Challenge @ NeurIPS 2019. The Annual Machine Learning Reproducibility Challenge*. https://reproducibility-challenge.github.io/neurips2019/. Accessed: 2023-01-04. 2019.

[155] Dai Quoc Nguyen et al. "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)*. Ed. by Marilyn Walker, Heng Ji, and Amanda Stent. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 327–333. DOI: 10.18653/v1/N18-2053. URL: https://aclanthology.org/N18-2053.

[156] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. "Holographic embeddings of knowledge graphs". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, Arizona: AAAI Press, 2016, pp. 1955–1961.

[157] Maximilian Nickel et al. "A Review of Relational Machine Learning for Knowledge Graphs". In: *Proceedings of the IEEE* 104.1 (Jan. 2016), pp. 11–33. ISSN: 1558-2256. DOI: 10.1109/jproc.2015.2483592. URL: http://dx.doi.org/10.1109/JPROC.2015.2483592.

[158] Natasha Noy et al. “Industry-scale Knowledge Graphs: Lessons and Challenges”. In: *Communications of the ACM* 62 (8) (2019), pp. 36–43. URL: https://cacm.acm.org/magazines/2019/8/238342-industry-scale-knowledge-graphs/fulltext.

[159] Simon Ott, Christian Meilicke, and Matthias Samwald. *SAFRAN: An interpretable, rule-based link prediction method outperforming embedding models*. 2021. arXiv: 2109.08002 [cs.AI].

[160] PapersWithCode. *Node Classification*. https://paperswithcode.com/task/node-classification. Leaderboards. Accessed: 2023-01-04. 2023.

[161] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[162] Hung Viet Pham et al. “Problems and Opportunities in Training Deep Learning Software Systems: An Analysis of Variance”. In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. ASE ’20. Virtual Event, Australia: Association for Computing Machinery, 2021, pp. 771–783. ISBN: 9781450367684. DOI: 10.1145/3324884.3416545. URL: https://doi.org/10.1145/3324884.3416545.

[163] Gabriele Piantadosi, Stefano Marrone, and Carlo Sansone. *On Reproducibility of Deep Convolutional Neural Networks Approaches*. 2019.

[164] Joelle Pineau et al. “Improving Reproducibility in Machine Learning Research (A Report from the NeurIPS 2019 Reproducibility Program)”. In: *Journal of machine learning research* 22.164 (2021), pp. 1–20. URL: https://jmlr.org/papers/v22/20-303.html.

[165] Hans Plesser. “Reproducibility vs. replicability: a brief history of a confused terminology”. In: *Frontiers in neuroinformatics* 11 (2018). DOI: 10.3389/fninf.2017.00076.

[166] PyTorch. *Reproducibility*. https://pytorch.org/docs/stable/notes/randomness.html. Accessed: 2023-01-04. 2022.

[167] Meng Qu and Jian Tang. “Probabilistic Logic Neural Networks for Reasoning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/13e5ebb0fa112fe1b31a1067962d74a7-Paper.pdf.

[168] Meng Qu et al. “{RNNL}ogic: Learning Logic Rules for Reasoning on Knowledge Graphs”. In: *International Conference on Learning Representations*. 2021.

[169] Edward Raff. “A Step Toward Quantifying Independently Reproducible Machine Learning Research”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/c429429bf1f2af051f2021dc92a8ebea-Paper.pdf.

[170] Simon Razniewski et al. “Completeness, Recall, and Negation in Open-world Knowledge Bases: A Survey”. In: *ACM Comput. Surv.* 56.6 (Feb. 2024). ISSN: 0360-0300. DOI: 10.1145/3639563. URL: https://doi.org/10.1145/3639563.

[171] Hongyu Ren et al. *Neural Graph Reasoning: Complex Logical Query Answering Meets Graph Databases*. 2023. arXiv: 2303.14617 [cs.DB].

[172] Matthew Richardson and Pedro Domingos. “Markov Logic Networks”. In: *Machine Learning* 62 (Feb. 2006), pp. 107–136. DOI: 10.1007/s10994-006-5833-1.

[173] Andrea Rossi et al. “Knowledge Graph Embedding for Link Prediction: A Comparative Analysis”. In: *ACM Transactions on Knowledge Discovery from Data* 15.2 (Jan. 2021), pp. 1–49. ISSN: 1556-472X. DOI: 10.1145/3424672. URL: http://dx.doi.org/10.1145/3424672.

[174] Nicolas P. Rougier et al. “Sustainable computational science: the ReScience initiative”. In: *PeerJ Computer Science* 3 (Dec. 2017), e142. DOI: 10.7717/peerj-cs.142.

[175] T. Konstantin Rusch, Michael M. Bronstein, and Siddhartha Mishra. *A Survey on Oversmoothing in Graph Neural Networks*. 2023. arXiv: 2303.10993 [cs.LG].

[176] Ali Sadeghian et al. “DRUM: End-To-End Differentiable Rule Mining On Knowledge Graphs”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/0c72cb7ee1512f800abe27823a792d03-Paper.pdf.

[177] Sherif Sakr et al. “The future is big graphs: a community view on graph processing systems”. In: *Communications of the ACM* 64.9 (Aug. 2021), pp. 62–71. ISSN: 1557-7317. DOI: 10.1145/3434642. URL: http://dx.doi.org/10.1145/3434642.

[178] Tian Sang, Paul Bearne, and Henry Kautz. “Performing Bayesian inference by weighted model counting”. In: *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 1*. AAAI 05. Pittsburgh, Pennsylvania: AAAI Press, 2005, pp. 475–481. ISBN: 157735236x.

[179] Md Kamruzzaman Sarker et al. *Neuro-Symbolic Artificial Intelligence: Current Trends*. 2021. arXiv: 2105.05330 [cs.AI].

[180] Franco Scarselli et al. “The Graph Neural Network Model”. In: *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.

[181] Michael Schlichtkrull et al. “Modeling Relational Data with Graph Convolutional Networks”. In: *The Semantic Web*. Ed. by Aldo Gangemi et al. Cham: Springer International Publishing, 2018, pp. 593–607. ISBN: 978-3-319-93417-4.

[182] Oleksandr Shchur et al. “Pitfalls of Graph Neural Network Evaluation”. In: *CoRR* abs/1811.05868 (2018). arXiv: 1811.05868. URL: http://arxiv.org/abs/1811.05868.

[183] Amit Singhal. *Introducing the Knowledge Graph: things, not strings*. https://blog.google/products/search/introducing-knowledge-graph-things-not/. May 2012.

[184] Koustuv Sinha et al. *CLUTRR: A Diagnostic Benchmark for Inductive Reasoning from Text*. 2019. arXiv: 1908.06177.

[185] Koustuv Sinha et al. "ML Reproducibility Challenge 2021". In: *ReScience C* 8.#48 (2 2022). DOI: 10.5281/zenodo.6574723.

[186] Evren Sirin et al. "Pellet: A practical OWL-DL reasoner". In: *Journal of Web Semantics* 5.2 (2007). Software Engineering and the Semantic Web, pp. 51–53. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2007.03.004. URL: https://www.sciencedirect.com/science/article/pii/S1570826807000169.

[187] Richard Socher et al. "Reasoning With Neural Tensor Networks for Knowledge Base Completion". In: *Advances in Neural Information Processing Systems*. Ed. by C.J. Burges et al. Vol. 26. Curran Associates, Inc., 2013. URL: https://proceedings.neurips.cc/paper_files/paper/2013/file/b337e84de8752b27eda3a12363109e80-Paper.pdf.

[188] Zhiqing Sun et al. *RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space*. 2019. arXiv: 1902.10197 [cs.LG].

[189] Zachary Susskind et al. *Neuro-Symbolic AI: An Emerging Class of AI Workloads and their Characterization*. 2021. arXiv: 2109.06133 [cs.AI].

[190] Zachary Susskind et al. "Neuro-Symbolic AI: An Emerging Class of AI Workloads and their Characterization". In: *CoRR* abs/2109.06133 (2021). arXiv: 2109.06133. URL: https://arxiv.org/abs/2109.06133.

[191] Thomas Pellissier Tanon, Gerhard Weikum, and Fabian M. Suchanek. "YAGO 4: A Reason-able Knowledge Base". In: *The Semantic Web* 12123 (2020), pp. 583–596.

[192] Yuandong Tian et al. *ELF OpenGo: An Analysis and Open Reimplementation of AlphaZero*. 2022. arXiv: 1902.04522 [cs.AI].

[193] Kristina Toutanova et al. "Representing Text for Joint Embedding of Text and Knowledge Bases". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Ed. by Lluis Marquez, Chris Callison-Burch, and Jian Su. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 1499–1509. DOI: 10.18653/v1/D15-1174. URL: https://aclanthology.org/D15-1174.

[194] Théo Trouillon et al. "Complex Embeddings for Simple Link Prediction". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 2071–2080.

[195] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[196] Petar Veličković et al. "Graph Attention Networks". In: *International Conference on Learning Representations*. 2018.

[197] Deepak Venugopal and Vibhav G Gogate. "Scaling-up Importance Sampling for Markov Logic Networks". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014.

[198] Hongwei Wang et al. *DKN: Deep Knowledge-Aware Network for News Recommendation*. 2018. arXiv: 1801.08284 [stat.ML].

[199] Jie Wang et al. "GUIDE: Training Deep Graph Neural Networks via Guided Dropout Over Edges". In: *IEEE Transactions on Neural Networks and Learning Systems* (2022), pp. 1–13. DOI: 10.1109/TNNLS.2022.3172879.

[200] Kuansan Wang et al. "Microsoft Academic Graph: When experts are not enough". In: *Quantitative Science Studies* 1 (Jan. 2020), pp. 1–18. DOI: 10.1162/qss_a_00021.

[201] Quan Wang et al. "Knowledge Graph Embedding: A Survey of Approaches and Applications". In: *IEEE Transactions on Knowledge and Data Engineering* 29.12 (2017), pp. 2724–2743. DOI: 10.1109/TKDE.2017.2754499.

[202] William Yang Wang, Kathryn Mazaitis, and William W. Cohen. "Programming with personalized pagerank: a locally groundable first-order probabilistic logic". In: *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*. CIKM '13. San Francisco, California, USA: Association for Computing Machinery, 2013, pp. 2129–2138. ISBN: 9781450322638. DOI: 10.1145/2505515.2505573. URL: https://doi.org/10.1145/2505515.2505573.

[203] Xiao Wang et al. "Heterogeneous Graph Attention Network". In: *The World Wide Web Conference*. WWW '19. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 2022–2032. ISBN: 9781450366748. DOI: 10.1145/3308558.3313562. URL: https://doi.org/10.1145/3308558.3313562.

[204] Zhen Wang et al. "Knowledge Graph Embedding by Translating on Hyperplanes". In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. AAAI'14. Québec City, Québec, Canada: AAAI Press, 2014, pp. 1112–1119.

[205] *Web Ontology Language (OWL)*. https://www.w3.org/OWL/. Accessed: 2023-03-20.

[206] Luisa Werner. "Neuro-Symbolic Integration for Reasoning and Learning on Knowledge Graphs". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 38.21 (Mar. 2024), pp. 23429–23430. DOI: 10.1609/aaai.v38i21.30415. URL: https://ojs.aaai.org/index.php/AAAI/article/view/30415.

[207] Luisa Werner et al. "Knowledge Enhanced Graph Neural Networks". In: *2023 IEEE 10th International Conference on Data Science and Advanced Analytics (DSAA)*. 2023, pp. 1–10. DOI: 10.1109/DSAA60987.2023.10302495.

[208] Luisa Werner et al. "Reproduce, Replicate, Reevaluate. The Long but Safe Way to Extend Machine Learning Methods". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 38.14 (Mar. 2024), pp. 15850–15858. DOI: 10.1609/aaai.v38i14.29515. URL: https://ojs.aaai.org/index.php/AAAI/article/view/29515.

[209] Robert West et al. “Knowledge base completion via search-based question answering”. In: *Proceedings of the 23rd International Conference on World Wide Web*. WWW ’14. Seoul, Korea: Association for Computing Machinery, 2014, pp. 515–526. ISBN: 9781450327442. DOI: 10.1145/2566486.2568032. URL: https://doi.org/10.1145/2566486.2568032.

[210] *Wikidata*. https://www.wikidata.org. Accessed: 2023-03-20.

[211] Fei Wu and Daniel S. Weld. “Automatically refining the wikipedia infobox ontology”. In: *Proceedings of the 17th International Conference on World Wide Web*. WWW ’08. Beijing, China: Association for Computing Machinery, 2008, pp. 635–644. ISBN: 9781605580852. DOI: 10.1145/1367497.1367583. URL: https://doi.org/10.1145/1367497.1367583.

[212] Lingfei Wu et al. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Singapore: Springer Singapore, 2022, p. 725.

[213] Yujun Yan et al. *Two Sides of the Same Coin: Heterophily and Oversmoothing in Graph Convolutional Neural Networks*. 2022. arXiv: 2102.06462 [cs.LG].

[214] Bishan Yang et al. *Embedding Entities and Relations for Learning and Inference in Knowledge Bases*. 2015. arXiv: 1412.6575 [cs.CL].

[215] Dingqi Yang et al. “Fast and Slow Thinking: A Two-Step Schema-Aware Approach for Instance Completion in Knowledge Graphs”. In: *IEEE Trans. on Knowl. and Data Eng.* 36.3 (Aug. 2023), pp. 1113–1129. ISSN: 1041-4347. DOI: 10.1109/TKDE.2023.3304137. URL: https://doi.org/10.1109/TKDE.2023.3304137.

[216] Haotong Yang, Zhouchen Lin, and Muhan Zhang. *Rethinking Knowledge Graph Evaluation Under the Open-World Assumption*. 2022. arXiv: 2209.08858 [cs.AI].

[217] Xiaocheng Yang et al. *Simple and Efficient Heterogeneous Graph Neural Network*. https://arxiv.org/abs/2207.02547. 2022. DOI: 10.48550/ARXIV.2207.02547.

[218] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. “Revisiting Semi-Supervised Learning with Graph Embeddings”. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML’16. New York, NY, USA: JMLR.org, 2016, pp. 40–48.

[219] Rex Ying et al. “Graph Convolutional Neural Networks for Web-Scale Recommender Systems”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 974–983. ISBN: 9781450355520. DOI: 10.1145/3219819.3219890. URL: https://doi.org/10.1145/3219819.3219890.

[220] L.A. Zadeh. “Fuzzy logic”. In: *Computer* 21.4 (1988), pp. 83–93. DOI: 10.1109/2.53.

[221] Hanqing Zeng et al. “GraphSAINT: Graph Sampling Based Inductive Learning Method”. In: *International Conference on Learning Representations*. 2020.

[222] Zefan Zeng, Qing Cheng, and Yuehang Si. “Logical Rule-Based Knowledge Graph Reasoning: A Comprehensive Survey”. In: *Mathematics* 11.21 (2023). ISSN: 2227-7390. DOI: 10.3390/math11214486. URL: https://www.mdpi.com/2227-7390/11/21/4486.

[223] Chuxu Zhang et al. "Heterogeneous Graph Neural Network". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 793–803. ISBN: 9781450362016. DOI: 10.1145/3292500.3330961. URL: https://doi.org/10.1145/3292500.3330961.

[224] Fuzheng Zhang et al. "Collaborative Knowledge Base Embedding for Recommender Systems". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 353–362. ISBN: 9781450342322. DOI: 10.1145/2939672.2939673. URL: https://doi.org/10.1145/2939672.2939673.

[225] Hanwang Zhang et al. "Visual Translation Embedding Network for Visual Relation Detection". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 3107–3115. DOI: 10.1109/CVPR.2017.331.

[226] Jing Zhang et al. "Neural, symbolic and neural-symbolic reasoning on knowledge graphs". In: *AI Open* 2 (2021), pp. 14–35. ISSN: 2666-6510. DOI: https://doi.org/10.1016/j.aiopen.2021.03.001. URL: https://www.sciencedirect.com/science/article/pii/S2666651021000061.

[227] Shuai Zhang et al. "Quaternion Knowledge Graph Embeddings". In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[228] Yongqi Zhang, Quanming Yao, and Lei Chen. *Efficient, Simple and Automated Negative Sampling for Knowledge Graph Embedding*. 2021. arXiv: 2010.14227 [cs.LG].

[229] Yongqi Zhang et al. *NSCaching: Simple and Efficient Negative Sampling for Knowledge Graph Embedding*. 2019. arXiv: 1812.06410 [cs.AI].

[230] Zhengyan Zhang et al. "ERNIE: Enhanced Language Representation with Informative Entities". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluis Marquez. Florence, Italy: Association for Computational Linguistics, July 2019, pp. 1441–1451. DOI: 10.18653/v1/P19-1139. URL: https://aclanthology.org/P19-1139.

[231] Ziwei Zhang, Peng Cui, and Wenwu Zhu. "Deep Learning on Graphs: A Survey". In: *IEEE Trans. on Knowl. and Data Eng.* 34.1 (Jan. 2022), pp. 249–270. ISSN: 1041-4347. DOI: 10.1109/TKDE.2020.2981333. URL: https://doi.org/10.1109/TKDE.2020.2981333.

A. Appendix

A.1. Experimental Details of Knowledge Enhancement of Graph Neural Networks

The training of KeGNN involves a set of hyperparameters. Batch normalization [101] is applied after each hidden layer of the GNN. The Adam optimizer [113] is used as optimizer for all models. Concerning the hyperparameters specific to the knowledge enhancement layers, the initialization of the preactivations of the binary predicates (which are assumed to be known) is taken as a hyperparameter. They are set to a high positive value for edges that are known to exist and correspond to the grounding of the binary predicate. Furthermore, different initializations of clause weights and constraints on them are tested. Moreover, the number of stacked knowledge enhancement layers is a hyperparameter. We further allow the model to randomly neglect a proportion of edges by setting an edges drop rate parameter. Further, we test whether the normalization of the edges with the diagonal matrix $\tilde{\mathbf{D}} = \sum_j \tilde{\mathbf{A}}_{i,j}$ (with $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$) is helpful.

To find a suitable set hyperparameters for each dataset and model, we perform a random search with up to 800 runs and 48h time limit and choose the parameter combination which leads to the highest accuracy on the validation set. The hyperparameter tuning is executed in Weights and Biases [22]. A random search is conducted over the following hyperparameter values and ranges.

- Adam optimizer parameters: β_1: 0.9, β_2: 0.99, ϵ: 1e-07
- Attention heads: {1, 2, 3, 4, 6, 8, 10}
- Batch size: {128, 512, 1024, 2048, full batch}
- Binary preactivation: {0.5, 1.0, 10.0, 100.0, 500.0}
- Clause weights initialization: {0.001, 0.1, 0.25, 0.5, random uniform distribution on [0,1)}
- Dropout rate: 0.5
- Edges drop rate: random uniform distribution [0.0, 0.9]
- Edge normalization: {true, false}
- Early stopping: δ_{min} : 0.001, patience: {1, 10, 100}
- Hidden layer dimension: {32, 64, 128, 256}
- Learning rate: random uniform distribution [0.0001, 0.1]
- Clause weight clipping: w_{min} : 0.0, w_{max}: random uniform distribution: [0.8, 500.0]

- Number of knowledge enhancement layers: $\{1, 2, 3, 4, 5, 6\}$
- Number of hidden layers: $\{2, 3, 4, 5, 6\}$
- Number of epochs 200 (unless training stopped early)

The obtained parameter combinations for the models KeMLP, KeGCN and KeGAT for Cora, Citeseer, PubMed and Flickr are displayed in Table A.1 and in Table A.2. We set the random seed for all experiments to 1234. The reference models MLP, GCN and GAT are trained with the same parameter set as the respective knowledge enhanced models.

	PubMed			**Flickr**		
Parameter	**KeMLP**	**KeGCN**	**KeGAT**	**KeMLP**	**KeGCN**	**KeGAT**
adam β_1	0.9	0.9	0.9	0.9	0.9	0.9
adam β_2	0.99	0.99	0.99	0.99	0.99	0.99
adam ϵ	1e-07	1e-07	1e-07	1e-07	1e-07	1e-07
attention heads	-	-	8	-	-	8
batch size	1024	full batch	1024	128	1024	2048
binary preactivation	10.0	1.0	10.0	10.0	500.0	500.0
clause weight initialization	0.001	random	0.5	0.001	0.001	0.1
dropout rate	0.5	0.5	0.5	0.5	0.5	0.5
edges drop rate	0.22	0.66	0.07	0.2	0.24	0.12
epochs	200	200	200	200	200	200
early stopping enabled	true	true	true	true	true	true
early stopping min delta	0.001	0.001	0.001	0.001	0.001	0.001
early stopping patience	100	10	10	10	10	100
hidden channels	256	256	256	32	128	64
learning rate	0.057	0.043	0.016	0.001	0.016	0.0039
max clause weight	350.0	322.0	118.0	55.0	135	113.0
min clause weight	0.0	0.0	0.0	0.0	0	0.0
normalize edges	false	false	true	true	true	false
KE layers	2	1	5	1	4	1
hidden layers	4	2	2	2	4	3
runs	50	50	50	10	10	10
seed	1234	1234	1234	1234	1234	1234

Table A.1.: KeGNN. Hyperparameter for PubMed and Flickr

A.2. Experimental Details of Knowledge Enhancement on Large-Scale Graphs

The hyperparameters used for the experiments on ogbn-arxiv in Chapter 7 are listed in Table A.3 for full-batch training. The hyperparameters for the experiments with RNS on ogbn-arxiv and ogbn-products are summarised in Table A.4 and Table A.5.

	Cora			CiteSeer		
Parameter	**KeMLP**	**KeGCN**	**KeGAT**	**KeMLP**	**KeGCN**	**KeGAT**
adam β_1	0.9	0.9	0.9	0.9	0.9	0.9
adam β_2	0.99	0.99	0.99	0.99	0.99	0.99
adam ϵ	1e-07	1e-07	1e-07	1e-07	1e-07	1e-07
attention heads	-	-	1	-	-	3
batch size	512	512	full batch	128	full batch	1024
binary preactivation	10.0	500.0	1.0	10.0	0.5	0.5
clause weight initialization	0.5	random	0.5	0.5	0.25	0.1
dropout rate	0.5	0.5	0.5	0.5	0.5	0.5
edges drop rate	0.47	0.17	0.27	0.01	0.35	0.88
epochs	200	200	200	200	200	200
early stopping enabled	true	true	true	true	true	true
early stopping min delta	0.001	0.001	0.001	0.001	0.001	0.001
early stopping patience	1	1	10	10	10	10
hidden channels	32	256	64	256	128	32
learning rate	0.026	0.032	0.033	0.028	0.037	0.006
max clause weight	104.0	254.0	250.0	34.0	243.0	110.0
min clause weight	0.0	0.0	0.0	0.0	0.0	0.0
normalize edges	true	false	true	true	false	true
KE layers	4	2	1	1	3	2
Hidden layers	2	2	2	2	5	2
runs	50	50	50	50	50	50
seed	1234	1234	1234	1234	1234	1234

Table A.2.: KeGNN. Hyperparameters for Citeseer and Cora

Parameters of Full-batch Training on ogbn-arxiv				
Parameter	**MLP**	**GCN**	**KE$_{GCN}$**	**KE$_{MLP}$**
Binary preactivation	500.0	500.0	500.0	500.0
Dropout	0.5	0.5	0.5	0.5
Epochs	600	600	600	600
Early stopping enabled	True	True	True	True
Early stopping δ	0.001	0.001	0.001	0.001
Early stopping Patience	10	10	10	10
Evaluation steps	10	10	10	10
Hidden layer dimension	256	256	256	256
Learning rate	0.01	0.01	0.01	0.01
Number of knowledge enhancement layers	-	-	3	3
Number of hidden layers	3	3	3	3
Number of runs	10	10	10	10

Table A.3.: Hyperparameters for Full-batch Training on ogbn-arxiv

RNS on ogbn-arxiv				
Parameter	**MLP**	**GCN**	**KE$_{GCN}$**	**KE$_{MLP}$**
Batch size	10000	10000	10000	10000
Sampling depth τ	-	3	3	3
Neighbor sampling size ρ	-	10	10	10
Binary preactivation	500.0	500.0	500.0	500.0
Dropout	0.5	0.5	0.5	0.5
Epochs	300	100	100	300
Early stopping snabled	True	True	True	True
Early stopping δ	0.001	0.001	0.001	0.001
Early stopping patience	10	10	10	10
Evaluation steps	10	10	10	10
Hidden layer dimension	256	256	256	256
Initialisation of clause weights	-	-	0.5	0.5
Initialisation of GCN layers	-	random (glorot)	-	random (glorot)
Initialisation of linear layers	random uniform	random uniform	random uniform	random uniform
Learning rate	0.01	0.01	0.01	0.01
Number of knowledge enhancement layers	-	-	3	3
Number of hidden layers (base neural network)	3	3	3	3
Runs	10	10	10	10

Table A.4.: Hyperparameters for RNS Training on ogbn-arxiv

RNS on ogbn-products				
Parameter	**MLP**	**GCN**	**KE$_{GCN}$**	**KE$_{MLP}$**
Batch size	10.000	10.000	10.000	10.000
Sampling depth τ	-	1	1	1
Neigbhor sampling sxize ρ	-	10	10	10
Binary preactivation	500.0	500.0	500.0	500.0
Dropout	0.5	0.5	0.5	0.5
Epochs	300	300	300	300
Early stopping enabled	True	True	True	True
Early stopping δ	0.001	0.001	0.001	0.001
Early stopping patience	10	10	10	10
Evaluation steps	10	10	10	10
Hidden layer dimension	256	256	256	256
Initialisation of clause weights	-	-	0.5	0.5
Initialisation of GCN layers	-	random (glorot)	-	random (glorot)
Initialisation of linear layers	random uniform	random uniform	random uniform	random uniform
Learning rate	0.01	0.01	0.01	0.01
Number of knowledge enhancement layers	-	-	1	1
Number of hidden layers (base NN)	3	3	3	3
Runs	10	10	10	10

Table A.5.: Hyperparameters for RNS Training on ogbn-products