Deusto

Universidad de Deusto
Deustuko Unibertsitatea
University of Deusto

Towards more interpretable graphs and Knowledge Graph algorithms

PhD dissertation by Unai Zulaika Zurimendi

within the doctoral program ENGINEERING FOR THE INFORMATION SOCIETY AND SUSTAINABLE DEVELOPMENT

Candidate	Advisors	
Unai Zulaika Zurimendi	Dr. Diego López-de-Ipiña González-de-Artaza	Dr. Aitor Almeida Escondrillas

"Quisiera dedicaros unos versos
que valieran lo que vale vuestra luz
pero no cabe en un folio el universo
tampoco en una frase cabe mi gratitud
han de saber que decidí no decir nombres
por si alguno se quedaba en el tintero
así que va por las mujeres y los hombres
por todos aquellos que me quieren y que quiero."
- *Sharif*

Abstract

The increase in the amount of data generated by today's technologies has led to the creation of large graphs and Knowledge Graphs that contain millions of facts about people, things and places in the world. Grounded on those large data stores, many Machine Learning models have been proposed to achieve different tasks, such as predicting new links or weights. Nevertheless, one of the main challenges of those models is their lack of interpretability. Commonly known as "black boxes", Machine Learning models are usually not understandable to humans. This lack of interpretability becomes even a more severe problem for Knowledge graph-related applications, including healthcare systems, chatbots, or public service management tools where end-users require an understanding of the feedback given by the models.

In this thesis, we present methods to increase the interpretability of graphs and Knowledge Graphs based Machine Learning models. We follow a taxonomy grounded on the output result obtained by the proposed methods. Each of the different methods is suitable for particular use cases and scenarios, and can help end-users in different manners. Precisely, we provide an interpretable link weight prediction method based on the Weisfeiler-Lehman graph colouring technique. Additionally, we present an adaption of the Regularized Dual Averaging optimization method for Knowledge Graphs to obtain interpretable representations in link prediction models. Lastly, we introduce the use of Influence Functions for

Knowledge Graph link prediction models to acquire the most important training facts for a given prediction. Through experiments in link weight prediction and link prediction, we show that our methods can successfully increase the interpretability of the machine learning models of graphs and Knowledge Graphs while maintaining competition with state-of-the-art methods in terms of performance.

Resumen

El aumento de la cantidad de datos generados por las tecnologías actuales ha llevado a la creación de grandes grafos y Grafos de Conocimiento que contienen millones de datos sobre personas, hechos, cosas y lugares del mundo. Utilizando estos grandes almacenes de datos, se han propuesto varios modelos de aprendizaje automático para realizar diferentes tareas, como la predicción de nuevos enlaces o la predicción de pesos. Sin embargo, uno de los principales problemas de estos modelos es su falta de interpretabilidad. Comúnmente conocidos como ”cajas negras”, los modelos de aprendizaje automático no suelen ser comprensibles para los humanos. Esta falta de interpretabilidad se convierte en un problema aún más grave para las aplicaciones relacionadas con los grafos de conocimiento, aplicaciones que incluyen, entre otros, sistemas de salud, chatbots, o herramientas de gestión de servicios públicos donde los usuarios finales requieren una comprensión de las decisiones tomadas por los modelos.

En esta tesis, se presentan diferentes métodos para aumentar la interpretabilidad los modelos de aprendizaje automático basados en grafos y grafos de conocimiento. Dichos métodos, se fundamentan en una taxonomía basada en el tipo de resultado obtenido por los mismos. Cada uno de los diferentes métodos es adecuado para casos de uso y escenarios particulares, y puede ayudar a los usuarios finales de diferentes maneras. En concreto, proporcionamos un método interpretable de predicción del peso de los enlaces basado en la técnica de coloración de gráfos de Weisfeiler-Lehman.

Además, presentamos una adaptación del método de optimización de "Regularized Dual Averaging" aplicado para Grafos de Conocimiento con el objetivo de obtener representaciones interpretables en modelos de predicción de enlaces. Por último, introducimos el uso de Funciones de Influencia para modelos de predicción de enlaces en Grafos de Conocimiento con el propósito de adquirir los ejemplos (hechos) de entrenamiento más importantes para una predicción dada. A través de experimentos en la predicción del peso de los enlaces y en la predicción de los enlaces mismos, demostramos que nuestros métodos pueden aumentar con éxito la interpretabilidad de los modelos de aprendizaje automático de los grafos y de los Grafos de Conocimiento, manteniendo a su vez la eficiencia con los métodos del estado del arte en términos de rendimiento.

Acknowledgements

Diegori aukera emateagatik eta **Aitorri** bideari eusten laguntzeagatik.

Rubas, **Perurena** eta **Oihaneri** ❤.

Egon zirenei, **Koldo** , **Enrique** eta **Mulero**.

Egunerokotasuean daudenei, **Emaldi**, **Dieguich**, **Aguilera** eta **kafekideei**.

Lagunei **Sendo**, **Odei**, **Rufo**, **Endika**, **Josulin**, **Jon**, **Sandra**, **Borja** eta falta direnei.

Lagunak ere diren futbolkideei, **Imanol**, **Garzo**, **Xabi**, **Bran** eta **Pintori**.

Familiari, **ama**, **aita**, **Igor**, **Naiara**, **Eider**, **Oier**, **Lur** eta **izenikgaberi**.

Zuei, guztiei (eta bidean lagundu duten orori), eskerrik asko!

Table of Contents

List of Figures **v**

List of Tables **vii**

List of Listings **viii**

Acronyms **ix**

1 Introduction **1**

1.1 Context and Motivation 4

1.1.1 About interpretability 5

1.2 Hypothesis, Objectives and Scope 10

1.3 Research Methodology 13

1.4 Main Contributions 14

1.5 Thesis Outline 16

2 Related work **17**

2.1 Link weight prediction in graphs 18

2.2 Link prediction in Knowledge Graphs 21

2.3 Interpretability in graphs and Knowledge Graphs 27

3 Leveraging structural nodes in graphs for link weight prediction **33**

3.1 Graph colouring and subgraph extraction for interpretable link weight prediction 34

3.2 Approach and methodology 35

3.2.1 Subgraph extraction . . . 37
3.2.2 Subgraph node ordering . . . 39
3.2.3 Model . . . 40
3.2.3.1 Edge to edge filter. . . . 42
3.2.3.2 Edge to node filter. . . . 42
3.3 Experiments and results . . . 43
3.3.1 Datasets and baselines . . . 44
3.3.2 Implementation details . . . 51
3.3.3 Comparison . . . 53
3.3.4 Scalability and complexity . . . 55
3.3.5 Ablation study . . . 57
3.3.6 Results and conclusions . . . 60
3.4 Interpretability . . . 61

4 Sparse Knowledge Graph Embeddings for intrinsic interpretability 67
4.1 Sparsity and interpretability from cognitive arguments . . . 68
4.2 Sparse Tensor Factorization . . . 69
4.2.1 Background . . . 69
4.2.2 RESCAL model . . . 70
4.2.3 Sparse RESCAL . . . 70
4.2.4 Optimization . . . 72
4.3 Experiments and results . . . 73
4.3.1 Datasets . . . 73
4.3.2 Implementation details . . . 74
4.3.3 Link prediction . . . 75
4.3.4 Interpretability . . . 78

5 Influence functions for interpretable link prediction in Knowledge Graphs 83
5.1 An influential instances method for explaining model decisions 84
5.2 Procedure and Methodology . . . 86
5.2.1 Knowledge Graph Embedding models . . . 86
5.2.2 Influence Functions . . . 87

5.2.3 KGInfluence 88
5.2.3.1 Summary 88
5.2.3.2 Basic approach 90
5.2.3.3 Reducing the complexity 91
5.3 Experiments and discussion 93
5.3.1 Datasets 93
5.3.2 Implementation details 93
5.3.3 Efficiency 94
5.3.4 Examples and conclusions 96

6 Conclusions and Future Work **99**
6.1 Summary of work and conclusions 100
6.2 Contributions 101
6.3 Hypothesis and objective validation 104
6.4 Relevant publications 106
6.4.1 International JCR Journals and Book Chapters 106
6.4.2 International Conferences 107
6.4.3 Technical Contributions 108
6.5 Future work 108
6.6 Final remarks 109

Bibliography **111**

List of Figures

1.1 A small knowledge graph that contains the fact (*Eiffel Tower, is_located_in, Paris*). Source: Knowledge Graphs on AWS. . 2

1.2 Summary of the thesis. Relationship between objectives, chapters and the developed interpretability methods. 12

1.3 Schematic representation of the research methodology followed in this dissertation. 15

2.1 Tensor representation of a KG ($\underline{\mathbf{Y}}$). Where the i and j axes correspond to entities and the k axe to relationships. In each entry, a triple (y_{ijk}) is registered as true or falso (0 or 1). . . . 25

3.1 Summary of the steps for the LWP-WL link weight prediction framework. 36

3.2 First iteration of the W-WL algorithm. We want to predict the weight (w, coloured in red) of the link for the target nodes (dashed and coloured in yellow). First, we generate initial colours for each node. Then we generate a string signature for the nodes. Finally, we evaluate all string signatures and select the lexicographically lowest one. Target nodes get the 1 and 2 numbers to preserve the topological directionality. 41

3.3 The special filters applied to a $K = 6$ size adjacency matrix. **Left**: edge to edge filter applies a convolution over the neighbouring edges for the target link. **Right:** edge to node filter that applies a convolution over the neighbouring edges of each node in the adjacency matrix. 42
3.4 The same graph presented in two different ways. After applying Weighted-WL, we obtain the same node ordering and signature strings (only signatures after the first step are given). 62
3.5 Two different subgraphs (left) are extracted to perform the node ordering algorithm. After applying W-WL (right), we can compare the node order to find similar roles between both subgraphs. Furthermore, we could check each node's signature to compare the level of similarity (the signature was not presented in the figure to facilitate the presentation). 64

4.1 Results on FB15k-237 for fixed hyperparameters except μ. . . 79

5.1 A linear model with one feature. Trained once on the full data and once without the influential instance. Removing the influential instance changes the fitted slope (weight/coefficient) drastically. From (Molnar, 2020). 85
5.2 Updating the model parameter (x-axis) by forming a quadratic expansion of the loss around the current model parameter, and moving 1/n into the direction in which the loss with upweighted instance z (y-axis) improves most. This upweighting of instance z in the loss approximates the parameter changes if we delete z and train the model on the reduced data.. From (Molnar, 2020). 88
5.3 Summary and example for the KGInfluence process. 89

6.1 Contributions, objectives and publications map. 104

List of Tables

2.1 Comparison of the state-of-the-art models and the most remarkable characteristics for link weight prediction. 21
2.2 Comparison table for the presented models. 27
2.3 Summary of the works for improving graph and KG interpretability. 30

3.1 Summary of the datasets used for experiments, $|V|$ represents the number of nodes and $|E|$ the number of links. 45
3.2 Mean Squared Errors, standard deviation and improvement percentage for the compared approaches on all of the datasets, results for pWSBM, bWSBM, SBM, DCWBM and ModelR belong to (Hou and Holder, 2017). 54
3.3 Comparison between different methods and our approach, Mean Squared Errors, standard deviation, parameter amount and improvement percentage in the big-sized Astro and Geom datasets. 57
3.4 Summary of the versions used for the ablation study. 58
3.5 Results for each version of the algorithm applied to the Airport dataset. We report the Pearson Correlation Coefficient and the Mean Squared errors with the Standard Deviation for 25 trials on each version. We also provide the improvement percentage of every version with regard to the previous one for each metric. 59

4.1 Datasets for link prediction and their number of entities and relations. 74

4.2 Best hyperparameters for sRESCAL for datasets where: LR denotes learning rate, dr decay rate, ls label smoothing, and $d\#k, k \in \{1, 2, 3\}$ dropout values applied on the subject entity embedding, relation matrix and subject entity embedding after it has been transformed by the relation matrix respectively. . . 75
4.3 Results in link prediction. 76
4.4 Sparsity, DistRatio and negativity results for FB15k-237. . . . 81
4.5 Sparsity, DistRatio and negativity results for WN18RR. . . . 81
4.6 Top 5 words of some dimensions in RESCAL and sRESCAL. . 82

5.1 Datasets for link prediction and their number of entities and relations. 93
5.2 Results obtained for KGInfluence in the FB15k-237 and WNRR datasets. 94
5.3 Example of the interpretability provided by KGInfluence. We provide the three most influential triples for a target triple. . . 98

Acronyms

BFS Breadth-First-Search

CNN Convolutional Neural Network

DL Deep Learning

GCN Graph Convolutional Network

IF Influence Function

KG Knowledge Graph

KGE Knowledge Graph Embedding

ML Machine Learning

RDA Regularized Dual Averaging

WL Weisfeiller-Lehmann

"Kaiola erraldoi horretatik irteteko".

CHAPTER

1

Introduction

The rise of Machine Learning (ML) and Deep Learning (DL) over the last decade (Bengio et al., 2013) has led to significant advances in many areas from healthcare (Miotto et al., 2018) to computer vision (Voulodimos et al., 2018), or natural language processing (Otter et al., 2020). Many of the most groundbreaking scientific discoveries and contributions in the last years came from ML models. Nevertheless, one of DL's leading issues and challenges is the lack of interpretability of the models (Gilpin et al., 2018). The concept of interpretability is further discussed in the following subsection; however, as an introduction to the concept, we understand interpretability as "the degree to which a human can understand the cause of a decision". Unfortunately, it is commonly hard for humans to understand why a model makes a concrete decision. In fact, many of these models are called "black-box" methods for this reason. Thus, the lack of interpretability raises one of the most challenging problems for the ML research community to solve.

Furthermore, as authors present in (Doshi-Velez and Kim, 2017), interpretability is used to confirm other important desiderata for ML systems and models. Properties such as fairness, privacy, robustness, causality or trust are closely related to interpretability, making it even more essential for ML

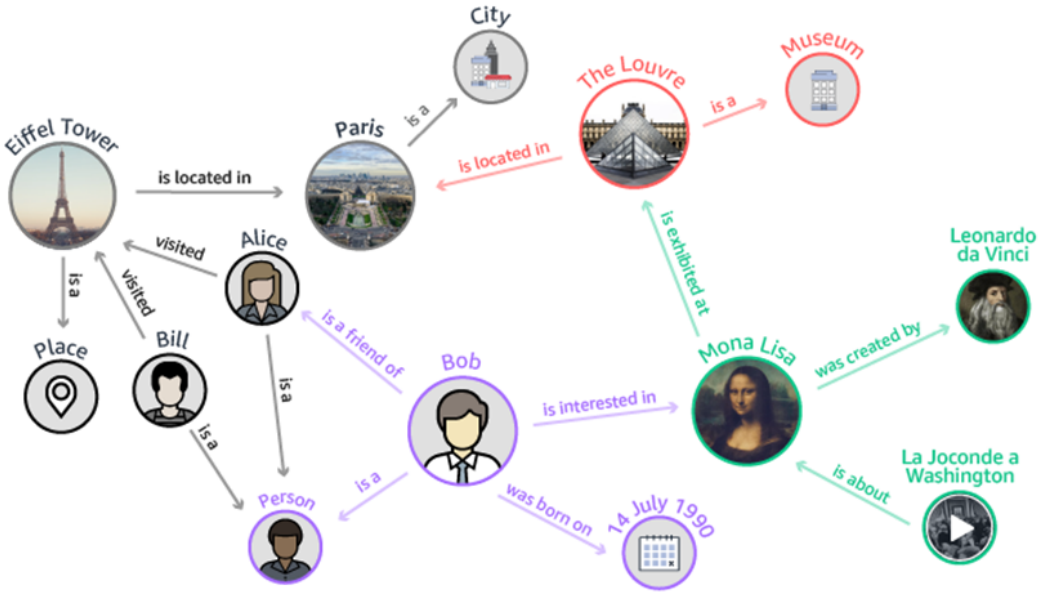


Figure 1.1: A small knowledge graph that contains the fact (*Eiffel Tower, is_located_in, Paris*). Source: Knowledge Graphs on AWS.

models. We further present and discuss the properties and aspects related to interpretability in the following subsection as the context of the thesis.

Representing human knowledge is one of the research challenges in ML. ML aims to find intelligent systems capable of solving complex tasks by leveraging such knowledge. In this context, Knowledge Graphs (KGs) have drawn the attention of academia and industry in the recent years (Fensel et al., 2020). KGs are graph representations of the real world in the form of facts. KGs consist of entities, relationships between them and semantic descriptions. Entities can be natural or abstract concepts, and relationships link those concepts together. For instance, the fact that the Eiffel Tower is in Paris is could be represented in a KG by linking together the *Eiffel Tower* and *Paris* entities by the relationship *is_located_in*. This link forms the fact (*Eiffel Tower, is_located_in, Paris*). A depiction of a KG is presented in Figure 1.1.

Knowledge Graphs (KGs) (Fensel et al., 2020) are commonly used to support ML models in different ways: as a method to incorporate world knowledge

(Ghosh et al., 2021), or as a source for learning concept representations (Nickel et al., 2015), and for explaining what is learned (Palmonari and Minervini, 2020). Thus, they are core in many applications and critical systems that citizens may use or be affected by every day: voice assistants, shopping recommendations, search engines, biomedicine tools, financial analytics... However, the models built upon KGs are not free of the issues that modern ML and DL methods have. Their lack of interpretability is a crucial aspect to overcome for their adoption at a large scale. Clinicians need to understand why a model gives a certain diagnosis to a patient, and citizens should understand why their loan was denied. Safety, ethics and fairness are vital properties that ML models must have in many use cases. Experts or not, end-users should understand the decision making process of the tools they are using.

The scientific community has been working to increase interpretability in ML models in recent years (Molnar, 2020). This research has led to significant contributions in the area. Local surrogate models (Ribeiro et al., 2016) apply an interpretable technique such as linear regression or decision trees to explain an individual prediction (instead of trying to explain every training sample) of the whole model, thus a "surrogate". On the other hand, feature visualisation (Olah et al., 2018), which is applied in Computer Vision, tries to provide an understandable human view of what abstract features and concepts neural networks learn from raw image pixels. Besides, pixel attribution techniques (Simonyan et al., 2013) also named saliency maps are also being used; these techniques highlight the most relevant pixels from an image when giving a prediction. Still, interpretability remains a pitfall to overcome, even more so in the case of KGs. As a specific application area, a few methods were proposed to increase interpretability in KGs models (Palmonari and Minervini, 2020), but there is scope for much more research on this area. As KGs grow in popularity due to their flexibility and heterogeneity, the need to develop new methods and techniques to interpret KG-based models increases. Hence, this dissertation aims to provide a set of tools for increasing interpretability in graphs and KGs and provide ways for end-users and citizens to understand better the insights gained through the tools that they use.

The remainder of this chapter is structured as follows: Section 1.1 explains the context and motivation of the research. Afterwards, Section 1.2 formulates the hypothesis, derived goals, and the work's scope. Section 1.3 describes the research methodology used to achieve the goals, and Section 1.4 summarises this dissertation's scientific and technical contributions. The chapter finalises with an outline of the dissertation in Section 1.5.

1.1 Context and Motivation

A KG, also named sometimes "knowledge base" or "semantic network", represents a network of real-world entities. These entities can be objects, events, situations, or concepts, and KGs illustrate how they relate to each other. The increase in the amount of data and the variety of data sources nowadays has turned into an upsurge of graphs and KGs. Due to their scalability and flexibility, KGs are ideal tools for many applications and use cases. Whenever an application needs to combine different databases and sources, or when complex queries and recommendations have to be delivered, KGs are a suitable tool. Such an upsurge has also raised the necessity of improving specific characteristics of KGs. From the building to the exploitation processes, new techniques have been developed. ML models and their interpretability are not an exception. Developers need to deliver efficient yet understandable KG-based ML models. End-users of many kinds, citizens, clinicians... use these models in applications where interpretability is necessary. Therefore, researching and developing new techniques to increase interpretability is crucial for society.

However, there are still many questions around interpretability in the research community: What is interpretability? Are there different types of interpretability? How do we evaluate it? Although these questions have widely been discussed in the research community, there is no clear consensus on the topic.

1.1.1 About interpretability

First, it is important to remark that the terms explainability and interpretability are often used interchangeably in machine learning. Though closely related, both terms are not equivalent. Interpretability is about how a cause and effect can be observed within a system. Meanwhile, explainability is the extent to which the internal mechanics of a machine or deep learning system can be explained in human terms. In simple terms, one can understand why something happens without being able to explain it in words, thus interpreting but not explaining.

In this work, we focus on interpretability. The novel methods and techniques developed through the thesis are meant to increase interpretability in different manners but not necessarily to explain the internal mechanics of the models. Explaining can be understood as the process of communicating what has been interpreted. Since interpretability is, after all, a prerrequisite for explainability, we leave increasing explainability for future work (further discussed in the last section).

On the definition of interpretability

In this thesis, we follow the definitions of interpretability given by (Miller, 2019) and (Kim et al., 2016). Respectively, interpretability "is the degree to which a human can understand the cause of a decision" and "is the degree to which a human can consistently predict the model's result". Ergo, the easier it is for someone to comprehend why certain decisions or predictions have been made, the better interpretable the model is. We ground this notion of interpretability in developing the work and research done through the thesis.

Interpretability and its characteristics

As we previously introduced, interpretability is of vital importance in ML. Indeed, there are many use cases and scenarios where interpretability is needed or recommended, although not always necessary. As authors from (Doshi-Velez and Kim, 2017) state, "The problem is that a single metric, such as classification accuracy, is an incomplete description of most real-world tasks".

There are many situations where we only care about the outcome of a model's predictions, i.e. we only care about the **what**. This kind of scenario

happens when we are in a situation where mistakes do have low or any consequences, e.g. a chatbot; or the model we are using has already been widely studied and tested, e.g. a speech recognition model. Nevertheless, there are many other situations where we do not only care about the what but also about the **why**. There might be a requirement for interpreting a model's predictions in those cases since the prediction itself might not solve the original problem. The following aspects can require interpretability in the ML model (Molnar, 2020):

1. **Human curiosity and learning**: human beings learn by relationship and understanding cause-effect processes. For scientists and researchers to properly understand why a model gives predictions and achieve scientific findings, it is sometimes required to increase interpretability. Through interpretability, the background processes of ML are better understood, allowing to research new directions and find the key aspects to look at.

2. **Safety**: there are many ML models which run in critical scenarios where safety is a must, e.g. a nuclear power station, governments decision-making tools or an autonomous car. In those scenarios, developers have to ensure that the system learns properly, error-free, and without exception. Having interpretability tools in such models is vital to understanding and ensuring how the model learned, securing the system's safety.

3. **Detecting bias**: in many cases, a ML model gives predictions that affect relevant things in people's lives. For instance, many entities use selection algorithms to hire new employees or accept candidate enrollments. Unfortunately, ML models pick up biases from the training data. These biases can turn into discriminative algorithms towards groups of people by different criteria, e.g. genre o ethnicity. Many reports include hiring bias towards women[1] or racial discrimination toward black people

[1] https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G

in face recognition systems[1]. Including interpretability in ML models, one can avoid or at least identify possible discriminations or bias in the system, thus developing a fairer technology.

4. **Social acceptance**: to successfully integrate algorithms and ML models into people's everyday routine, they need to be socially accepted. As authors in (Heider and Simmel, 1944) present, humans tend to assign beliefs, desires and intentions to objects. As a conclusion of the experiment, those machines that can explain (or at least people can interpret) their predictions will be more acceptable, thus, better integrated into people's everyday life. Indeed, explanations are an important part of social interactions. Explanations share intentions and goals, and thanks to them, we can empathise and accept whatever is trying to interact with us.

5. **Debug and audit**: related to safety, a ML model can almost solely be debugged and audited when it is interpretable. Understanding predictions and finding possible errors or outliers helps developers ensure a production-ready system. Furthermore, in case a system needs to be audited, the only way to do is through interpretability tools.

Until now, we have presented the most vital aspects to consider when introducing interpretability in a ML model. In this thesis, we focus on inducing interpretability for KG-based models. As already stated, KGs are widely used in many different use cases, and scenarios (Ji et al., 2021). Therefore, each aspect could be considered relevant depending on the situation. However, most commonly, KGs are used in low-risk scenarios where there is much end-user interaction, e.g. recommender systems, chatbots, public service management, transport systems... We argue that detecting biases and having higher social acceptance are the most relevant aspects in those use-cases. Whenever we offer a service to citizens, it is critical to avoid biases to provide an honest and valuable service, e.g. a government system must never show any bias toward

[1] https://sitn.hms.harvard.edu/flash/2020/racial-discrimination-in-face-recognition-technology/

anybody. Moreover, many people commonly use KG-based systems such as chatbots or recommender systems every day. To confirm those systems are practical and have an impact, they have to be socially accepted. Otherwise, we would provide systems that are not used due to the "black box" behaviour that they exhibit.

Finally, we remark on the characteristics derived from interpretability when delivering a ML system. By detecting possible biases, interpretability helps achieve a higher degree of **fairness**. We can identify possible discriminations by features or characteristics and ensure an honest system. Interpretability also enhances **privacy** since by checking why the model gives predictions, one can ensure that there is no information leakage. Additionally, it also increases **reliability** because we check that predictions are consistent and that any change in the input does not affect our system's output. Interpretability also improves trust, increases social acceptance and consequently, users are more prone to rely on intelligent systems offering interpretability. Furthermore, governments and different entities require all those characteristics to fulfil guidelines and various legal aspects. For instance, the European Commission proposed the Ethics Guidelines for Trustworthy AI [1] as part of its AI strategy.

How do we classify interpretability methods?

There are different criteria to classify machine learning interpretability techniques. Those criteria build different taxonomies that specify the state-of-the-art. Such criteria can be if the interpretability method is local or global, intrinsic to the model, or is applied post hoc. To further examine the current taxonomies and criteria, refer to (Molnar, 2020).

However, this research work focuses on one specific criterion and taxonomy, the result of the interpretation method. Within this criterion, we can identify different types of methods (a summary of objectives, chapters and this taxonomy is presented in Figure 1.2). We consider this taxonomy the most complete and flexible to match our goal, offering a set of interpretability tools for KG-based models. Due to the variety of applications and use cases of KGs,

[1] https://ec.europa.eu/futurium/en/ai-alliance-consultation.1.html

the different categorisations given by this taxonomy are suitable in different scenarios. Thus, we present the following categorisations:

1. **Feature summary statistics**: many interpretation methods are capable of providing an indicator or statistic for a given feature (or all of them). Those statistics provide the importance or an indication of how a feature affects the model's prediction. This statistic is usually a real number or value.

2. **Feature summary visualisation**: given a feature statistic, it is common to develop a visualisation technique for such information. This categorisation belongs to feature statistics that are better understood when plotted or visually presented.

3. **Model internals**: this category belongs to the methods that make a model intrinsically interpretable. There are different ways to make a model intrinsically interpretable, e.g. forcing linear or sparse weights. These techniques try to clear the black-box perspective of many ML models. However, sometimes this category can be mixed with the "feature summary statistic"; for example, the value of the weights can also be considered a statistic.

4. **Data point**: in this category falls every method that provides a data point (an example) to understand a model better. For instance, given a prediction, a model can provide the most influential example or return a similar prediction with different features to detect the most relevant features in prediction-time.

5. **Intrinsically interpretable model**: a straightforward solution to make a model more interpretable is to approximate such model with an interpretable model.

This taxonomy is essential in this work. We ground this thesis on developing a set of tools corresponding to these different types of methods (except the intrinsically interpretable model). Each of them is suitable for different contexts and needs so that it can help precise scenarios.

Evaluating interpretability

Last, we follow (Doshi-Velez and Kim, 2017) who propose three primary levels for the evaluation of interpretability: **a)** application-level evaluation (real task) where the end-user directly tests the explanations of the model; **b)** human-level evaluation (simple task) is a simplified application-level evaluation where testers are not experts in the application domain and **c)** function level evaluation (proxy task), which does not require humans and testing is automatic. We use the function level evaluation task in this work, i.e. we do not require humans for evaluation. We employ state-of-the-art interpretability metrics and obtain an efficient and valuable evaluation.

In this context, we believe that developing new ways to increase interpretability (and thus fairness and transparency) in graphs and KGs will help society in multiple application areas. For that, we follow the given definitions and criteria on interpretability and look to develop various interpretability tools that can be helpful in different conditions.

1.2 Hypothesis, Objectives and Scope

Based on the current state of graphs and KGs and the need of developing new techniques to offer interpretability techniques in a wide sort of application areas, the hypothesis of this dissertation is:

> **Hypothesis.** For specific application needs and models applied to graphs and knowledge graphs, developing novel techniques focusing on output results makes it possible to increase users' interpretability at different levels.

Hence, this dissertation sets the following goal in order to validate the hypothesis above:

> **Goal.** To design and implement a set of interpretability tools for graphs and knowledge graphs that are suitable for specific scenarios and provide different output results.

This general goal can be achieved by addressing the following more specific and measurable objectives:

(**O1**) To study the current state of the art on machine learning applied to graphs and KGs and their interpretability techniques.

(**O2**) To design and implement a feature summary statistics method applicable to graph or KG-based ML models.

(**O3**) To design and implement a technique to make model internals (learned representations) more interpretable in KG-based ML models.

(**O4**) To design and implement a data point method applicable to KG-based ML models.

(**O5**) To provide an appropriate evaluation methodology for each implemented method and their corresponding tasks.

(**O6**) To analyse the impact of the implemented interpretability tools in the current state-of-the-art models in terms of performance.

We provide an overview of the relationship between the objectives, the chapters of this dissertation and the different types of methods developed in this work in Figure 1.2.

The resulting interpretability tools should also fulfil the following requirements:

1. *Audience variety*: the developed set of interpretability tools should offer solutions to every kind of user, experts or not.

2. *Model independence*: the provided interpretability tools should, if possible, be usable by different state-of-the-art models and not be model-specific.

The work presented in this dissertation does not deal with the following conditions:

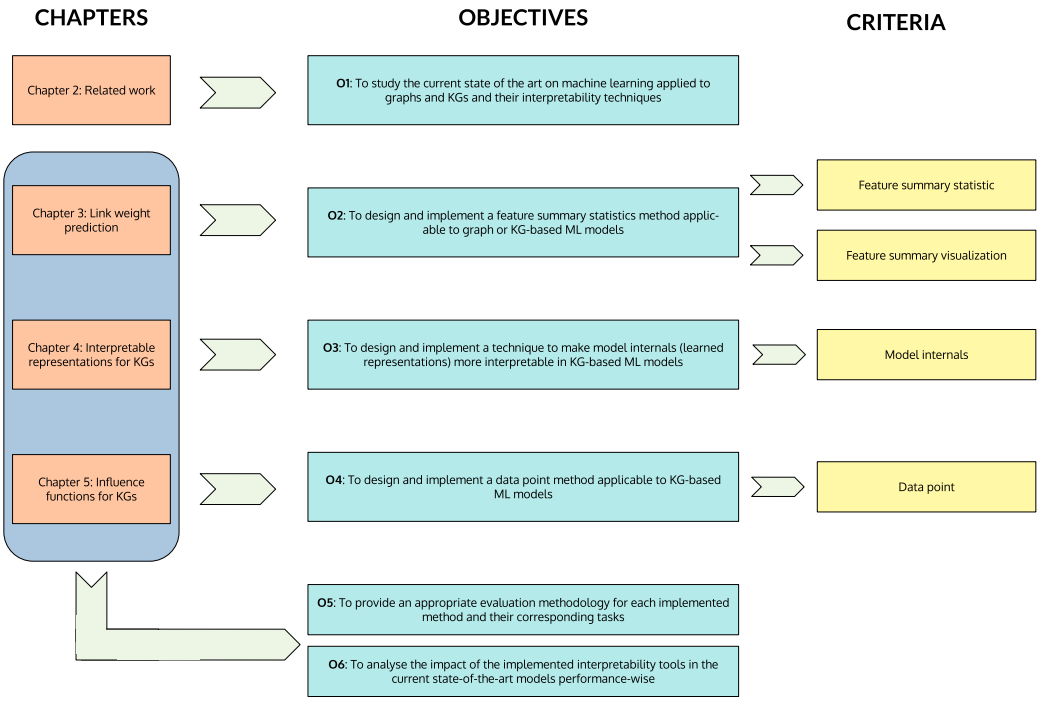


Figure 1.2: Summary of the thesis. Relationship between objectives, chapters and the developed interpretability methods.

1. Single solution: this work does not propose to find a unique interpretability solution for every graph and KG model. We consider finding a single method suitable for every scenario, use case and user type impossible. Instead, we propose to offer a set of tools that might fit specific contexts.

2. Temporal or dynamic graphs: the scope of this thesis are static graphs and KG-based models, and thus we do not deal with temporality. The used datasets correspond to the state-of-the-art KG models, which are static.

3. Evaluation with real people: although we propose to offer a set of interpretability tools for users, experiments with real people are out of the scope. The evaluation provided in this work corresponds to quantitative

analysis via proxy tasks that have been previously used in state-of-the-art methods.

Finally, it is essential to remark that this work's scope is to offer tested tools to increase interpretability in KG models. However, these tools are not meant to be off-the-shelf. The main contribution of this dissertation is the research techniques developed, not a production-ready library.

1.3 Research Methodology

In order to achieve the statement and derived goals presented in the previous Section 1.2, the next strategy was followed:

1. **Exploratory phase**: we explored the literature related to graphs, KGs and interpretability to build a solid theoretical background. This background has been the core knowledge of the whole research process. Revision of the literature is presented as an initial step of the research work. However, literature revision was performed throughout the entire research process iteratively.

2. **Definition of the scope and validation scenario**: after the first revision of the state-of-the-art was completed, the scope, context and motivation of this work were defined. Then, we defined the scenario in which the project would be developed. Such a scenario needed to be aligned with the objectives set and comprehend the research proposal's limitations and advantages.

3. **Specification, design and development of the interpretability tools**: at this stage, the solution that better fits the identified requirements had to be determined. We chose the interpretability tools to design and develop according to those requirements.

4. **Evaluation of the interpretability tools**: after the implementation of the tools, we continued with their evaluation. This evaluation was

needed to assess if the tools fulfilled the specified requirements (efficiently improved interpretability of the models). Furthermore, the evaluation had to verify their validity and applicability in real environments and contexts via proxy tasks.

5. **Conclusions, dissemination and writing the PhD dissertation**: the final task of the research process was the analysis of the obtained results. Such analysis led to deriving the conclusions and the contributions of this process. Finally, the research will be completed and refined with the present dissertation's submission and defence.

Figure 1.3 illustrates this research methodology, specifying the different phases and tasks and the iterative process followed to refine the initial conceptualization of the implemented solutions.

1.4 Main Contributions

The following scientific contributions can be found in this dissertation:

- A state-of-the-art model for link weight prediction in graphs. Such a model relies on a subgraph extraction process, the Weisfeiler-Lehman graph colouring method and Convolutional Neural Networks (CNNs). Additionally, the graph colouring method allows providing a feature summary statistic technique to increase the interpretability of the process. We present the model in Chapter 3.

- A new optimisation method to increase interpretability in KG embeddings. This process, grounded on cognitive arguments, internally introduces sparsity in the learned embeddings. Consequently, they are semantically more meaningful. The method is explained in Chapter 4.

- A novel method to find the most influential training samples (triples) for link prediction in KGs. We use influence functions to offer a data point method to interpret how much training samples influenced a model's prediction. We introduce the method in Chapter 5.

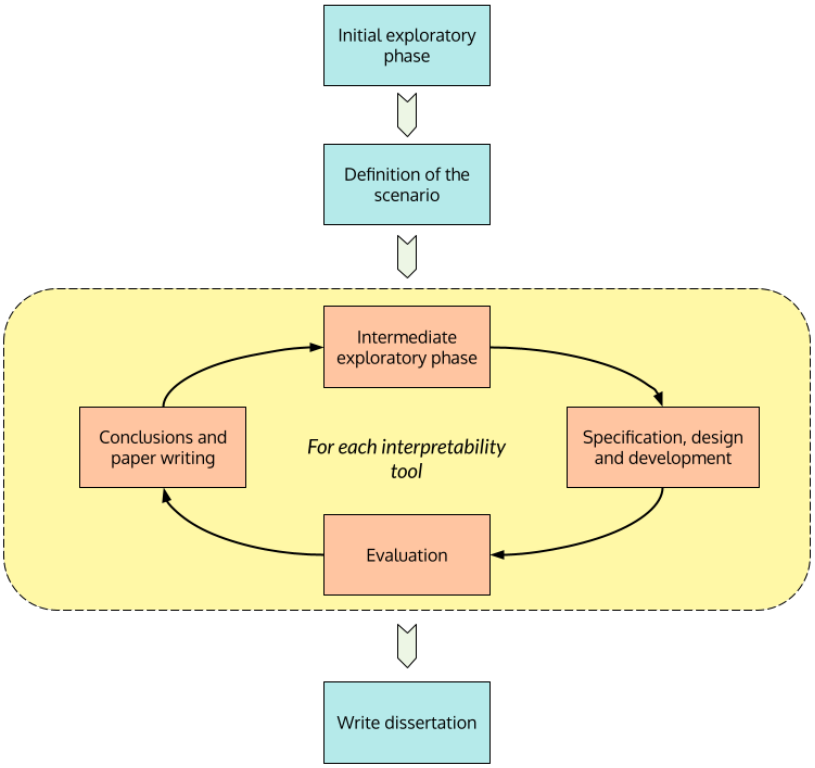


Figure 1.3: Schematic representation of the research methodology followed in this dissertation.

- An extensive evaluation for each of the models and methods. We perform such an evaluation in the model's corresponding tasks, e.g. link prediction, and evaluate interpretability. Moreover, we analyse the tradeoff between increasing interpretability and losing performance. Each of the chapters corresponding to novel methods, i.e. Chapters 3, 4, 5 and 6, contain their corresponding evaluation.

The following technical contribution is also an outcome of this thesis:

- A sparse version of the RESCAL (Nickel et al., 2011) model embeddings. These sparse embeddings are more semantically interpretable than the original ones.

1.5 Thesis Outline

This PhD dissertation is structured in 7 chapters. A summary of objectives and chapters is presented in Figure 1.2. The current section, *Chapter* 1, introduces the dissertation explaining its context, motivation, objectives, methodology and contributions.

Chapter 2 presents an analysis of the state-of-the-art on graphs and KGs, link prediction and interpretability.

Chapter 3 describes and evaluates our LWP-WL model. The model leverages the Weisfeiler-Lehmann graph colouring method, a subgraph extraction process and CNNs to become the state-of-the-art model in link weight prediction. Besides, the graph colouring method increases the interpretability of the whole process.

Chapter 4 presents a novel optimization technique and evaluation for interpretable KG embeddings. We propose introducing sparsity in the embeddings based on cognitive arguments to achieve more semantic meaning. We adopt a generalized version of the Regularized Dual Averaging algorithm to introduce sparsity.

Chapter 5 defines and evaluates the KGInfluence method. KGInfluence applies influence functions to find the most influential training instance for link prediction in Knowledge Graphs. This technique allows understanding why a model makes a particular prediction.

Chapter 6 summarises the main findings and contributions of this PhD dissertation and proposes future work.

"Those who do not move, do not notice their chains".

Rosa Luxemburg

CHAPTER

2

Related work

KNOWLEDGE study is an active research area in computer science. Several fields, such as machine learning or the semantic web, try to represent knowledge in a machine-readable way. Thus, knowledge representation is an essential field in order to create more intelligent and cognitive systems.

Graphs (Balakrishnan and Ranganathan, 2012) are powerful and flexible data structures representing objects as nodes and relationships among them as edges. Their flexibility makes them a helpful knowledge representation form. In ML, many areas such as social systems, ecosystems, biological networks or information systems leverage graphs to learn and perform different tasks, e.g. classification or link prediction.

In recent years the term Knowledge Graph has popularized. KGs, are entities that hold contextualized information in a graph-structured data model. Importantly, KGs are graphs that include semantics in their nodes and edges (often via topologies). Wikidata (Vrandečić and Krötzsch, 2014) or Freebase (Bollacker et al., 2008) are examples of cross-domain knowledge graphs and

Amazon's Product Graph [1] or Siemens Knowledge Graph[2] are instances of domain-specific KGs. These KGs are powerful tools that may act as enablers for transferring human knowledge to machines in an easy way.

However, most models for graphs and specially KGs, lack interpretability. This work aims to enhance interpretability for graphs and KGs by providing different tools. To do so, we ground on the link prediction task (the link weight prediction variant for standard graphs). This task is commonly used in the area as a base task and serves to develop models for many other tasks. Thus, in this chapter, we first review link prediction for graphs and KGs. Next, we also provide related work on interpretability and graph and KGs.

2.1 Link weight prediction in graphs

In chapter 3 we present a link weight prediction approach which achieves state-of-the-art results on the task. Furthermore, we provide a summary feature and visualisation technique that provides an interpretability tool for graph-based models. However, the related work on interpretability and graphs is presented in Section 2.3. We now present the research for link weight prediction in graphs.

Link weight prediction is a subtask of link prediction which focuses on predicting weight values for linked entities in networks. It is applicable in weighted graphs such as collaboration networks (Newman, 2003), transport networks (Li and Cai, 2004) or protein interaction networks (Jeong et al., 2001). The link weight prediction task (Hou and Holder, 2017; De Sá and Prudêncio, 2011; Fu et al., 2018; Liu et al., 2017) has been addressed in several previous works in the literature. For years, all of the approaches for this task relied on graph heuristics and latent features. Many works based on heuristics used node similarity (Zhao et al., 2015; Osaba et al., 2020), proximity measures (Murata and Moriyasu, 2007) or local rankings (Yang and Wang, 2020) to solve the task, however different alternative measures could

[1] https://blog.aboutamazon.com/innovation/making-search-easier

[2] https://www.siemens.com/innovation/en/home/pictures-of-the-future/digitalization-and-software/artificial-intelligence-industrial-knowledge-graph.html

be used, e.g. PageRank (Page et al., 1999). Methods relying on latent features such as Stochastic Block Model (Stanley et al., 2019; Aicher et al., 2014) and matrix factorization (Duma and Twala, 2018; Miller et al., 2009) also were presented. The models based on Stochastic Block Models and their variants were developed as generative models to find communities in the graph. Unlike the model presented in chapter 3, they are limited by design to perform link weight prediction since they find groups of nodes belonging to a community.

However, there was no approach based on deep learning for link weight prediction until recently. ModelR (Hou and Holder, 2017) provided a novel deep learning-based technique for the task. This approach relies on a neural network that receives the target nodes' ids as input, maps those ids to a feature vector and passes them through two fully-connected hidden layers to an output layer. Although ModelR is the most similar method to the presented in this work, our model applies a series of techniques that make it more advanced. The generalisation power of ModelR is limited by the randomness of the node ordering process, i.e. the nodes of the graph are assigned arbitrarily, and thus there is no consistency between different graphs when applying ModelR. Our method applies a novel graph labelling technique to overcome this issue. Furthermore, our method is more efficient and scalable than ModelR.

Nevertheless, graph-related works appeared in the literature in recent years (Hamilton et al., 2017b). Besides factorisation-based methods, two main approaches are considered in the literature: random-walk-based techniques and Graph Neural Networks (GNNs). Random-walk-based techniques (Grover and Leskovec, 2016; Perozzi et al., 2014) rely on performing random walks on the graph to generate node representations that are similar to each other when nodes co-occur on those random walks. In this work, we compare our model to node2vec (Grover and Leskovec, 2016) which is a widely used approach compared to random-walk-based methods in the literature. On the other hand, GNNs (Zhou et al., 2020) are networks capable of performing convolutional operations on non-Euclidean data domains. We compare our approach to GCNs (Kipf and Welling, 2016a), one of the most robust approaches in the field of GNNs. GCNs and our method share characteristics that make them comparable: a) they both apply a structural role to the nodes by using the

WL algorithm, and b) they both use convolutions to aggregate the nodes' information. The main difference between them relies on how they perform the convolution. GCNs do a neighbourhood aggregation, while our method uses special filters for a CNN (Convolutional Neural Network) layer (further explained in Section 3.2. In this manner, our method aggregates information in a more meaningful process for the link weight prediction task.

Various research studies have applied graph labelling algorithms for different graph-related tasks. Graph labelling was introduced as a technique in (Niepert et al., 2016) which presented an approach for learning a convolutional neural network for graph classification. Graph labelling has also been recently used for link prediction. The Weisfeiler-Lehman Neural Machine (Zhang and Chen, 2017) which inspired this work, proposed an algorithm based on the original WL algorithm for labelling non-weighted graphs. The SEAL framework (Zhang and Chen, 2018) also presented a novel node labelling method based on the radius from each node to the target link. These works focus on simple link prediction, i.e. they are not suited for weights. Their node labelling algorithm cannot handle weighted nodes and, thus, does not apply to this work's task. Furthermore, the method we present uses CNNs with special filters to further enhance the weight prediction, while Weisfeiler-Lehman Neural Machine and SEAL use other techniques.

Few approaches in the literature have used Convolutional Neural Networks over graph data, and none we know for link weight prediction. Due to the non-grid structure of graphs, these kinds of networks are not suitable for graphs. However, successful approaches have used CNNs for other graph-related tasks. ConvE (Dettmers et al., 2018) presented an approach for link prediction on knowledge graphs that applies a convolution over 2D-shaped embeddings of nodes. ConvKB (Nguyen et al., 2017) also employs a CNN to capture global relationships and transitional characteristics between entities and relations in knowledge bases and perform link prediction. However, these models were conceived for multi-relational data and cannot be applied for the link weight prediction task. Remarkably, they served as an inspiration for this work.

The experiments performed in this section present the LWP-WL algorithm as the new state-of-the-art model for link weight prediction. The approach

	Subgraph extraction	Node ordering	Representation learning	Interpretable
Block Models (Holland et al., 1983; Aicher et al., 2014)	no	no	no	no
ModelR (Hou and Holder, 2017)	no	no	yes	no
Node2Vec (Grover and Leskovec, 2016)	no (but applies random walks)	passive	yes	no
GCN (Kipf and Welling, 2016a)	no	yes	yes	no
LWP-WL (ours)	yes	yes	yes	yes

Table 2.1: Comparison of the state-of-the-art models and the most remarkable characteristics for link weight prediction.

provides three main improvements over the compared models: **a)** the extraction and ordering of nodes in subgraphs which provides a structural organisation pattern for the model, **b)** the performance and scalability obtained from the use of specific CNNs for the link weight prediction task, and **c)** interpretability due to the graph colouring algorithm and the subgraph extraction process. Different models in the literature provide one or the other improvements, but none provide all of them. We further present an in-depth comparison of the most notable models for link weight prediction in Section 3.3. Additionally, 2.1 provides a brief overview of the state-of-the-art models and the most remarkable characteristics for link weight prediction.

2.2 Link prediction in Knowledge Graphs

In chapters 4 and 5 we present two different interpretability methods (model internal and data point methods) to increase interpretability in Knowledge Graph-based models. Link prediction is the core task we use to achieve those methods, however, the proposed methods can be extended to almost any KG-related task. We now provide an overview of link prediction in KGs.

Link prediction in KGs has been a broad topic of study in the research coomunity (Nickel et al., 2015). The rise of KGs and representation-based models led to the development of novel techniques for finding missing data in these KGs. The main idea is to leverage the facts within the KG to train

a model that predicts new facts or missing data. So, for instance, if a KG contains the facts: (*Paris, located_in, France*) and (*French Parliament, located_in, Paris*), a model can derive the fact (*Paris, capital_of, France*).

Although there have been different approaches to solving such an issue, Knowledge Graph Embeddings (KGEs) have been the most successful method (Wang et al., 2017). In summary, KGEs embed the concepts and relationships in the KG to continuous vectors, i.e. the embeddings. The core idea is that we can maintain the KG structure in the vector space while using gradient-based learning methods for training. Furthermore, those embeddings can be used for several tasks such as link prediction (Bordes et al., 2013; Wang et al., 2014b), relation extraction (Weston et al., 2013; Riedel et al., 2013), entity classification (Nickel et al., 2011, 2012) or entity resolution (Nickel et al., 2011; Bordes et al., 2014). Even though there are approaches that use more information than the one contained within a KG (Guo et al., 2015; Lin et al., 2015; Wang et al., 2014a, 2015), we focus only on embedding models that use the facts from the KG alone.

We now present the notation used for the KGE models and present the most notable ones.

Notation

Throughout this dissertation, we use a boldface lower-case letter $\mathbf{x}$ to represent a vector. A matrix, is represented by a boldface upper-case letter $\mathbf{X}$. We use an underlined boldface capital letter $\underline{\mathbf{X}}$ to represent a three-mode tensor. The ijk-th entry of a tensor is denoted as $[\underline{\mathbf{X}}]^{ijk}$. We further use $\underline{\mathbf{X}}^{[i,:,:]}$, $\underline{\mathbf{X}}^{[:,j,:]}$ and $\underline{\mathbf{X}}^{[:,:,k]}$ to denote the i-th, j-th, and k-th slice along the first, second, and third mode of a tensor, respectively.

Let $\circ : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ denote the Hadamard product between two vectors, i.e.:

$$[\mathbf{a} \circ \mathbf{b}]_i = [\mathbf{a}]_i \cdot [\mathbf{b}]_i.$$

For the KG, let $\mathcal{E}$ denote the set of all entities and $\mathcal{R}$ the set of all relations present in a knowledge graph. We denote the collection of triples (facts) in a KG as $\mathcal{D}$ and each triple is represented as $(s, r, o) \in \mathcal{D}$, with $s, o \in$

$\mathcal{E}$ denoting subject and object entities respectively and $r \in \mathcal{R}$ the relation between them. We use n_e and n_r to denote the number of entities and relations within the KG. The corresponding embeddings for the fact (s, r, o) are denoted as $(\mathbf{e}_s, \mathbf{r}, \mathbf{e}_o)$ each $\in \mathbb{R}^d$, where d is the embedding dimensionality (However, the relation is rarely represented as a vector). In some cases d can vary between entities and relation embeddings, we will denote d_e and d_r for entity and relation size when corresponds.

In the link prediction task the objective is to learn a scoring function ϕ which scores, $s = \phi(s, r, o) \in \mathbb{R}$, whether a triple is true or false. To complete the task, we observe a subset of all true triples, aiming to correctly score all the missing ones.

Furthermore, we can naturally represent a KG and all its triples $(\mathcal{E} \times \mathcal{R} \times \mathcal{E})$ in a third-order tensor $\underline{\mathbf{Y}} \in 0, 1^{n_e \times n_r \times n_e}$, where n_e and n_r are, respectively, the number of entities and relationships within the KG. Thus, we are building a tensor whose entries correspond whether a fact exists in the KG

$$y_{ijk} = \begin{cases} 1, & \text{if the triple } (s, r, o) \text{ exists} \\ 0, & \text{otherwise} \end{cases}$$

Refer to Figure 2.1 for a visual representation of a tensor based KG.

Knowledge Graph Embedding models

Once the notation was presented, we now provide an overview of the KGE models. A KGE model commonly has three steps: 1) representing the entities and relationships, 2) defining a scoring function and 3) learning those entities and relation representations, i.e. the embeddings.

For the first step, entities are usually represented as deterministic points in a vector space (Nickel et al., 2011; Bordes et al., 2013; Wang et al., 2014b). However, other approaches have proposed alternatives, such as using hyperbolic embedding spaces (Nickel and Kiela, 2018) or Gaussian distributions (He et al., 2015). Relationships are generally understood as operations (between the entities) in the vector space. Those operations can be represented in different manners: vectors (Bordes et al., 2013), matrices (Bordes et al., 2014), tensors (Socher et al., 2013)...

The second step is to define a scoring function $s = \phi(e_s, r, e_o) \in \mathbb{R}$. The output of the scoring function s indicates the plausibility of the fact. The higher the score, the more likely the fact is true. The categorisation provided by (Wang et al., 2014b) divide the KGE models into two groups: translation distance models and semantic matching models. The categorisation criteria reside in their scoring functions. The translation distance models use distance-based functions, and the semantic matching models use similarity-based.

Finally, the third step consists in optimising the model to maximise the number of true facts to predict. KG optimisation is a whole research area and includes assumptions whether a KG is complete or not, various optimisation functions, strategies for the generation of false triples and hyperparameter tuning. Refer to (Wang et al., 2017) and (Ruffinelli et al., 2019) for more details on KG optimisation.

Different approaches were presented for KGE models, e.g. models where the scoring function relies on translation distance such as TransE (Bordes et al., 2013), TransH (Wang et al., 2014b) or RotatE (Sun et al., 2019) or neural network-based architectures like the Neural Tensor Network (Socher et al., 2013) or ConvE (Dettmers et al., 2018). However, in this work, we focus on tensor factorisation-based models. A tensor is a multidimensional array. For instance, a first-order tensor is a vector, and a second-order tensor is a matrix. Knowledge Graphs can be viewed as a third-order tensor, as shown in 2.1. Tensor factorisation or decomposition (Sidiropoulos et al., 2017; Kolda and Bader, 2009) is then used to obtain factorised representations of rows and columns of the KG. These rows and columns correspond to the entities and relationships within the KG. Hence, the obtained factorised representations are the pretended embeddings. Tensor factorisation models have demonstrated strong performance for building KGEs and are widely used in the research community. Thus, we select them to develop and evaluate the interpretability techniques presented in this thesis.

Various tensor factorisation models for link prediction have been proposed in the literature; we present a few of the most important ones:

RESCAL (Nickel et al., 2011) is a bilinear model which associates entities with a vector that captures latent (non-observable) semantics. Relations are

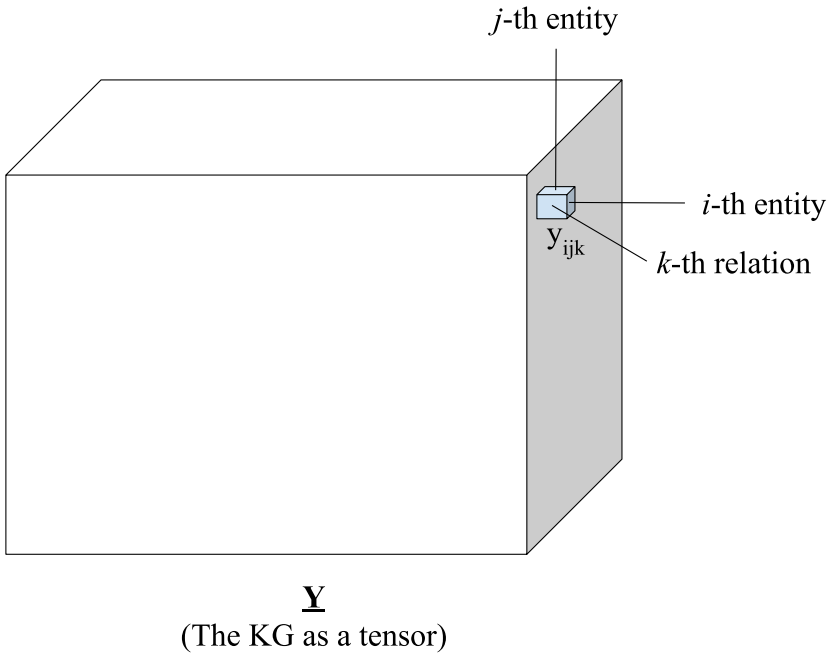


Figure 2.1: Tensor representation of a KG ($\underline{\mathbf{Y}}$). Where the i and j axes correspond to entities and the k axe to relationships. In each entry, a triple (y_{ijk}) is registered as true or falso (0 or 1).

represented as matrices that model pairwise interactions between the latent factors. The scoring function is the following

$$\phi(\mathbf{e}_s, r, \mathbf{e}_o) = \mathbf{e}_s^T \mathbf{M}_r \mathbf{e}_o = \sum_{i=0}^{d-1} \sum_{j=0}^{d-1} [\mathbf{M}_r]_{ij} \cdot [\mathbf{e}_s]_i \cdot [\mathbf{e}_o]_j,$$

where $\mathbf{M}_r$ is the matrix of the relation. RESCAL was one of the first KGE method to be proposed and serves as the ground for many other models. In chapter 4, we present an sparse and non-negative adaption of the RESCAL model to achieve embeddings that are interpretable.

DistMult (Yang et al., 2014) is a specific case of RESCAL where relations are diagonal matrices. Therefore, relations only capture pairwise interactions

between the same dimensions. While this model requires fewer parameters, it cannot model asymmetric relations because of its diagonal property. The scoring function is

$$\phi(\mathbf{e}_s, r, \mathbf{e}_o) = \mathbf{e}_s^T \text{diag}(\mathbf{M}_r)\mathbf{e}_o = \sum_{i=0}^{d-1} [\mathbf{r}]_i \cdot [\mathbf{e}_s]_i \cdot [\mathbf{e}_o]_i,$$

where $\text{diag}(\mathbf{M}_r) = \mathbf{r}$ as the diagonal of a matrix can be represented as a vector.

ComplEx (Trouillon et al., 2017) is an extension of DistMult which outputs complex-valued embeddings, thus the embedding space correspond to $\mathbb{C}^d$. This property makes the model asymmetric, improving the performance in non-symmetric relations. Its scoring function is

$$\phi(e_s, r, e_o) = \text{Re}(\mathbf{e}_s^T \text{diag}(\mathbf{M}_r)\bar{\mathbf{e}_o}) = \text{Re}(\sum_{i=0}^{d-1} [\mathbf{r}]_i \cdot [\mathbf{e}_s]_i \cdot [\bar{\mathbf{e}_o}]_i),$$

where $\bar{\mathbf{e}_o}$ is the conjugate of $\mathbf{e}_o$ and $\text{Re}(\cdot)$ means taking the real part of a complex value.

TuckER (Balažević et al., 2019) is a linear model based on the Tucker decomposition (Tucker et al., 1964) and previous linear models can be interpreted as exceptional cases of this. Its power relies on a core tensor which allows multi-task learning by sharing knowledge between entities and relations. The corresponding scoring function is

$$\phi(e_s, r, e_o) = \mathcal{W} \times_1 \mathbf{e}_s \times_2 \mathbf{r} \times_3 \mathbf{e}_o,$$

where $\mathbf{r} \in \mathbb{R}^{d_r}$ are the rows of the relation embedding matrix which form the relation embedding vector. $\mathcal{W} \in \mathbb{R}^{d_e \times d_r \times d_e}$ represents the core tensor derived from the Tucker decomposition (Tucker et al., 1964).

In table 2.2 a summary with the presented models is shown. We provided the most remarkable models, the reader can refer to (Wang et al., 2017) or (Ruffinelli et al., 2019) to check more KGE models and methods.

All of the presented models (tensor factorisation-based and others) produce embeddings which are not interpretable. The obtained embeddings are

Method	Entity emb.	Relation emb.	Scoring func.	Space complexity
RESCAL	$\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}^d$	$\mathbf{M}_r \in \mathbb{R}^{d \times d}$	$\mathbf{e}_s^T \mathbf{M}_r \mathbf{e}_o$	$\mathcal{O}(n_e d + n_r d^2)$
DistMult	$\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}^d$	$\mathbf{r} \in \mathbb{R}^d$	$\mathbf{e}_s^T \text{diag}(\mathbf{M}_r) \mathbf{e}_o$	$\mathcal{O}(n_e d + n_r d)$
ComplEx	$\mathbf{e}_s, \mathbf{e}_o \in \mathbb{C}^d$	$\mathbf{r} \in \mathbb{C}^d$	$\text{Re}(\mathbf{e}_s^T \text{diag}(\mathbf{M}_r) \bar{\mathbf{e}_o})$	$\mathcal{O}(n_e d + n_r d)$
TuckER	$\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}_e^d$	$\mathbf{r} \in \mathbb{R}_r^d$	$\mathcal{W} \times_1 \mathbf{e}_s \times_2 \mathbf{r} \times_3 \mathbf{e}_o$	$\mathcal{O}(n_e d_e + n_r d_r)$

Table 2.2: Comparison table for the presented models.

useful and efficient in performing a set of tasks. Nevertheless, the embeddings are only vectors of real values that lack any meaning for humans. To solve such issue, in chapter 4 we provide an adaption of the RESCAL model to develop more interpretable embeddings. In chapter 5, we present a data point method for DistMult which outputs the most influential training fact for a new predicted triple. It is important to remark that even though we used RESCAL and DistMult for this work, this dissertation's interpretability methods can be extended to every other tensor factorisation-based method.

2.3 Interpretability in graphs and Knowledge Graphs

Interpretability in Machine Learning has widely been studied in the research community (Molnar, 2020; Carvalho et al., 2019; Gilpin et al., 2018). Nevertheless, it still remains one of the most critical challenges to overcome. Interpretability applied to graphs and KGs is no exception. However, interpretability methods are often classified and applied by the technique they were developed for rather than by data type (graphs, images, text...). For instance, much work on interpretability has been applied to Convolutional Neural Networks (Olah et al., 2018; Bau et al., 2017) and computer vision. Regardless, there also exist various methods that are model-agnostic (Goldstein et al., 2015; Lundberg and Lee, 2017; Ribeiro et al., 2016). In this context, we first introduce the interpretability works on graphs and present

various approaches to improve interpretability for KGE models. Furthermore, we also review the interpretability methods related to the specific techniques we propose in this thesis: CNN visualisation (feature summary and visualisation), sparse representations (model internals) and influence functions (data point method).

Graph-related ML has been mainly based on graph heuristics and node features such as the Katz index until recently (Zhao et al., 2015; Murata and Moriyasu, 2007; Yang and Wang, 2020). Many of those models are interpretable; thus, they do not require ad-hoc interpretability methods. However, recently approaches grounded on Graph Neural Networks (GNNs) (Kipf and Welling, 2016a; Wu et al., 2020) have achieved state-of-the-art results in many tasks. Unfortunately, GNN-based models are not interpretable. To increase their interpretability different methods have been proposed (Veličković et al., 2017; Ying et al., 2019; Huang et al., 2020).

Nevertheless, in this work (chapter 3), we present a link weight prediction method based on the Weisfeiler-Lehman (WL) colouring technique (Weisfeiler and Lehman, 1968). We extract a subgraph and then feed the nodes within the subgraph to a CNN to perform the prediction. On the one hand, relying on the WL method allows the approach to be intrinsically interpretable as the colouring process orders nodes by their role within the subgraph. By looking at the order of the nodes, we can understand how a graph behaves. On the other hand, we apply a visualisation method to the CNN to better interpret the whole process. The WL method has widely been used in the research community (Zhang and Chen, 2017; Shervashidze et al., 2011). It has also been related to GNNs (Kipf and Welling, 2016a). There exist also various approaches to increase the interpretability of CNNs (Olah et al., 2018; Bau et al., 2017). Regardless, the combination of the WL technique, the subgraph extraction process and the use of a CNN makes our approach unique and highly interpretable.

Authors in the literature have proposed different methods for interpreting KGEs. In (Sengupta et al., 2017) authors induce interpretability in the embeddings by adding a measure of coherence as a regularisation term of the overall loss function using additional entity co-occurrence statistics from the text.

The coherence measure allows automated evaluation of the quality of topics learned by topic modelling methods by using additional Point-wise Mutual Information (PMI) for word pairs. In (Gusmão et al., 2018), authors adopt "pedagogical approaches" to interpret KGEs and extract weighted Horn rules to increase their interpretability. In the work (Barbieri et al., 2014), authors present a model that does predict links and decides whether it is a "topical" or a "social" link. Researchers in (Xiao et al., 2016) propose a semantic representation method for knowledge graphs with a two-level hierarchical generative process that globally extracts many aspects and then locally assigns a specific category in each aspect for every triple. A different work, (Xie et al., 2017), presents a translational model (ITransF) which learns associations between relations and concepts via sparse interpretable attention vectors. Authors in (van Engelen et al., 2016) use the topological properties of a network to explain the contribution of particular categories of features in link prediction. Finally, in (Allen et al., 2019) authors analyse the latent structure and semantic meaning of KGEs based on theoretical interpretations of word embeddings. We present a summary table for graph and KG specific interpretability works in 2.3.

As previously mentioned, in chapter 4 we introduce a method to make the embeddings obtained from KGs interpretable. This method is categorised as a model internal method within the followed taxonomy (see Section 1.2). In summary, the method achieves sparse and non-negative embeddings with semantic meaning. There are several properties that differentiate our approach: *a)* it directly increases the interpretability in the embeddings, *b)* the process is done at training time, and *c)* it does not use external knowledge in the process. Although no similar works were presented in the context of KGs, comparable approaches were developed in Natural Language Processing.

Techniques for interpreting word embeddings by sparsity mechanism have also been proposed. In (Murphy et al., 2012), authors apply a sparse non-negative matrix factorisation model to produce word embeddings. In the work (Faruqui et al., 2015), they use sparse coding techniques to derive sparse word embeddings from dense word representations. Similar to our work, authors from (Sun et al., 2016) modify the Continuous Bag of Words model and add

	Data domain	Interpretability technique
(Zhao et al., 2015; Murata and Moriyasu, 2007; Yang and Wang, 2020)	Graphs	Intrinsically interpretable by Katz index.
(Veličković et al., 2017; Ying et al., 2019; Huang et al., 2020)	Graphs	Attention mechanism and model explanation applied to GNN.
(Sengupta et al., 2017)	KGE	Additional entity co-occurrence statistics from text to induce coherence in the embeddings.
(Barbieri et al., 2014)	KGE	Horn rules.
(Xiao et al., 2016)	KG	Semantic representation for explaining each triple.
(Xie et al., 2017)	KGE	Sparse interpretable attention vectors.
(van Engelen et al., 2016)	KG	Use of the topological properties of a network to explain link prediction.
(Allen et al., 2019)	KGE	Latent structure and semantic analysis of the embeddings.

Table 2.3: Summary of the works for improving graph and KG interpretability.

the l_1 regulariser in online training by employing the Regularized Dual Averagaging method. Authors in (Subramanian et al., 2018) present a k-sparse denoising autoencoder to produce a sparse non-negative high dimensional projection of word embeddings. Finally, (Panigrahi et al., 2019) produces word embeddings by an underlying LDA-based generative model, which helps generate sparse vectors. These approaches are closer to the approach presented in this work (chapter 4) since they produce sparse embeddings. However, they all focus on word embeddings which are developed differently. Our approach works on tensor factorisation techniques that can not be used to build word embeddings.

Additionally, in chapter 5 we present a data point method which outputs the most important training samples for a given prediction. This approach uses Influence Functions (IFs) to obtain the data points. To our knowledge, IFs had never been used for KGs. However, IFs have been used in other

Machine Learning areas to increase the interpretability of many applications. Authors from (Koh and Liang, 2017) were the first to propose using IFs to increase interpretability on linear models and convolutional neural networks. In (Cheng et al., 2019) authors propose a novel fast influence analysis technique to understand the prediction made by latent factor models in recommendation systems. Authors in (Kong and Chaudhuri, 2021) use IFs to sample influential instances in Variational Autoencoders.

"Toxicidad fuera, mala vibra fuera."

- Ibai Llanos

CHAPTER

3

Leveraging structural nodes in graphs for link weight prediction

This chapters presents the work which inspired and grounded the thesis. Derived from the proposed approach, a feature summary statistic method is provided. To be concrete, we develop a new link weight prediction approach named **L**inked **W**eight **P**rediction **W**eisfeiler-**L**ehman, LWP-WL. The approach first extracts an enclosing subgraph from the neighbourhood of the target link. Then, the enclosing subgraph is fed into a neural network model (where the first layer is a Convolutional Neural Network (CNN)) that predicts the output value for the target link. The network is trained on target links to learn a link weight prediction model.

Although the model was thought of with performance in mind, we identified the interpretability derived from the WL algorithm and the subgraph extraction process. Such interpretability inspired us to develop the other tools within this thesis.

The chapter is divided into four sections: Section 3.1 analyzes the role of the graph colouring algorithm and the graph subtraction process as a feature summary statistic method. Following, Section 3.2 explains the proposed approach for the link weight prediction. Next, Section 3.3 illustrates the experiments and results the approach. Finally, Section 3.4 provides an overview of the interpretability achieved by the method.

3.1 Graph colouring and subgraph extraction for interpretable link weight prediction

This chapter presents the work which served as inspiration and motif for the thesis. When analysing the link weight prediction approach we herein present, we realised that the approach is interpretable due to the graph colouring algorithm and the subgraph extraction process. The link weight prediction approach was then categorised as a feature summary statistic method within the taxonomy followed in Section 1.1.

The proposed link weight prediction approach, LWP-WL, learns graph structure features from subgraphs. First, we present a graph labelling technique for weighted graphs that allows a neural network model to predict the weights of links. Additionally, this neural network contains a Convolutional Neural Network layer that operates over the extracted subgraphs' adjacency matrices. This CNN applies special filters taking advantage of the graph representation to enhance the output prediction further. Most importantly, LWP-WL can learn graph structure features in different graphs, providing a solid baseline for an extensive set of applications.

The two main aspects of the approach which provide a high grade of interpretability are the graph colouring algorithm, i.e. the Weisfeiler-Lehman variant, and the graph subtraction process.

Graph colouring[1] is the procedure of assigning colours (or any identification element) to each vertex of a graph so that no adjacent vertices get the same colour. Graph colouring algorithms such as the WL method assign a

[1]Note that through the dissertation, we use the term graph colouring, but precisely we refer to the process of vertex labelling.

unique set of features for most graphs. This is important because every node is assigned a feature uniquely describing its role in the graph. Then, a model can exploit each node's unique feature (colour) to perform almost any task, i.e. node classification. Additionally, the colour given to each node describes its role in the graph. Thus, nodes that fulfil similar roles have almost identical "colours". We exploit this property to improve the interpretability of our model.

In this work, the process of subgraph extraction consists in extracting enclosing subgraphs for target links. The process allows avoiding the need to input a representation for the whole graph, notably increasing the scalability of the whole model. By extracting the subgraph, only the graph's most relevant nodes and edges are considered for the link weight prediction.

The combination of both the colouring algorithm and the subgraph extraction process lets to obtain a feature representation of every node in the graph in a scalable manner. Then, the comparison of the representations obtained can be used to acquire an idea of the similarity of the roles between subgraphs, increasing the interpretability of the whole process.

This chapter presents a link weight prediction approach that achieves state-of-the-art results on the task. The proposed model is scalable, efficient, and fairly interpretable by leveraging a graph colouring algorithm and a subgraph extraction process.

3.2 Approach and methodology

In this section, we introduce our approach for link weight prediction, LWP-WL. It does contain the following steps:

1 Subgraph extraction.

2 Subgraph node ordering.

3 Feeding the ordered subgraph's adjacency matrix into the model.

A summary figure of the whole approach is shown in Figure 3.1.

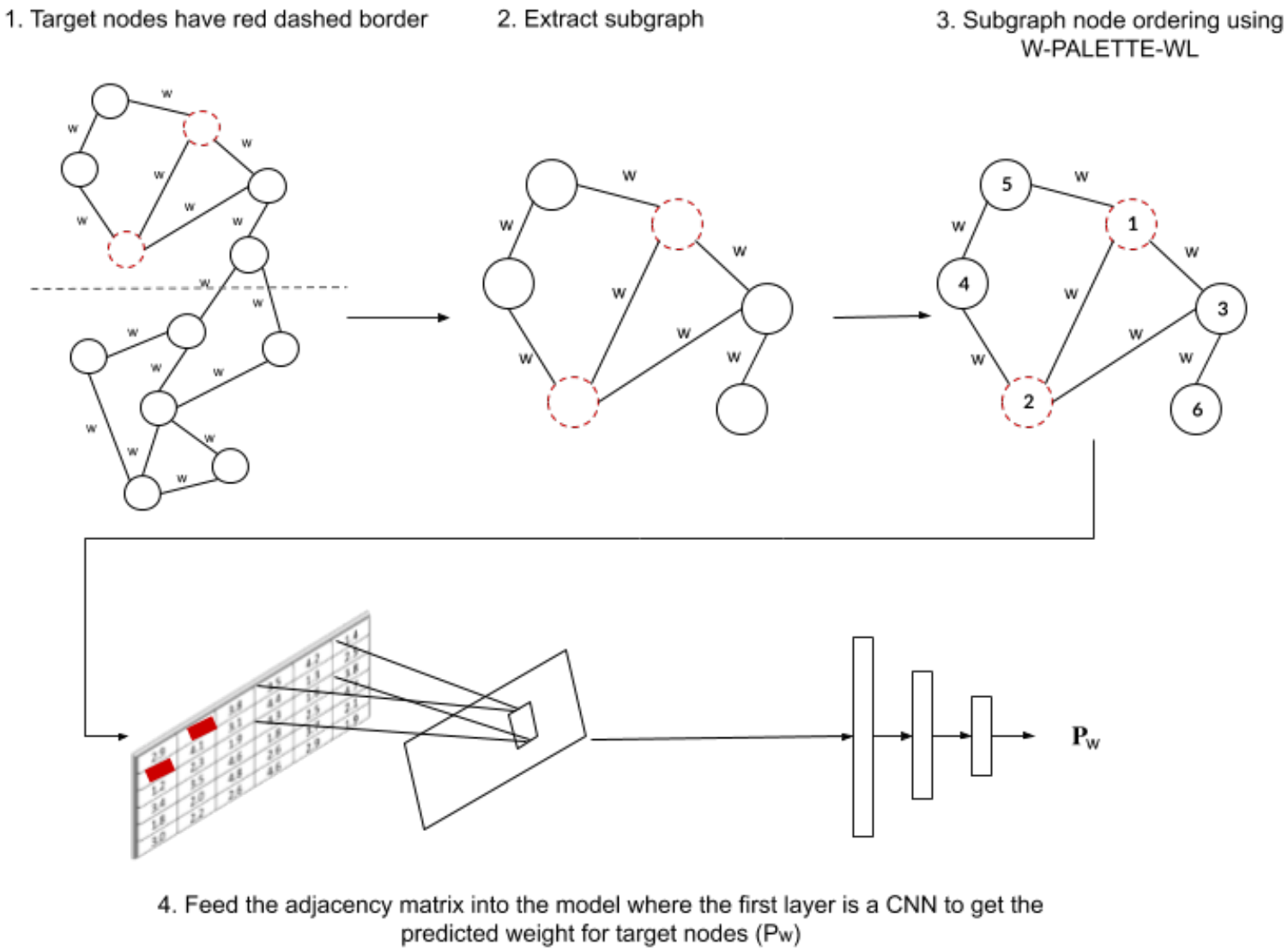


Figure 3.1: Summary of the steps for the LWP-WL link weight prediction framework.

3.2.1 Subgraph extraction

The first step of the approach is to extract enclosing subgraphs for target links. We identify the target nodes of a target link as the pair of nodes that perform the connection for the specific link.

For the target link, e_{v_x,v_y}, we extract a set of nodes of a user-defined size K. Starting from the nodes that hold the target link - i.e. v_x and v_y - we apply the Breadth-First-Search (BFS) (Lee, 1961) strategy to add nodes to the subgraph. Although we opted for BFS, other strategies can be applied. In this way, we start adding nodes from the 1-hop neighbourhood of the target nodes and then we iteratively continue to the next neighbourhood until we reach the number of nodes K and finally get the set of nodes for the subgraph, L_K. In order to obtain the representation for the target link, we add the same amount of nodes for each of the target nodes, so we apply $BFS(v_x)$ to get a set of nodes L_{v_x} of size $K/2$ and $BFS(v_x)$ to get a set of nodes L_{v_y} of size $K/2$. Finally, we combine both sets $L_{v_x} \cup L_{v_y}$ to get the final set L_K. We ensure that nodes from both sets are different so $L_{v_x} \cap L_{v_y} = \emptyset$. We provide the algorithm for the process in Algorithm 1.

The enclosing subgraph extraction process provides two remarkable additional features to the LWP-WL approach:

1. It can extract a **similar context representation for all links**, even from different graphs. By applying the same strategy for the subgraph extraction on every link, the model will, later on, be able to generalize between different graphs, nodes and links.

2. It is scalable. The extraction of a fixed size K subgraph for every target link **avoids the need to input a representation for the whole graph** such as the adjacency matrix. In this manner, the input to the LWP-WL approach will always be of size $K \times K$ while avoiding inputs of size $V \times V$, which are not scalable as they grow with the number of nodes within the graph.

Algorithm 1: Subgraph extraction process. *BFS* stands for breadth-first search. $\Gamma(n)$ stands for n-hop neigbourhood.

input: graph $G = (V, E)$, target nodes (x, y), subgraph size K
output: subgraph $G(L_K)$ for (x, y)
$L_k = []$
$L_x = [v_x]$
$L_y = [v_y]$
$n = 1$
while $|L_k| < K$ **do**
  for *node in* L_x **do**
    neighbourhood = $BFS_n(node)$
    for *neighbour in neighbourhood* **do**
      if $|L_x| < K/2$ **then**
        add *neighbour* to L_x
      end
    end
  end
  for *node in* L_y **do**
    neighbourhood = $BFS_n(node)$
    for *neighbour in neighbourhood* **do**
      if $|L_y| < K/2$ **then**
        add *neighbour* to L_y
      end
    end
  end
  n = n + 1
  $L_k = L_x \cup L_y$
end
return subgraph $G(L_K)$

3.2.2 Subgraph node ordering

The second step of the approach is to order the extracted enclosing subgraph. So, we need a function that orders the set of nodes extracted from the enclosing subgraph.

$$l : L_K \rightarrow O_K \tag{3.1}$$

The purpose of the ordering is to provide a **consistent manner of labelling nodes** in a way that those nodes that have similar topological features concerning the subgraph are labelled similarly. Then, we can train a model that receives as input a representation of the ordered subgraph, namely, the adjacency matrix.

We adapt the WL graph labelling method to apply it for weighted graphs using the 1-dimensional Weisfeiler-Lehman (WL) algorithm. Authors from (Zhang and Chen, 2017) also developed a node ordering algorithm based on WL. However, their approach was not developed for weighted graphs. We now mention the requirements for the graph labelling algorithm as they did in their work:

1. The graph labelling algorithm must provide **similar labels to nodes that have similar topological features** within the enclosing subgraph.

2. It must preserve the **topological directionality towards the target link**, i.e. the order of the nodes has to be conditioned by the target nodes, and the distance to them must be reflected in the ordering.

Both requirements are needed for developing models that use the nodes ordering for learning. We adapted the algorithm for weighted graphs because the WL method does not fulfil the second requirement.

We propose Weighted-WL (W-WL), a graph labelling algorithm whose objective is to order the set of nodes from the extracted subgraph, L_K from 1 to K. However, as we want to keep the topological directionality towards the target link, the target edges will always be given orders 1 and 2. First, nodes are assigned initial colours (rankings) according to the sum of their

Algorithm 2: The W-WL Algorithm.

input: enclosing subgraph $G(L_K)$ centered at target link e_{v_x,v_y}, which is extracted by Algorithm 1
output: ordered set of nodes (O_K) from $v \in L_K$
$L_1 = x, L_2 = y$
calculate $d(v) := d(v,x) + d(v,y)$ for all $v \in L_K$
get initial colours $c(v) = f(d(v))$
while $|O_K| < |L_K|$ **do**
  calculate signature strings $s(v)$ for all $v \in L_K$
  add $lowest(s(v))$ to O_K
end
return: O_K

shortest paths to the target nodes, v_x and v_y (edges' weights are computed to calculate the shortest path). Afterwards, we assign a **signature string** to each of the nodes by using the Weisfeiler-Lehman algorithm, s_v. For this, we concatenate the initial colour of each node with the initial colours of its neighbours lexicographically ordered from minimum to maximum, generating a unique signature string. Then, the **lexicographically lowest string signature** corresponds to the next node for the ordered list. Finally, we iterate this process until every node is assigned a number. Because we have considered weights, it is unlikely to have ties in signature strings; otherwise, they are solved by choosing randomly. The process is defined in Algorithm 2. For a figure depicting the first steps of the process, see Figure 3.2.

Once the node ordering process has finished and we have obtained the ordered set, we extract the adjacency matrix for the subgraph, $A_{s(O_K)}$, so the matrix's rows and columns correspond to the ordered set. This adjacency matrix is then used as input to a neural network. Before feeding it, we make sure that the values of the matrix that represent the weight for the link are not visible to the model, $A_{1,2}$ and $A_{2,1}$ as they are the labels for the model.

3.2.3 Model

After obtaining the subgraph and the list of ordered nodes, we get an adjacency matrix $A(O_K)$; the next step is to train a classifier. For this classifier,

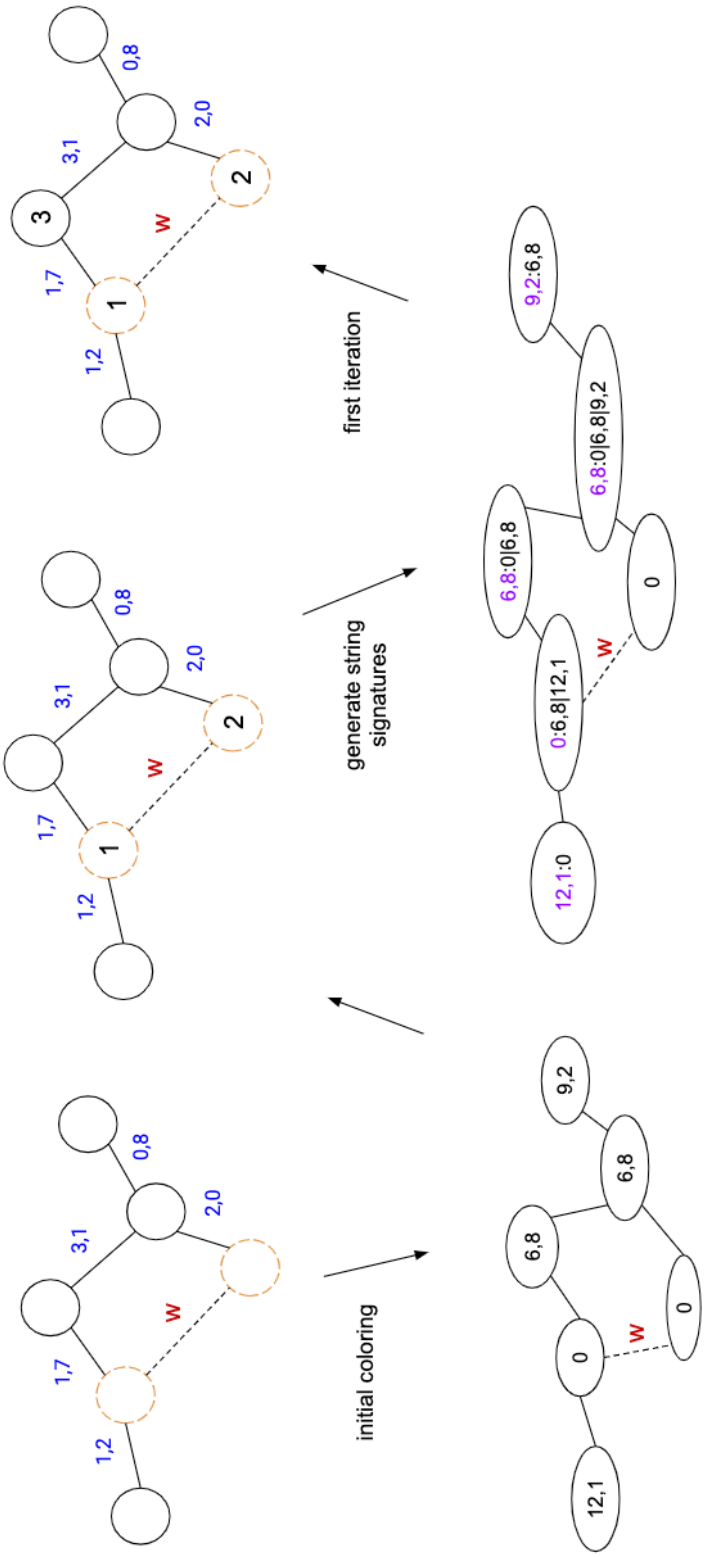


Figure 3.2: First iteration of the W-WL algorithm. We want to predict the weight (w, coloured in red) of the link for the target nodes (dashed and coloured in yellow). First, we generate initial colours for each node. Then we generate a string signature for the nodes. Finally, we evaluate all string signatures and select the lexicographically lowest one. Target nodes get the 1 and 2 numbers to preserve the topological directionality.

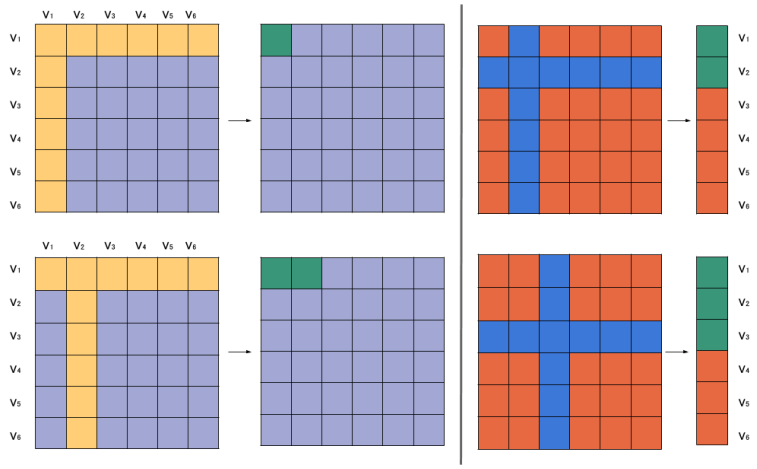


Figure 3.3: The special filters applied to a $K = 6$ size adjacency matrix. **Left**: edge to edge filter applies a convolution over the neighbouring edges for the target link. **Right:** edge to node filter that applies a convolution over the neighbouring edges of each node in the adjacency matrix.

we use a neural network.

As our main contribution, we use a Convolutional Neural (CNN) in the first layer of the neural network. This layer applies special filters adapted to the graph representation used, the adjacency matrix.

3.2.3.1 Edge to edge filter.

For a given edge e_{v_x,v_y} this filter applies a convolution over edges that are neighbours of both v_x and v_y. In this manner, the filter learns about the edge's neighbourhood and the patterns among the nodes.

3.2.3.2 Edge to node filter.

For a given node v_x, we apply a convolution for every neighbour edge of the node, unlike the edge to edge filter; it does focus on nodes learning patterns for their neighbourhood.

A figure depicting these filters is shown in Figure 3.3. Then, the output of the CNN is flattened and goes through a set of fully-connected hidden layers to an output layer that performs a linear regression for the predicted weight.

For training a graph $G = (V, E)$, we build the samples by selecting all the links and their weights $e, w_e \in E$. Then, we divide those samples into training, validation and test sets. Next, we extract all subgraphs for each edge sample, and we train the system by providing as input the ordered adjacency matrix of the enclosing subgraph for the target link, $A_e(O_K)$, and the weight of that edge as the output w_e.

3.3 Experiments and results

In this section, we conduct a set of experiments to evaluate the performance of the LWP-WL approach. We first describe the datasets used for the experiments. Then, we provide an overview of the setup for those experiments. Next, we run a set of experiments comparing our approach with several methods, including the state-of-the-art, ModelR, and address the scalability and complexity aspects of our approach. Finally, an ablation study is presented, demonstrating how gradual enhancements to the LWP-WL approach impact the performance.

The conducted experiments aim to solve the next questions:

1. How well does LWP-WL perform in the link weight prediction task compared to the state-of-the-art models? Do the best performing models share any common characteristics?

2. How scalable is LWP-WL compared to the best models? Does it perform well in big graphs?

3. What is the impact of structural roles of nodes and, thus, the Weighted-WL graph labelling algorithm? And the special edge-to-edge and edge-to-node filters?

Each of those corresponding to the following subsections: *Comparison*, *Scalability and Complexity* and *Ablation Study*.

3.3.1 Datasets and baselines

In this work, we have used six networks to evaluate our model. We first select the standard four datasets used for link weight prediction included the state-of-the-art approach (Hou and Holder, 2017) to perform a comparison. Additionally, we also add two extensive networks further to analyse the performance of our model in big graphs. These weighted graphs are real-world networks that belong to different domains and applications and report significant values for the task.

- *Airport* (Colizza et al., 2007) contains the busiest airports in the US where nodes represent airports, links connections between them and weights correspond to the number of passengers travelling from one airport to the other.
- *Collaboration* (Pan et al., 2012) represents world nations as nodes, links and their weights are the number of papers written by authors from the two nations.
- *Congress* (Porter et al., 2005) is the network of the 102nd US Congress committees where nodes represent members and links, and their weights interlock value of shared members from the two committees.
- *Forum* (Opsahl and Panzarasa, 2009) is a student social network where nodes are users and links and weights the number of messages sent from one student to the other.
- *Geom* (Batagelj and Mrvar, 2014) authors collaboration network in computational geometry obtained from the Computational Geometry Database geombib.
- *Astro* (Newman, 2001) a network of coauthorships between scientists posting preprints on the Astrophysics E-Print Archive.

As shown in the dataset statistics summary in table 3.1, the six datasets are different in the number of vertices and edges and provide different values

Name	$\|V\|$	$\|E\|$
Astro	16,706	121,251
Airport	500	5,960
Collaboration	226	20,616
Congress	163	26,569
Geom	7,343	11,898
Forum	1,899	20,291

Table 3.1: Summary of the datasets used for experiments, $|V|$ represents the number of nodes and $|E|$ the number of links.

in their weights, providing a variety of settings for testing different models and performances.

Different baselines that have been proposed in the literature for this task are evaluated with these datasets:

- SBM (Stochastic Block Model) (Holland et al., 1983): conceived for graph clustering in unweighted graphs, this approach partitions nodes into L subsets $C_i \subset V$ where $i \in L \in \mathbb{N}$.

  Then, SBM fits parameters θ for building the adjacency matrix A where $A_{ij} \sim B(1, \theta_{C,C_j})$ follows a Bernoulli distribution:

  $$P(A|C,\theta) = \prod_{ij} \theta_{C_i,C_j}^{A_{ij}} (1 - \theta_{C_i,C_j})^{1-A_{ij}} \tag{3.2}$$

  , the log likelihood of A can be rewritten as an exponential family:

  $$\log P(A|C,\theta) = \sum_{ij} T(A_{ij}) \eta(\theta_{C,C_j}) \tag{3.3}$$

  where $T(A_{ij}) = (A_{ij}, 1))$ is the vector function of sufficient statistics of

the Bernoulli random variable and

$$\eta(\theta) = \left(\log \left(\frac{\theta}{1-\theta} \right), \log(1-\theta) \right) \tag{3.4}$$

is the vector function of natural parameters of the Bernoulli random variable.

Stochastic Block Models are generative models meant to find community structures in graph data. Therefore, they are limited by design to perform link weight prediction. While LWP-WL is a model conceived specifically to predict weights in graphs, SBMs in their standard version do exact recovery techniques without taking the link's weights into account.

- pWSBM (pure Weighted Stochastic Block Model) (Aicher et al., 2014): unlike SBM, it only uses link weight information. The main difference is that the adjacency matrix follows a normal distribution in its values, therefore $A_{ij} \sim N(\mu_{c,c_j}, \sigma^2_{c,c_j})$ and θ_{C,C_j} becomes the weight distribution parameter:

$$\theta_{C,C_j} = (\mu_{C,C_j}, \sigma^2_{c,c_j}) \tag{3.5}$$

and $T(A_{ij}) = (A_{ij}, A^2_{ij}, 1))$ becomes the vector function of sufficient statistics of the normal random variable, and

$$\eta(\theta) = \left(\frac{\mu}{\sigma^2}, -\frac{1}{2\sigma^2}, -\frac{\mu^2}{2\sigma^2} \right) \tag{3.6}$$

is the vector function of the natural parameters of the normal random variable. Then, it fits by

$$\log P(A|C,\theta) = \sum_{ij} \left(A_{ij} \frac{\mu_{C,C_j}}{\sigma^2_{C,C_j}} - A^2_{ij} \frac{1}{2\sigma^2_{C,C_j}} - \frac{\mu^2_{C,C_j}}{\sigma^2_{C,C_j}} \right) \tag{3.7}$$

pWSBM is a generalised version of SBM for weighted graphs. Although it shares the same strengths, it is suited for link weight prediction. The idea behind pWSBM is to find the latent structural roles of the nodes in the graph. To do so, the WSBM models each weighted edge as a draw

from a parametric exponential family distribution and recovers the whole graph and its weights. Our model, LWP-WL, conceives the same idea of finding latent structural roles in the graph more sophisticatedly. First, it finds the structural role by applying the W-WL labelling algorithm, which uses the nodes' topological features. It is essential to realise that W-WL does not try to fit each node into a block structure, so it is more flexible when finding latent structures. Second, the edge to edge and edge to node filters in the LWP-WL model allows predicting weights at an individual level for each of the edges, while the WSBM depends on group memberships and their parameters for the prediction.

- bWSBM (balanced Weighted Stochastic Block Model) (Aicher et al., 2014): it does mix the SBM and pWSBM approaches by using the pair (T_e, η_e) which denotes the family of link existence distributions in SBM and the pair (T_w, η_w) that denotes the family of the link weight distributions in pWSBM and $\alpha \in \mathbb{R} \in [0, 1]$ is a hyperparameter for their relative importance, E is the set of observed interactions, and W is the set of weighted edges. Thus, the fitting follows:

$$\log P(A|C, \theta) = \sum_{ij} \Big(T_e(A_{ij}) \eta(\theta_{C,C_j}) \Big) + (1 - \alpha) \sum_{ij \in W} (T_w(A_{ij}) \eta_w(\theta_{C,C_z})) \tag{3.8}$$

  As bWSBM is a hybrid version of SBM and pWSBM, it also shares its differences with our model. The final purpose of block models is to find communities by recovering the graph. Balancing the characteristics of SBM and pWSBM does not provide any benefit over LWP-WL, and our model overperforms bWSBM for the same reasons stated before.

- DCWBM (Degree Corrected Weighted Stochastic Block Model) (Aicher et al., 2014): it incorporates node degree by replacing pair (T_e, η_e) in the bWSBM with:

$$T_e(A_{ij}) = (A_{ij}, -d_i d_j) \tag{3.9}$$

  and

$$\eta_e(\theta) = (log\theta, \theta) \tag{3.10}$$

where d_i is the degree of node i.

Adding the node degree to the SBM is a method for explicitly incorporating more information into the model. However, DCWBM has less accurate predictions in predicting edge existence and edge weight (see table 3.2). Furthermore, as pointed out in (Aicher et al., 2014), degree correction methods perform better when doing community detection tasks. In LWP-WL, node degrees are not explicitly added to the model. Nevertheless, their structural roles are considered when performing the W-WL ordering algorithm. While explicitly adding node degree information to models does not improve performance in the link weight prediction task, information obtained from structural roles does (as the ablation study presented in this paper concludes).

- ModelR (Hou and Holder, 2017): is a deep learning approach for link weight prediction. It follows a simple embedding approach with a node mapping layer that maps a given node (by id) to its vector representation. Then, multiple fully hidden connected hidden layers of rectified linear units lead to an output layer which performs a linear regression:

$$f : \{n_{id\_i}, n_{id\_j}\} \rightarrow w_{ij} \tag{3.11}$$

where $\{n_{id\_i}, n_{id\_j}\}$ are the ids for nodes i and j and $w_{ij} \in \mathbb{R}$ is the predicted weight for their link. The vector representations are updated via backpropagation (SGD is chosen for optimization). It is the state-of-the-art approach for link weight prediction.

ModelR is the most comparable method with LWP-WL. Both models use deep learning to solve the task. However, LWP-WL applies a series of techniques that make it more advanced. ModelR follows a simple schema where each node is assigned to a representation, and then a set of operations are applied through multiple fully hidden layers to end up on a weight prediction. This procedure limits the generalisation power of the model due to the randomness of the node ordering process, i.e. the

nodes are assigned arbitrarily, and thus there is no consistency between different graphs.

Additionally, by giving each node a representation, the scalability of the approach is linearly limited by the number of nodes in the graph, making the method unsuitable for large graphs. LWP-WL does solve this by applying the node ordering algorithm and extracting subgraphs to make the approach's scalability independent of the number of nodes in the graph. Furthermore, the application of CNNs and the special filters presented in this work correspond to an improvement over the fully connected layers of ModelR.

- Node2vec (Grover and Leskovec, 2016): is a method that learns node embeddings based on random walk statistics. The difference between node2vec and other random walk approaches relies on its random walk producing strategy. It does bias the random walk using two parameters: p and q. These parameters control how the random walk explores the graph, concretely how exploring and leaving the neighbourhood of the starting node works, i.e. it does approximately balance the exploring procedure between Breadth-First Sampling and Depth-First Sampling methods. To learn the node embeddings, node2vec uses a decoder:

$$\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j) = \frac{\exp^{\mathbf{z}_i^\top \mathbf{z}_j}}{\sum_{v_k \in \mathcal{V}} \exp^{\mathbf{z}_i^\top \mathbf{z}_j}} \approx p_{\mathcal{G},T(v_j|v_i)}, \tag{3.12}$$

where $p_{\mathcal{G},T(v_j|v_i)}$ is the probability of visiting v_j on a T-length random walk starting from v_i. Then the approach minimizes:

$$\mathcal{L} = \sum_{(v_i,v_j)\in\mathcal{D}} -\log(\mathrm{DEC}(\mathbf{z}_i, \mathbf{z}_j)), \tag{3.13}$$

where $\mathcal{D}$ is the random walks training set. For further information about the algorithm's optimization, refer to (Grover and Leskovec, 2016). To compare this work to our model, we use the node2vec version, which uses

weights when doing the random walks. Furthermore, after obtaining the node embeddings, the link weight prediction is performed by:

$$e_{(v_i,v_j)} = z_i^\top z_j, \quad \text{with} \quad z_i, z_j \in Z = \text{node2vec}(G), \tag{3.14}$$

where G is the graph with the nodes, edges and weights: $G = (V, E, W)$.

Node2vec is an interesting, flexible and powerful method. It can be used in a wide variety of tasks and settings. Unlike our model, it is not explicitly designed for link weight prediction, so it is expected to perform worse. Random walk based approaches are dependant on the quality of the random walks, and those are highly dependant on the hyperparameter values. Thus, the necessity for finding the best possible hyperparameters for every dataset. Our model, however, is not that dependant on hyperparameters. Furthermore, LWP-WL always encodes structural roles and takes a precise edge-to-edge and node-to-edge filter when performing predictions. Node2vec, on the other hand, may not capture the structural role of every node in the graph necessarily and only takes into account the weights obtained in the random walk choosing process.

- Graph Convolutional Network (GCN) (Kipf and Welling, 2016a): they learn node embeddings through graph convolution. For this, they use a message-passing framework where node embeddings are updated by aggregating embeddings of their neighbours. The layer-wise propagation rule for GCNs for the l-th layer of a target v_i node to update is:

$$h_{v_i}^{(l+1)} = \sigma\left(\sum_j \frac{1}{c_{ij}} h_{v_j}^{(l)} W^{(l)}\right), \tag{3.15}$$

where v_j are the neighbouring nodes, h_v corresponds to the embedding of the node v, c_{ij} is a normalisation constant for the edge and $W^{(l)}$ is the matrix of weights for the l-th layer. Then, we can use the output embeddings Z to perform and train any supervised task. For link weight prediction in this work, we follow the same framework as authors in

(Kipf and Welling, 2016b) and calculate the link weight value for the edge (v_i, v_j) by:

$$e_{(v_i,v_j)} = z_i^\top z_j, \quad \text{with} \quad z_i, z_j \in Z = \text{GCN}(X, A), \tag{3.16}$$

where X is the identity matrix of the graph (as we do not input features to the model) and A is the weighted adjacency matrix for the graph. Finally, we use the Mean Squared Error to optimise the model.

GCNs and LWP-WL share two main characteristics that make them comparable: a) they both apply a structural role to the nodes by using the WL algorithm (see (Kipf and Welling, 2016a) Appendix A for the relation of GCNs with the WL algorithm), and b) they both use convolutions to aggregate the nodes information. The main difference between LWP-WL and GCNs relies on how they perform the convolution. GCNs do a neighbourhood aggregation, while LWP-WL aggregate is based on the special node-to-node and node-to-edge filters presented in this work. In this manner, LWP-WL aggregates information in a more meaningful process for the link weight prediction task.

3.3.2 Implementation details

We open source the PyTorch implementation of the LWP-WL model on GitHub[1].

We perform the same experiment for each dataset. All the link weights are normalised using the L2 norm. Each experiment consists of 25 independent trials. In each trial, we randomly split the dataset into three subsets:

- 70% into the training set.
- 10% into validation set.
- 20% into the testing set.

[1] https://github.com/unai-zulaika/LWP-WL-Pytorch

The network contains a CNN in its first layer, which applies the edge to edge and node to edge filters. Then, four fully-connected layers of sizes 32, 16, 18, 1 are applied. All of the layers, but the last one use the ReLu activation. The size of the subgraph, K, is set to 10, and we perform gradient descent by the Adam algorithm with a learning rate of 0.001. We use the best possible hyperparameters provided by the authors for all compared models. We performed a bayesian hyperparameter optimisation for node2vec and GCNs since no link weight prediction experiments were reported in the original works. For node2vec with the following setting: learning rate in range $(0.001, 0.1)$, epochs in range $(100, 1000)$, random walk length in range $(30, 100)$, number of random walks in range $(8, 16)$, embedding size in range $(8, 20)$, p in range $(0.2, 1.0)$ and q in range $(1.0, 5.0)$. For GCN, we searched: learning rate and epochs in the same ranges as for node2vec, the hidden layer size in the range $(16, 64)$ and the embedding size in the range $(4, 32)$. The final values for node2vec were: epochs = 692, random walk length = 34, learning rate = 0.08, number of random walks = 11, p = 1.0, q = 1.24 and embedding size = 1. For GCN: epochs = 641, learning rate = 0.028 hidden layer size = 27 and embedding size = 26.

For our model, the parameters were selected based on a random hyperparameter search, with the first layer comprehending sizes from 64 to 18 and the consequent layers being half the size, the size of K was selected from $5, 8, 10, 15, 20, 30$ and the learning rate selected from $0.1, 0.01, 0.001$.

The size of the fully-connected layers is proportional to the complexity of the task, and the presented setting is standard in prediction tasks. The learning rate is also a common value for the Adam algorithm. Finally, the size of the subgraph and filters, K, was selected because its performance was the best overall. K is a hyperparameter that needs to be balanced between having enough information to properly perform the prediction and not having a big size where noisy and irrelevant nodes are added to the subgraph. We empirically obtained $K = 10$ to be the most suitable value for finding the most important nodes for the prediction and not adding too many nodes that worsen the performance of LWP-WL.

We use mean squared error (MSE) as the prediction accuracy metric. For each experiment, we report the mean and standard deviation of the error from all the trials.

The experiments ran on a Linux server with an i5-4590 CPU @ 3.30GHz and TITAN X GPU with 12GB of memory. All of the experiments used Adam optimiser with a batch size of 100.

3.3.3 Comparison

In this section, we compare the experimental results of LWP-WL with several techniques, including the state-of-the-art method, ModelR. We use the setup described in the previous subsection, the same used by the authors from ModelR and the modern methods node2vec and GCNs, which were never used for the link weight prediction task. The comparison is shown in Table 3.2.

Concerning Block Model-based approaches, we argue that our approach and ModelR can predict fine-grained per node weights while Block Models can only work at the Block level. This characteristic, and the non-linear learning ability, improves both deep learning-based approaches' performance and leave Block Models behind. Furthermore, we advocate that the Weighted-WL algorithm explicitly encodes Blocks into each node's label; however, we leave this study for a next work.

As results from the table 3.2 show, our model outperforms the state-of-the-art method (ModelR). Unlike ModelR, our approach does encode structural roles due to the use of a node ordering algorithm (W-WL) based on Weisfeiler-Lehman. This feature allows our approach to better generalise over different graphs and subgraphs. Additionally, we use the adjacency matrices of extracted subgraphs as input to a Convolutional Neural Network (CNN). The use of this adjacency matrix improves our model's performance because CNNs can find patterns in links and weights between nodes, reinforced by the edge to edge and edge to nodes filters we developed for this task.

Node2vec presents good results on the task compared to block models and ModelR. However, the performance is lower when compared to GCNs and LWP-WL. Node2vec presents different performance between datasets (see

	Airport	**Collaboration**	**Congress**	**Forum**
pWSBM	$4.86 \times 10^{-2} \pm 6 \times 10^{-4}$	$4.07 \times 10^{-2} \pm 1 \times 10^{-4}$	$5.71 \times 10^{-2} \pm 4 \times 10^{-4}$	$7.26 \times 10^{-2} \pm 3 \times 10^{-4}$
bWSBM	$5.43 \times 10^{-2} \pm 5 \times 10^{-4}$	$4.62 \times 10^{-2} \pm 1 \times 10^{-4}$	$5.94 \times 10^{-2} \pm 4 \times 10^{-4}$	$8.45 \times 10^{-2} \pm 3 \times 10^{-4}$
SBM	$6.32 \times 10^{-2} \pm 8 \times 10^{-4}$	$4.97 \times 10^{-2} \pm 3 \times 10^{-4}$	$6.34 \times 10^{-2} \pm 6 \times 10^{-4}$	$8.51 \times 10^{-2} \pm 4 \times 10^{-4}$
DCWBM	$7.46 \times 10^{-2} \pm 9 \times 10^{-4}$	$5.00 \times 10^{-2} \pm 2 \times 10^{-4}$	$6.53 \times 10^{-2} \pm 4 \times 10^{-4}$	$8.82 \times 10^{-2} \pm 4 \times 10^{-4}$
ModelR	$1.3 \times 10^{-2} \pm 1 \times 10^{-3}$	$3.0 \times 10^{-2} \pm 1 \times 10^{-3}$	$3.6 \times 10^{-2} \pm 3 \times 10^{-3}$	$3.7 \times 10^{-2} \pm 1 \times 10^{-3}$
Node2vec	$3.73 \times 10^{-1} \pm 1.89 \times 10^{-1}$	$1 \times 10^{-3} \pm 1 \times 10^{-3}$	$6 \times 10^{-3} \pm 7 \times 10^{-3}$	$5.3 \times 10^{-2} \pm 1.1 \times 10^{-2}$
GCN	$2.75 \times 10^{-4} \pm 3.7 \times 10^{-5}$	$1.20 \times 10^{-4} \pm 5.8 \times 10^{-5}$	$1.89 \times 10^{-4} \pm 2.7 \times 10^{-5}$	$3.6 \times 10^{-5} \pm 5 \times 10^{-6}$
LWP-WL	$1.88 \times 10^{-4} \pm 4.6 \times 10^{-5}$	$1.39 \times 10^{-4} \pm 1.36 \times 10^{-4}$	$1.95 \times 10^{-4} \pm 3.4 \times 10^{-5}$	$7.2 \times 10^{-5} \pm 1.56 \times 10^{-4}$

Table 3.2: Mean Squared Errors, standard deviation and improvement percentage for the compared approaches on all of the datasets, results for pWSBM, bWSBM, SBM, DCWBM and ModelR belong to (Hou and Holder, 2017).

table 3.2). We hypothesise that the hyperparameter sensitivity is high for the model, and thus, each of the datasets requires further hyperparameter optimisation (we performed the hyperparameter search for the mean result in every dataset). Nevertheless, we choose to share the same hyperparameter setting for the comparison to be fair, in the same manner as we did for every model. Overall, node2vec is a model with good performance and scalability, but it lacks the characteristics that make GCNs and LWP-WL stand out, encoding structural roles and aggregating graph information.

GCNs prove themselves as a powerful method for the task. The results, as expected, are equivalent to LWP-WL, even outperforming on some of the datasets both in small-medium (see table 3.2) and big sized graphs (see table 3.3). The similarities that both methods share make them the best performing algorithms. They can both encode structural roles via the WL algorithm and apply convolution operations to aggregate information within the graph, being how they apply those convolutions the architectural difference between the methods. However, another aspect that differences LWP-WL from GCNs is scalability. Scalability is further analysed in the following subsection.

3.3.4 **Scalability and complexity**

We perform experiments with two big datasets (Geom and Astro) to address the scalability of our approach. We compare with ModelR since it is the state-of-the-art approach and GCNs due to their excellent performance and similarities with LWP-WL.

As shown in Table 3.3, LWP-WL also outperforms ModelR for big datasets. Moreover, ModelR follows an embedding approach where each node is given a representation that is updated at training time. Authors of ModelR state that they set up the size of the embedding layer d following the rule $d = \log_2(N)$ where N is the number of nodes in the graph. Thus, the number of parameters in the embedding layer w_e scales (we ignore the following hidden layers since their size is not directly associated with the input) :

$$w_e = N \times d = N \log_2(N) \quad . \tag{3.17}$$

Thus, ModelR presents a highly scalable approach but incomparable performance-wise with LWP-WL.

On the other hand, GCNs scale in $\mathcal{O}(Lmd + Lmd^2)$ (Wu et al., 2020) being L the number of layers, m the number of edges and d the embedding size. This imposes a limitation on m the number of edges, which increases linearly and makes GCNs a less scalable model for big or growing graphs. Although some works proposed alternatives to overcome this limitation (Hamilton et al., 2017a), it remains incomparable to LWP-WL as it only depends on the size of the subgraph K. Thus, although GCNs prove to be a reliable and well-performing model for the link weight prediction task, LWP-WL presents much better scalability making it suitable for larger graphs.

Unlike ModelR and GCNs, LWP-WL provides a user-defined parameter that constrains the number of parameters of the method, the size of the subgraph (K). This parameter defines the size of the subgraph to be the input to the neural network, keeping uniform the number of parameters needed by the LWP-WL model and doing it invariant to the number of nodes in the graph. The sum of the edge-to-edge and edge-to-node layers for the LWP-WL network will have parameters

$$w = K^2 C \tag{3.18}$$

where $C \in \mathbb{N}$ is the number of channels of the convolutional layers, we set it up to 10 by default.

However, the property of not scaling to the number of nodes in the graph comes at the cost of the computational and time complexity of the labelling algorithm. The W-WL algorithm takes K iterations to converge. The computations needed to complete the algorithm are the following, *i)* the hash function for every node, performed in $\mathcal{O}(K)$ time, *ii)* the evaluation of all hashes, it is done in $\mathcal{O}(K^2)$ time, and *iii)* the sorting of the nodes which is completed in $\mathcal{O}(Klog(k))$ time. Therefore, the algorithm has $\mathcal{O}(K^2)$ time complexity in each iteration, and it needs $\mathcal{O}(K^3)$ time to finish in a graph of

	Astro	**Geom**
ModelR	$4.7 \times 10^{-2} \pm 9 \times 10^{-3}$	$2.17 \times 10^{-2} \pm 2 \times 10^{-3}$
GCN	$5.5 \times 10^{-5} \pm 1.3 \times 10^{-5}$	$8 \times 10^{-6} \pm 1 \times 10^{-6}$
LWP-WL	$9 \times 10^{-6} \pm 2 \times 10^{-6}$	$5.4 \times 10^{-5} \pm 1.9 \times 10^{-5}$

Table 3.3: Comparison between different methods and our approach, Mean Squared Errors, standard deviation, parameter amount and improvement percentage in the big-sized Astro and Geom datasets.

K nodes. It is important to notice that the labelling process is parallelisable.

We argue that in the tradeoff between having to complete the labelling algorithm in $\mathcal{O}(K^3)$ time for a subgraph of size K that usually varies between 5 and 25 (10 by default), and having a model that scales $N \log_2(N)$ for the full-size of the graph, LWP-WL is a more scalable approach that can handle big-sized graphs with even millions of nodes.

3.3.5 Ablation study

We also performed an ablation study with the final goal of understanding the impact level of different features of our model. For this matter, we used a single dataset and measured the performance while gradually adding features. The purpose of this study is to gather reliable knowledge about our approach and the impact that different features have on it. There are two main features that we want to analyse.

Node labelling algorithm (W-WL). The W-WL node labelling algorithm for weighted graphs is one of the main contributions of this work. It provides consistency to the model by labelling roles that have a similar structural role with similar labels. We compare this algorithm with random labelling for nodes.

Version	Node labeling	CNN
1	random	no
2	W-WL	no
3	random	yes
4	W-WL	yes

Table 3.4: Summary of the versions used for the ablation study.

Convolutional Neural Network layer. The CNN is the first layer for the model that applies a unique edge to node and edge to edge filters to the input adjacency matrix. We want to measure this layer's impact in predicting and finding patterns between nodes, edges, and their weights.

We developed different versions of our approach by combining these features; a summary of those is shown in Table 3.4.

1. The baseline version, we do not add any feature to the approach. After extracting the enclosing subgraph, nodes are randomly ordered, and the flattened adjacency matrix is fed to a model without the CNN layer.

2. We add the W-WL node labelling algorithm to the baseline version.

3. In this version, CNN is used. However, nodes are randomly ordered.

4. Final version with all the features, the W-WL node labelling algorithm, the weighted adjacency matrix and the CNN layer are both used.

We used the Airport dataset for this study with the same setup used for the comparison subsection. The results obtained are shown in Table 3.5.

It is noticeable how features impact the model's performance, and the combination of these features allowed us to create a state-of-the-art approach for link weight prediction.

Results clearly demonstrate the effectiveness of the W-WL labelling algorithm, as it does significantly improve the versions that do not have it.

Ver.	MSE	Impr.	PCC	Impr.
1	$1.89 \times 10^{-4} \pm 2.6 \times 10^{-5}$		0.456±0.033	
2	$1.52 \times 10^{-4} \pm 3.2 \times 10^{-5}$	24.34% (v1)	0.595±0.129	23.36% (v1)
3	$2.05 \times 10^{-4} \pm 3.2 \times 10^{-5}$	-25.85% (v2)	0.412±0.088	-44.41% (v2)
4	$1.69 \times 10^{-4} \pm 4.2 \times 10^{-5}$	21.3% (v3)	0.601±0.127	31.44% (v3)

Table 3.5: Results for each version of the algorithm applied to the Airport dataset. We report the Pearson Correlation Coefficient and the Mean Squared errors with the Standard Deviation for 25 trials on each version. We also provide the improvement percentage of every version with regard to the previous one for each metric.

Furthermore, the recent literature has demonstrated promising results on approaches using enclosing subgraph extraction. Besides our work, there have been two recent approaches for link prediction, WLNM (Zhang and Chen, 2017), and SEAL (Zhang and Chen, 2018), that achieved state-of-the-art results for the task. Therefore, a proper representation for the target link has been proven enough to predict links and their weights, avoiding whole processing graphs. However, it is compulsory to provide a node labelling technique to use an enclosing subgraph extraction approach. Further models can be generalised over different subgraphs by consistently labelling nodes to learn predictions. Additionally, the node labelling technique must preserve the topological directionality towards the target link to get the best performance, providing a mechanism for the model to focus on specific nodes.

Regarding the CNN layer, it does not improve the baseline when nodes are randomly ordered (version 1 to 3). However, it improves the efficiency when we apply the W-WL algorithm (version 3 to 4). This fact is due to the randomness of node order in the subgraph. When the labelling technique is applied, the nodes' structure and roles arise, providing proper criteria for the CNN to learn.

3.3.6 Results and conclusions

Finally, we present the conclusions and answers obtained to the questions formulated at the beginning of the section. These experiments and comparisons match the objectives (**O1**) and (**O5**) of the thesis. Precisely, we studied the current state of the art in machine learning applied to graphs and link weight prediction and properly evaluated our method according to the task.

How well does LWP-WL perform in the link weight prediction task compared to the state-of-the-art models? Do the best performing models share any common characteristics?

The experiments performed in this section present the LWP-WL algorithm as the new state-of-the-art model for link weight prediction. The approach provides two main improvements over the compared models: **a)** the extraction and ordering of nodes in subgraphs which provides a structural organisation pattern for the model, and **b)** the performance and scalability obtained from the use of specific CNNs for the link weight prediction task.

Remarkably, the best performing models, i.e. LWP-WL, GCN, Node2vec and ModelR, are based on representation learning techniques. They all learn representations for the nodes in the graph to perform the link weight prediction task. Furthermore, LWP-WL, GCN and Node2vec use a convolution (Node2vec does it via random walk) to obtain such representations. We conclude that the models most capable of generating the best representations for the nodes in the graph are the most successful. To do so, aggregating information from the most significant neighbours and nodes is critical. However, the main issue is that models need consistent node ordering criteria when learning such representations. Otherwise, whether a node plays a vital role in the graph or lies as a simple disconnected node is irrelevant for the model. LWP-WL and GCNs are the best performing models because they encode structural roles via the WL algorithm, and thus, they have a criterion to generalise over training samples.

How scalable is LWP-WL compared to the best models? Does it perform well in big graphs? How can we achieve graph size-wise independent scalability?

We conclude that LWP-WL is a scalable approach that can handle big-sized graphs with even millions of nodes. Furthermore, the presented experiments show LWP-WL as an even contender with GCNs for big-sized graphs.

The main problem when working with graph-based methods is to make them independent of the graph size. The subgraph extraction method makes LWP-WL invariant to the number of nodes in the graph to solve such an issue, i.e. to predict a link weight, we only need a minor part of the graph. However, not scaling to the number of nodes in the graph comes at the cost of the computational and time complexity of the W-WL labelling algorithm. Nevertheless, the time needed to compute the W-WL algorithm for a given subgraph will always stay the same, while models that process the whole graph will linearly scale to the number of nodes in such graph.

What is the impact of the structural roles for nodes and, thus, the Weighted-WL graph labelling algorithm? And the special edge-to-edge and edge-to-node filters?

As we presented in the ablation study, the W-WL algorithm is the essential aspect of LWP-WL. The algorithm improves the model's performance, even making the LWP-WL without the CNN version significantly worse. This conclusion aligns with the strong performance of GCNs for the task. GCNs, as LWP-WL do also encode structural roles in the model, improving the performance and making GCNs a strong baseline.

Nevertheless, the CNN layer and the special edge-to-edge and edge-to-node filters prove themself an important addition to the model by improving its performance once the W-WL algorithm has been applied. Due to the structure that the algorithm provides, CNNs can properly apply convolutions over graph data and obtain superior performance compared to fully hidden layers.

3.4 **Interpretability**

In this section, we analyze the interpretability provided by LWP-WL. Although conceptually, LWP-WL was not meant with interpretability in mind, the final algorithm provides a fair grade of interpretable processes.

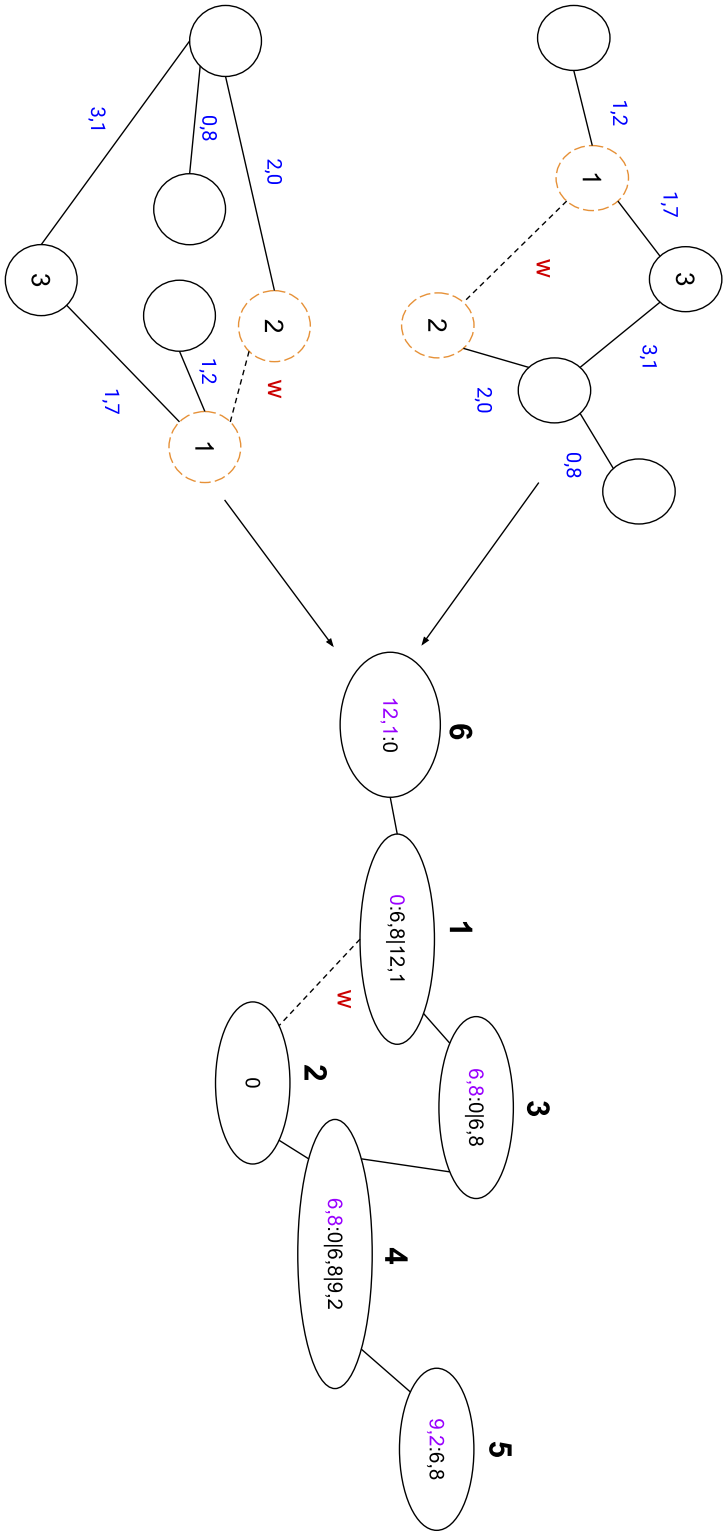


Figure 3.4: The same graph presented in two different ways. After applying Weighted-WL, we obtain the same node ordering and signature strings (only signatures after the first step are given).

As already stated, two main aspects of the approach provide a high grade of interpretability: the graph colouring algorithm, i.e. the Weisfeiler-Lehman variant, and the graph subtraction process.

One of the main characteristics of the subgraph extraction process is that it **can extract a similar context representation for all links**. Thus, by observing the obtained subgraph, every human can associate and generalize how the subgraph extraction process happened.

Nevertheless, the most crucial aspect is the graph colouring algorithm. As already presented, the weighted variant of Weisfeiler-Lehman proposed in this work fulfils two requirements:

- The graph labelling algorithm provides **similar labels to nodes with similar topological features** within the enclosing subgraph.
- It preserves the **topological directionality towards the target link**, i.e. the order of the nodes has to be conditioned by the target nodes, and the distance to them must be reflected in the ordering.

Those requirements allow de facto to develop an interpretation of the adjacency matrix, which is input to the CNN. In this manner, even though the final process performed by the CNN to obtain the final weights is not directly interpretable (a possible future work would be to apply interpretability methods to the CNN, e.g. saliency methods or correlating abstract neurons), we can interpret the data it uses. For instance, let us suppose we have two identical graphs. However, those graphs are ordered differently, and we humans cannot identify that both graphs are the same (in a graph with many nodes, it is easy to happen). After using our node ordering method, the Weighted-WL algorithm, the obtained node ordering and signature for each node are the same; thus, a human can identify that it was the same graph. A Figure depicting this phenomenon is presented in 3.4.

The concept of obtaining **similar labels to nodes with similar topological features** can be further used for interpretability. Nodes with similar roles have identical signatures and numbers in the ordered set. Figure 3.5 provides an overview of this idea. We have two different subgraphs that were

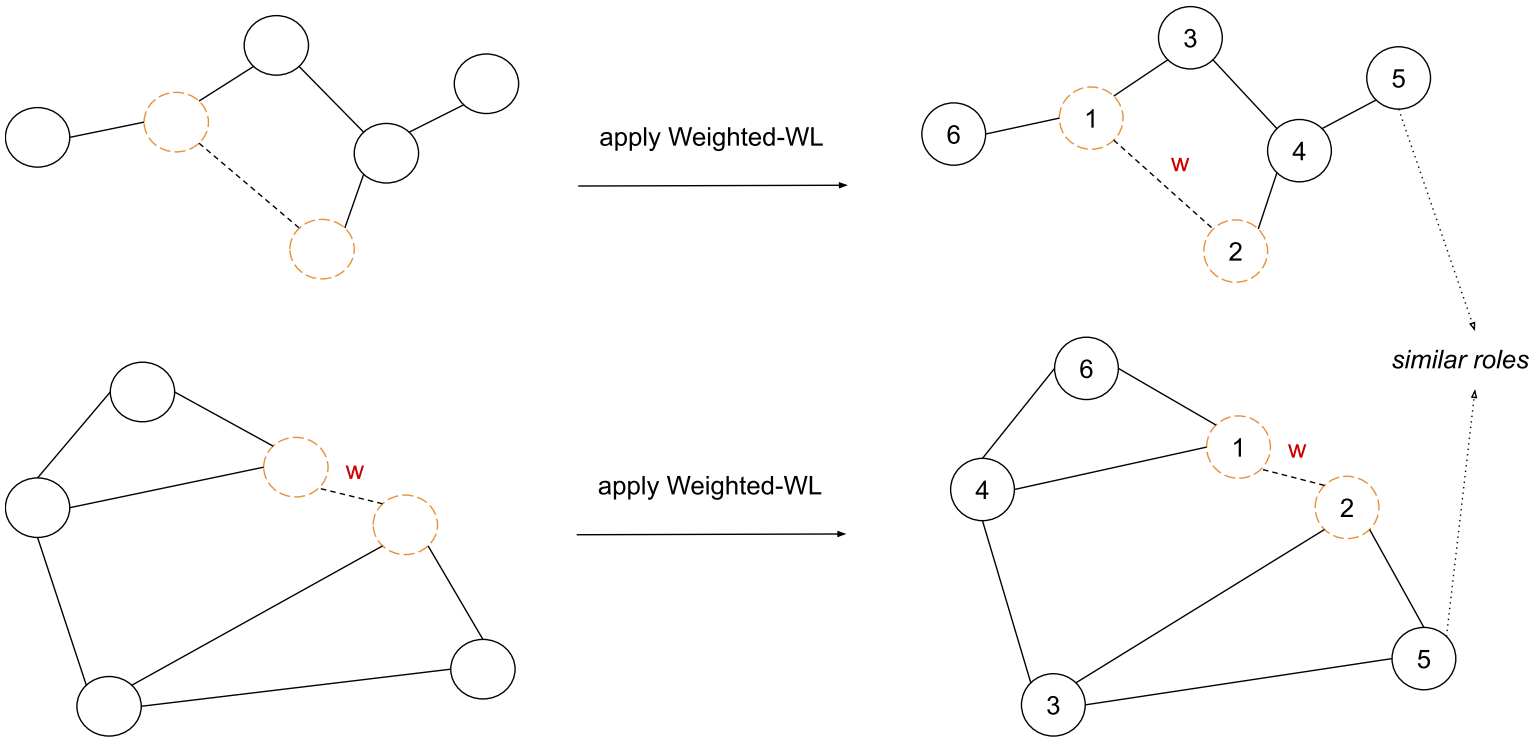


Figure 3.5: Two different subgraphs (left) are extracted to perform the node ordering algorithm. After applying W-WL (right), we can compare the node order to find similar roles between both subgraphs. Furthermore, we could check each node's signature to compare the level of similarity (the signature was not presented in the figure to facilitate the presentation).

extracted from the graph. Before applying the W-WL algorithm (left), humans can not associate nodes or roles between the two graphs. After applying the algorithm, we can associate the number each node was given with its homologous in the other graph. For example, the nodes with the number 5 fulfil a similar role within their corresponding subgraphs. Indeed, we could check the final signature for both nodes to compare how similar they are. If the signatures matched almost completely, both nodes would fulfil the same role within their subgraphs. In contrast, if their signature is different, although they have the same number in the node ordering, they do not fulfil the same role within their subgraph.

In summary, the combination of both the node ordering algorithm and the subgraph extraction process lets to obtain a feature representation of every node in the graph in a scalable manner. Then, the comparison of the representations obtained can be used to acquire an idea of the similarity of the roles between subgraphs, increasing the interpretability of the whole process.

"Al final y al cabo la gloria no es tanto, la gloria debe ser morirse en una islita griega mirando al mar."

- Gata Cattana

CHAPTER

4

Sparse Knowledge Graph Embeddings for intrinsic interpretability

In this chapter we present a novel technique for learning sparse Knowledge Graph Representations. The approach uses the generalized Regularized Dual Averaging online optimization algorithm and applies the l_1 regularization into the RESCAL model.

The chapter is divided into three sections: Section 4.1 introduces a set of cognitive arguments for applying sparsity to the representations. We also contextualise the presented interpretability tool within the taxonomy followed in this work. Then, Section 4.2 explains the proposed approach for developing sparse representations. Finally, Section 4.3 presents the experiments and results of the interpretability tool.

4.1 Sparsity and interpretability from cognitive arguments

In the taxonomy (Molnar, 2020) introduced in Section 1.1 different categorisations were provided. In the current chapter, we develop an interpretability tool corresponding to the **model internals**. This category belongs to the methods that make a model intrinsically interpretable. There are different ways to make a model intrinsically interpretable, e.g. forcing linear or sparse weights. The key aspect of the category is that it affects the internal process or values of the model.

The application of the internal model method proposed in this chapter follows a specific approach. We conceive an approach to develop sparse embeddings (representations) for Knowledge Graph Embedding models. Those models produce embeddings for the concepts and relationships within a KG. Then, the embeddings will be used to perform any task related to the KG, e.g. link prediction. Embeddings are, however, difficult to interpret. They are real-valued vectors whose individual dimensions can not be understood, i.e. their latent dimensions have low semantic meaning.

First, we look at two important cognitive arguments to overcome the lack of interpretability of the embeddings. These arguments will help us identify the desirable characteristics for an embedding to be more interpretable:

1. Several cognitive arguments based on the economy of storage maintain that a small set of features is not enough to describe every semantic concept and domain in our vocabularies (Schunn, 2020; Murphy, 2004; Murphy et al., 2012). From the cognitive point of view, some words require more characteristics to be described, and others require less. Besides, certain properties belong to specific semantic domains and sharing properties is not natural for humans.

2. We do not describe concepts based on what they are not since it would be uneconomical; thus, we do not store that a desk is not a vegetable or a glass can not fly. Following these arguments, as authors in (Murphy et al., 2012) presented, the feature set of a representation should only

store positive facts, a wide range of feature types, and only a small quantity of these features should be used to describe a concept.

The corresponding characteristics of these arguments for an embedding are sparsity and non-negativity. Sparsity corresponds to the idea of describing a concept with specific semantic domains. Non-negativity matches the argument that we describe concepts based on what they are.

This chapter presents a solution to get the sparsity property in training time for Knowledge Graph Embeddings. Nevertheless, while we do not strictly induce non-negativity in our method, it highly (almost completely) enforces embeddings to have high non-negativity values, as we demonstrate in Section 4.3.4.

4.2 Sparse Tensor Factorization

This section presents our approach for applying the l_1 regularization to the RESCAL model via the gRDA optimization algorithm. First, we introduce the background and the RESCAL model itself, then describe the details of the approach. The same technique can be applied to other KGE models such as TuckER or DistMult.

4.2.1 Background

Let $\mathcal{E}$ denote the set of all entities and $\mathcal{R}$ the set of all relations present in a knowledge graph. We denote the collection of triples in a KG as $\mathcal{D}$ and each triple is represented as $(e_s, r, e_o) \in \mathcal{D}$, with $e_s, e_o \in \mathcal{E}$ denoting subject and object entities respectively and $r \in \mathcal{R}$ the relation between them.

In the link prediction task the objective is to learn a scoring function ϕ which scores, $s = \phi(e_s, r, e_o) \in \mathbb{R}$, whether a triple is true or false. To complete the task, we observe a subset of all true triples, aiming to score all the missing ones correctly. In this work, we only consider scoring functions given by a tensor factorization technique, e.g. RESCAL.

4.2.2 RESCAL model

The RESCAL model (Nickel et al., 2011) is a relaxed version of the DEDICOM (Harshman, 1978) matrix factorization method, which decomposes a matrix into two matrices that provide an asymmetric relation between entities.

In the case of KGs the tensor is three-mode, therefore the original tensor $\mathcal{X} \in \mathbb{R}^{n_e \times n_e \times n_r}$, is decomposed by RESCAL into a entity matrix $\mathbf{A} \in \mathbb{R}^{n_e \times d}$ and k relation matrices $\mathbf{R}_k \in \mathbb{R}^{d \times d}$ where n_e represents the number of entities in the KG and d is a hyperpameter for the embedding dimensionality. Thus, the k-th slice of the tensor is factorized as

$$\mathcal{X}_k \approx A\mathbf{R}_k\mathbf{A}^T, \text{for } k = 1, ..., n_r \tag{4.1}$$

where n_r is the number of slices (relations) in the tensor.

RESCAL is a latent feature model which scores triples by the interaction of the latent features of the subject and object entities. The scoring is given by

$$\phi(e_s, r, e_o) = \mathbf{e}_s^T\mathbf{R}_k\mathbf{e}_o, \tag{4.2}$$

where $\mathbf{e}_s, \mathbf{e}_o \in \mathbb{R}^d$ are respectively the subject and object embeddings from the entity embedding matrix $\mathbf{E}$ ($\mathbf{A}$ in equation 4.1), and $\mathbf{R}_k$ is the asymmetric relation matrix corresponding to the k-th relation in the KG.

4.2.3 Sparse RESCAL

We aim to introduce sparsity into the embeddings via l_1 regularization. As we know, the l_1 regularization applies a penalty in the learned weights, which makes them push towards 0, leading to finding sparse solutions for the embeddings.

The RESCAL model was originally (Nickel et al., 2011) optimized using the Alternating Least Squares (ALS) method. Nevertheless, different works have pointed out that newer training techniques improve model performance (Kadlec et al., 2017; Lacroix et al., 2020). Furthermore, in a recent work (Ruffinelli et al., 2019) that provides an extensive analysis on KG models and their training processes, authors present the Adam (Kingma and Ba, 2014)

method to be the best for training RESCAL. However, as Adam is a stochastic gradient descent (SGD) extension, it can not produce sparse solutions directly applying l_1 regularization to the loss function. The minor updates which SGD makes in the vectors' values difficult the output of many zero entries (Duchi et al., 2011), i.e. it is quite challenging to get two float numbers to add up and equal zero. To overcome this issue, we propose using generalized Regularized Dual Averaging (gRDA) for optimizing the RESCAL model with sparse constraints. gRDA is itself a generalization of the RDA algorithm for sparse neural networks (Xiao, 2010), which can be very useful for sparse online learning with l_1 regularization as it can explicitly exploit the regularization structure. In each iteration of the RDA algorithm, the weights of the model are updated, taking into account not only the loss function but also the whole regularization term introduced to achieve the sparsity.

To apply l_1 regularization to our model, we first update the RESCAL model's loss function:

$$\mathcal{L}_{sRESCAL} = \mathcal{L}_{RESCAL} + \lambda \sum_{w \in W} ||w||_1 \tag{4.3}$$

where λ is a hyperparameter that controls the importance of the regularization term.

We follow the same training procedure as (Balažević et al., 2019). We apply the data augmentation method used by (Dettmers et al., 2018) adding reciprocal relations for every triple in the dataset. We also use *1-N scoring*, i.e. we score all entity-relation pairs $\{(e_s, r\})$ and the corresponding inverse $\{(e_o, r^{-1}\})$ with every entity $e \in \mathcal{E}$. We use the Binary Cross Entropy (BCE) loss function:

$$\mathcal{L}_{RESCAL} = -\frac{1}{n_e} \sum_{i=1}^{n_e} \{(\mathbf{y}^i \log\{(\mathbf{p}^i\}) + \{(1 - \mathbf{y}^i\}) \log\{(1 - \mathbf{p}^i\})\}), \tag{4.4}$$

where $\mathbf{p} \in \mathbb{R}^{n_e}$ is the vector of predicted probabilities and $\mathbf{y} \in \mathbb{R}^{n_e}$ is the binary label vector.

4.2.4 Optimization

In gRDA, at each iteration, the learning weights are adjusted by solving a simple optimization problem that involves the running average of all past subgradients of the loss function. Its update rule is the following:

$$\mathbf{w}_{n+1} = \underset{\mathbf{w}\in\mathbb{R}^d}{\arg\min} \left\{ \mathbf{w}^T\{(-\mathbf{w}_0 + \gamma \sum_{i=0}^{n} \Delta f\{(\mathbf{w}_i; Z_{i+1}\})\}) \right. \\ \left. + g\{(n,\gamma\})\mathcal{P}(\mathbf{w}) + F(\mathbf{w}) \right\} \tag{4.5}$$

where γ is the step size, $\mathcal{P}(\mathbf{w})$ is the penalty function, $F(\mathbf{w})$ is a deterministic and convex regularizer which stabilizes the optimization process in the same manner as proposed in (Shalev-Shwartz et al., 2012) and $g\{(n,\gamma\})$ is a deterministic non-negative function of n, γ. Notice that RDA is a special case of gRDA where $g\{(n,\gamma\}) = n\gamma$ and $\mathbf{w}_0 = 0$.

Since we want to apply the l_1 regularization to the RESCAL model, we follow the same criteria given by (Chao and Cheng, 2019) for gRDA-l_1. We set the strongly convex function $F(\mathbf{w}) = \frac{1}{2}\{||\mathbf{w}\}||_2^2$ and the penalty function to be $\mathcal{P}(\mathbf{w}) = \{||w\}||_1$. We also follow the function

$$g(n,\gamma) = c\gamma^{\frac{1}{2}}(n\gamma - t_0)_+^{\mu}, \tag{4.6}$$

which is conjectured by authors to be a universal formula for applying gRDA-l_1 in difficult tasks. In equation 4.6, $c = \gamma$ and μ are hyperparameters and $t_0 \geq 0$ is the time mean dynamics. Furthermore, μ is the trade-off hyperparameter between accuracy and sparsity.

Following the stochastic mirror descent representation of gRDA (see (Chao and Cheng, 2019) Section 2) and given $F(\mathbf{w}) = \frac{1}{2}\{||\mathbf{w}\}||_2^2$, we can use the closed-form proximal operator of $g(n,\lambda)\mathcal{P}(\mathbf{w})$ (Parikh et al., 2014), for $j = 1, 2, ..., d$,

$$\Delta\Psi^*_{n,\gamma,j}(\mathbf{v}) = \text{sgn}(v_j) \cdot \{(\{|v_j\}| - g\{(n,\gamma\})\})_+, \tag{4.7}$$

which will serve as our penalty function $\mathcal{P}$ for gRDA-l_1. Thanks to the closed-form proximal operator, the computational cost per iteration in the

Algorithm 3: gRDA-l_1 algorithm for sparse RESCAL

Input: $\mathcal{D}$
Initialize: $\mathbf{e}, \forall \mathbf{e} \in \mathbf{A}, \mathbf{w}, \forall \mathbf{w} \in \mathbf{R}$
for n ***in*** $\mathcal{D}$ **do**
  $t \rightarrow$ update time of triple n
  accumulate gradients
  $v := c\gamma^{\frac{1}{2}}(n\gamma - t)_+^{\mu}$
  for $j = 1$ ***to*** d **do**
    $w_{j+1} := sgn(v_j) \cdot (\{|v\}| - \gamma \Delta f\{(\mathbf{w}_j; Z_{j+1}\}))$

stochastic mirror descent gRDA is as cheap as SGD.

We present the algorithm for optimizing the sparse RESCAL model (sRESCAL) in algorithm 3. We first initialize the weights of the entity and relation matrices (for each slice) $\mathbf{E}, \mathbf{R}_k$. Then, for each triplet n in the training set $\mathcal{D}$, we update the accumulator of gradients v by applying equation 4.6. Finally, we set the weights by following the update rule in equation 4.7.

4.3 Experiments and results

In this section, we analyze the performance in the link prediction task and the interpretability of the proposed algorithm. We compare our sparse model with several dense model baselines. First of all, we describe the datasets and the setup used for the experiments. Then, we study the link prediction task and compare our model with the state-of-the-art dense models. Finally, we analyze the interpretability of the model using the word intrusion task.

4.3.1 Datasets

The datasets used for the evaluation are the following (see Table 4.1):

- **FB15k** (Bordes et al., 2013) is a subset of the Freebase database which contains facts about the world.

Dataset	$\mid \mathcal{E} \mid$	$\mid \mathcal{R} \mid$
FB15k	14,951	1,345
FB15k-237	14,541	237
WN18	40„943	18
WN18RR	40,943	11

Table 4.1: Datasets for link prediction and their number of entities and relations.

- **FB15k-237** (Toutanova et al., 2015) is a better-suited version of FB15k where the inverse of many relations was removed to increase the difficulty.
- **WN18** (Bordes et al., 2013) is a subset of WordNet, a hierarchical database containing lexical relations between words.
- **WN18RR** (Dettmers et al., 2018) is a better-suited version of WN18 where the inverse of many relations was removed to increase the difficulty.

Both datasets were created for the link prediction task.

4.3.2 Implementation details

We open source the PyTorch implementation of the sRESCAL model on GitHub[1].

We select the hyper-parameters using random search by validation set performance. For FB15k and FB15k-237, we select the entity and relation embedding size from 128, 248, 512. Embedding size for WN18 and WN18RR is chosen from 50, 100, 200 due to the small number of relations in the dataset. Additionally, we apply batch normalization (Ioffe and Szegedy, 2015) and dropout (Srivastava et al., 2014) to ease the training procedure. We choose the dropout value in the range (0.1, 0.5) for every dataset. The learning rate

[1]https://github.com/unai-zulaika/sRESCAL.

Dataset	lr	μ	c	d_e	d_r	d#1	d#2	d#3	ls
FB15k	0.2	0.65	0.00005	248	128	0.36	0.4	0.4	0.17
FB15k-237	0.2	0.65	0.00005	248	128	0.36	0.4	0.4	0.17
WN18	0.9	0.4	0.00005	200	50	0.36	0.4	0.4	0.17
WN18RR	0.9	0.4	0.00005	200	50	0.36	0.4	0.4	0.17

Table 4.2: Best hyperparameters for sRESCAL for datasets where: LR denotes learning rate, dr decay rate, ls label smoothing, and $d\#k, k \in \{1, 2, 3\}$ dropout values applied on the subject entity embedding, relation matrix and subject entity embedding after it has been transformed by the relation matrix respectively.

is selected within the range (0.1, 0.9) also for both datasets as we observed high learning rate values necessary to overcome local minima. We set c to 0.00005 since we found it to be a suitable value for the starting sparsity in the embeddings. We choose μ in the range (0.6, 0.8) where high values correspond to higher sparsity in embeddings. We analyze the impact of μ in the following subsections. Finally, the batch size is selected from 248, 512, 1028. The best hyperparameters for each dataset are presented in table 4.2.

4.3.3 Link prediction

For the evaluation of the link prediction task, we create all possible candidate triples by adding every entity in $\mathcal{E}$ to the test entity-pair, then we use our model to score and rank each of the candidates. We apply the filtered setting where all known true triples in the Knowledge Graph $\mathcal{D}$ are removed from the candidate set. We use the standard metrics for the task: mean reciprocal rank and hits@k, k $\in$ 1, 3, 10. Mean reciprocal rank is the average of the inverse of the mean rank assigned to the true triple overall candidate triples. Hits@k measures the percentage of times a true triple is ranked within the top k candidate triples.

We present the link prediction results in Table 4.3. Although sRESCAL does not outperform the state-of-the-art models, it does remain competitive while finding more interpretable and sparse solutions (see Subsection 4.3.4).

		RESCAL	DistMult	ComplEx	TuckER	sRESCAL
FB15k	MRR	.64	**.84**	.83	.79	.38
	Hits@10	.82	**.90**	.89	.89	.55
	Hits@3	.70	**.86**	.85	.83	.42
	Hits@1	.54	**.80**	**.80**	.74	.29
FB15k-237	MRR	**.35**	.24	.24	**.35**	**.34**
	Hits@10	**.54**	.41	.42	**.54**	.52
	Hits@3	**.39**	.26	.27	**.39**	.37
	Hits@1	**.26**	.15	.15	**.26**	**.25**
WN18	MRR	.94	.94	**.95**	**.95**	.89
	Hits@10	.95	.95	.95	.95	.94
	Hits@3	.95	.94	.95	.95	.92
	Hits@1	.94	.93	.94	**.95**	.87
WN18RR	MRR	.46	.43	.44	**.47**	.43
	Hits@10	.51	.49	.51	**.52**	.49
	Hits@3	**.48**	.44	.46	**.48**	.45
	Hits@1	.43	.39	.41	**.44**	.39

Table 4.3: Results in link prediction.

Regarding the simplest versions of the datasets, in WN18, sRESCAL achieves high results on every metric and gets close to the other models. For FB15k, sRESCAL suffers the same phenomena as the standard RESCAL model. The performance on the dataset is considerably lower when compared to other models. We argue that this happens due to not finding the optimal hyperparameters for the dataset. Results from (Ruffinelli et al., 2019; Wang et al., 2018) also provide low values for RESCAL in such dataset. However, as we will see next, in the more challenging version of the dataset, FB15k-237, RESCAL and sRESCAL provide significantly better results, being two of the strongest models overall.

Regarding the most challenging datasets, sRESCAL achieves fantastic results, being competitive in every metric. Our model equates DistMult and is close to ComplEx performance-wise in the WN18RR dataset and stays close to the state-of-the-art model, TuckER, by less than 4 points in the metrics. For FB15k-237, sRESCAL outperforms both DistMult and ComplEx by a significant margin in every metric. Furthermore, the performance almost equals RESCAL and TuckER providing fantastic results on every metric.

While the performance of sRESCAL remains competitive to other models, it does also achieve high sparsity and interpretability levels. Thus, we present sRESCAL as a valid and robust model for the link prediction task. Furthermore, as stated before, the induction of sparsity in the resulting embeddings will increase their interpretability by raising semanticity in their latent dimension. The increase of interpretability is further analyzed in Subsection 4.3.4 and the outcome is observable in table 4.6. However, the introduction of sparsity to the embeddings does also have a penalization on performance. Thus we analyze the tradeoff between sparsity, thus interpretability, and performance.

Sparsity and performance trade-off

The trade-off between the sparsity level and performance in the embeddings is a crucial aspect to study. sRESCAL is a flexible model that can be tuned to increase or decrease its sparsity levels and, thus, its performance.

Since sRESCAL uses the gRDA-l_1 optimization algorithm, sparsity is defined by hyperparameters c (initial sparsity level) and μ, which defines the penalization applied to the learning weights. By tuning those hyperparameters, we can achieve the sparsity level we want at the cost of lowering the final performance of the model. To demonstrate the variation on those hyperparameters, we present different metrics: (a) mean rank value, (b) DistRatio (an interpretability metric based on the word intrusion task) value and (c) sparsity percentage for dataset FB15k-237 in Figure 4.1. We provide results on different values of μ while fixing every other hyperparameter (we also fix c since we found the value 0.0005 to be the most optimal for sRESCAL in every case).

Results show a clear trade-off between μ, thus the sparsity level of the solution, and the final results on both performance and interpretability. Figure 4.1 (a) presents a difference between μ values where high values, 0.85 and 0.65, achieve worst results on mean rank when compared to the lower, 0.25, 0.45 values. However, we find that $\mu = 0.65$ remains competitive to lower values as the difference in the mean rank is quite small. Furthermore, when revising the interpretability metric DistRatio (further explained in next subsection) in (b), the difference between the most interpretable models with high μ values is

evidenced. As expected, solutions with a high μ value achieve good results on DistRatio while those with low μ perform worse. This fact is directly related to the sparsity level of the solution, which is shown in (c), where high μ valued solutions have higher sparsity levels. As we stated, gRDA allows sRESCAL to be a model that obtains high sparsity values and improves its interpretability [1]. Nevertheless, this comes with a trade-off on the standard link prediction performance metrics such as mean rank, lowering the model's performance on those but raising the interpretability of the outcome embeddings.

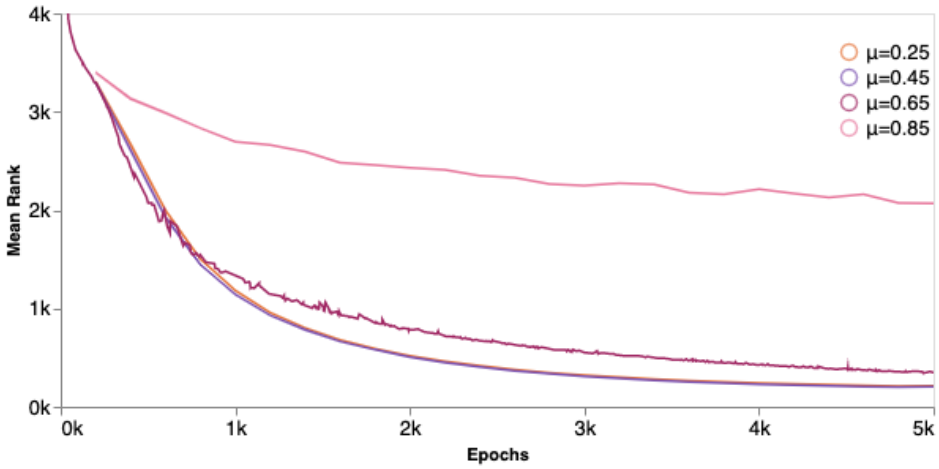
We analysed and compared our model to the others performance-wise in this subsection. The following subsection presents an analysis of the interpretability level achieved by our model. Thus, in this chapter, we fulfilled the following thesis objectives:

- (**O1**) as we studied link prediction models and how we could make them more interpretable by inducing sparsity.
- (**O3**) as we present a novel technique that makes the model's internals more interpretable.
- (**O5**), we presented an analysis in the link prediction task and provided an evaluation of the impact on the interpretability of the methods.
- (**O6**) we analysed the sparsity and performance trade-off of our model.

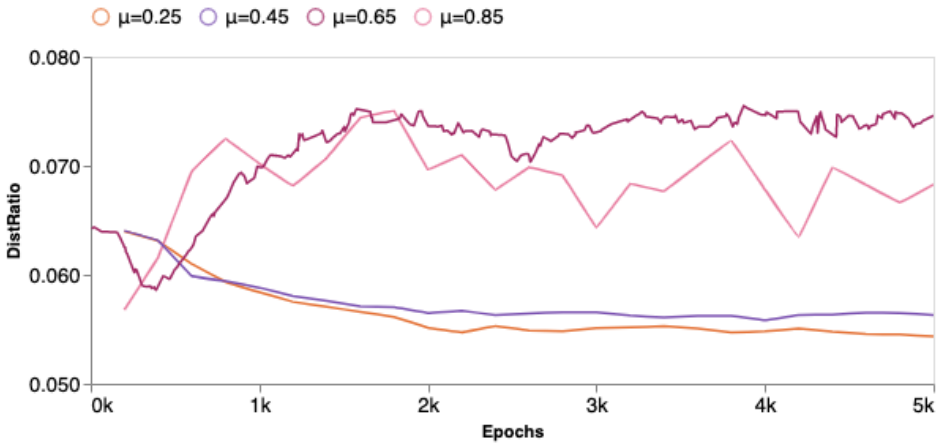
4.3.4 Interpretability

In this section, we analyze the interpretability of our sparse model compared to the dense state-of-the-art models. In the same manner as (Murphy et al., 2012; Faruqui et al., 2015), we perform experiments on the word intrusion task to evaluate the interpretability of our model. For this section, we only consider the datasets FB15k-237 and WN18RR due to their difficulty and the fact that they are the standard datasets in the current literature.

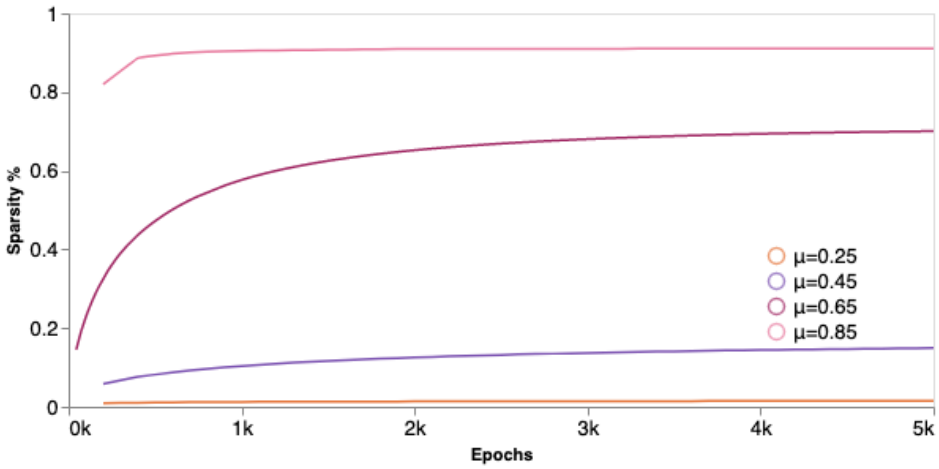
[1] gRDA can be reduced to RDA by setting $g(n, \gamma) = n\gamma$ and $\mathbf{w}_0 = 0$, which at the same time is an un-penalized version of SGD as setting $\mathcal{P}(\mathbf{w})$ and $F(\mathbf{w}) = \frac{1}{2}\{||w\}||_w^2$. In such a case, we could convert sRESCAL into RESCAL itself.



(a) mean rank on FB15k-237 for fixed hyperparameters except μ.



(b) distRatio on FB15k-237 for fixed hyperparameters except μ.



(c) sparsity percentage on FB15k-237 for fixed hyperparameters except μ.

Figure 4.1: Results on FB15k-237 for fixed hyperparameters except μ.

The word intrusion task evaluates coherence regarding the semantic meaning in each dimension belonging to the representations. Since sparse models create embeddings with only a few dimensions activated, i.e. the value is not 0, each dimension should correspond to a meaningful semantic concept. The task works in the following way: for each dimension i of the embeddings, it finds the k ($k \in \mathbb{N}$) most relevant Knowledge Graph entities, which are those that have the highest values in the corresponding dimension. Then, a non-relevant Knowledge Graph entity (one that haves a low value in the corresponding dimension) is chosen, and it is combined with the top-k entities to create a set of $k + 1$ entities for dimension i. We refer to the non-relevant Knowledge Graph entity as the intruder, and we seek to identify it. An example set of entities for $k = 5$: {Screenwriter, Film director, Film producer, Television producer, Actor, Erie} where *Erie* is the intruder entity since it is a city and does not belong to the TV and film industry. Human annotators usually perform the word intrusion task; however, we adopt the automatic version of the task presented in (Sun et al., 2016). In this version, the DistRatio evaluation metric is applied. The idea is that to find the intruder entity automatically, its distance to the top-k entities should be high. We measure the distance ratio between the intruder entity and the top-k entities to the distance between the top-k entities themselves. High ratio values mean better interpretability levels because the intruder entity is far (in the embedding space) from the top words (which are close due to their semantic meaning). The metric can formally be presented as:

$$DistRatio = \frac{1}{d}\sum_{i=1}^{d}\frac{InterDist_i}{IntraDist_i} \tag{4.8}$$

$$IntraDist_i = \sum_{w_j \in top_k(i)} \sum_{\substack{w_j \in top_k(i) \\ w_k \neq w_j}} \frac{dist(w_j, w_k)}{k(k-1)} \tag{4.9}$$

$$InterDist_i = \sum_{w_j \in top_k(i)} \frac{dist(w_j, w_{bi})}{k} \tag{4.10}$$

	Entity			Relation		
Model	Sparsity	DistRatio	Negativity	Sparsity	DistRatio	Negativity
Rescal	0	.056	.464	0	.052	.499
TuckER	0	.055	.509	0	**.059**	.503
sRESCAL	**.684**	**.067**	.094	**.954**	.047	.022

Table 4.4: Sparsity, DistRatio and negativity results for FB15k-237.

	Entity			Relation		
Model	Sparsity	DistRatio	Negativity	Sparsity	DistRatio	Negativity
Rescal	0	.051	.499	0	.061	.498
TuckER	0	.050	.507	0	.058	.508
sRESCAL	**.275**	**.053**	.335	**.436**	**.066**	.280

Table 4.5: Sparsity, DistRatio and negativity results for WN18RR.

where $top_k(i)$ denotes the top-k entities corresponding to dimension i, w_{bi} denotes the intruder entity for dimension i, $dist(w_j, w_k)$ is the distance between entities w_j and w_k, $IntraDist_i$ is the average distance between the top-k entities on dimension i and $InterDist_i$ denotes the average distance between the intruder entity and top-k words on dimension i. We set $k = 5$ and the distance function to be the euclidean distance.

We present the interpretability results on FB15k-237 and WN18RR in table 4.4 and table 4.5 respectively. We provide sparsity percentages, DistRatio and negativity percentages for entities and relations for sRESCAL, RESCAl and TuckER models. sRESCAL achieves the best results on DistRatio for both datasets, except in relations for FB15k-237, while maintaining high sparsity values. Both RESCAL and TuckER do not have any sparsity level since they are optimized using the Adam method. Furthermore, our experiments demonstrate the effectiveness of sparsity for developing more interpretable embeddings. Moreover, we present results on negativity since gRDA-l_1, while not constraining to positive values, enforces non-negativity on embeddings. Positivity is an important characteristic for the interpretability of the embeddings since human people do not describe a concept by what is not. Besides, positivity allows to perform additive combinations and

Model	Top-5 words
RESCAL	condition, device, fastener, quality, computer
	mammal family, unpleassant person, masculinization, photograph, process
	break, city, asterid dicot genus, right, meaning
	decoration, tract, mining, magic, improvement
	rescue, touch, ethnic group, stay, stand
sRESCAL	Bulldog, Golden Retriever, Labrador Retriever, Yorkshire Terrier, German Shepherd
	Finding Neverland, Showgirls, Remember the Titans, Cry Freedom, Shadowlands
	Woody Allen, Christopher Hitchens, George Carlin, Harold Pinter, Oliver Stone
	Electronic keyboard, Percussion, Electric guitar, Acoustic guitar, Guitar
	Miami Heat, Orlando Magic, San Antonio Spurs, Chicago Bulls, Boston Celtics

Table 4.6: Top 5 words of some dimensions in RESCAL and sRESCAL.

create representations from differents parts (Lee and Seung, 1999; Murphy et al., 2012). Results present sRESCAL as a highly non-negative method when compared to RESCAL and TuckER, improving interpretability in the embeddings.

We also provide a qualitative evaluation of the interpretability results. We select the top 5 words of a few dimensions in RESCAL and sRESCAL and present them in table 4.6. While the top 5 words provided by the standard RESCAL model do not have any coherence or semantic meaning, the sRESCAL model gets clear topics. From the first row to the fifth: dog breed, films, actors and directors, music topics (instruments) and basketball teams.

"Sometimes science is more art than science, Morty.
A lot of peeople don't get that."

- Rick Sánchez

CHAPTER

5

Influence functions for interpretable link prediction in Knowledge Graphs

Within this chapter we address the lack of interpretability of the KGE models with a data-point method. We find the most influential facts when predicting a new link (a new fact) in the Knowledge Graph. The method provides an interpretable and intuitive approach to understanding the link prediction process. Specifically, we try to solve the following questions for a new inferred fact: which were the most important facts to derive this one? How much did each fact matter? i.e. we look to give a quantity that leads a user to understand which were the most influential facts.

The chapter is divided into three sections: Section 5.1 provides an introduction of the method within the thesis and overviews influential instance methods. Then, Section 5.2 explains the procedure to identify the most influ-

ential facts. Finally, Section 5.3 illustrates the experiments and results of the interpretability tool.

5.1 An influential instances method for explaining model decisions

This chapter corresponds to the data-point categorisation from the taxonomy introduced in Section 1.1. A data point method includes all methods that return data points (already existent or newly created) to make a model interpretable. The technique presented herein can be classified as an influential instances technique or a counterfactual explanation method (where the features correspond to the entities and relations of the KG).

We propose a novel approach for KGs grounded on Influence Functions (IFs) (Koh and Liang, 2017). KG models assign a score to a possible fact when predicting new ones. The higher the score is, the more probable the fact is true. Using IFs, we approximate how much the scoring of a new fact would change if we removed a training sample (a training fact) without retraining the whole model. Nevertheless, such an approximation is computationally expensive and cannot scale to KGs where thousands and millions of training facts are used. Thus, we propose a novel technique, KGInfluence, to scale influence functions up for KGs efficiently. KGInfluence allows measuring the impact of training facts when inferring and increases the interpretability of KG link prediction models.

An influential instance is identified when its deletion from the training data significantly impacts the model's parameters. Consequently, the more the model's parameters are affected by a deletion, the more influential an instance is. The effect of an influential instance is shown in Figure 5.1. Although the figure represents a linear regression model, we could extrapolate to a KGE model where the y-axis can be understood as the score of a fact, and the X-axis contains the training facts.

In contrast to other interpretability techniques that focus on the instances' features, we focus on the instances themselves. We analyse the model's beha-

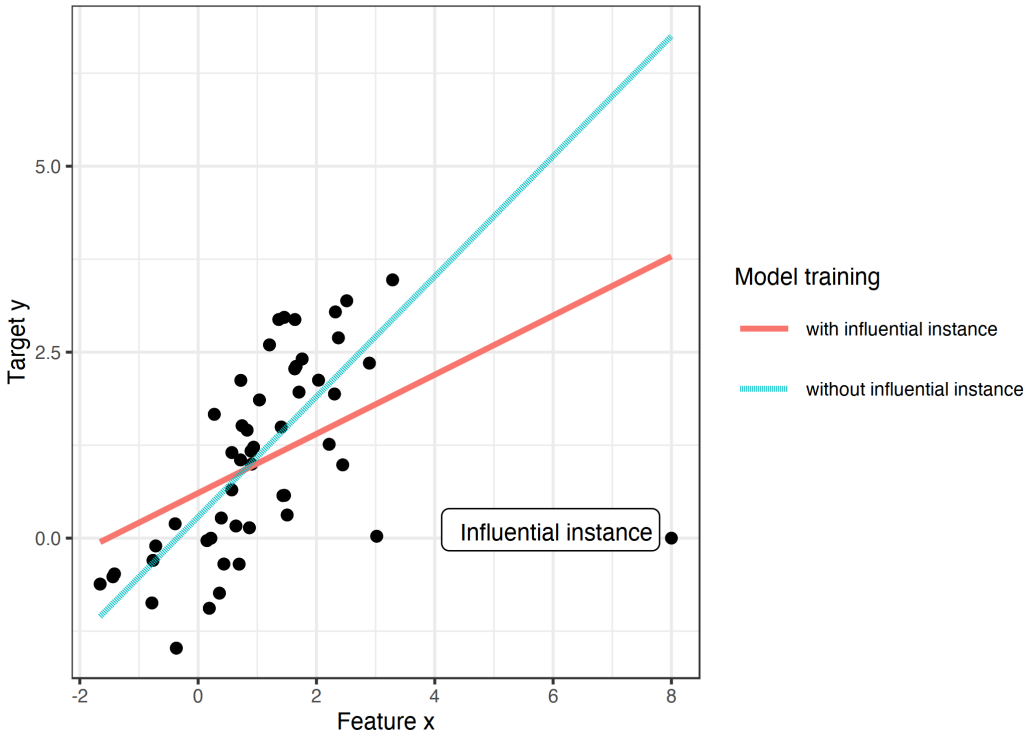


Figure 5.1: A linear model with one feature. Trained once on the full data and once without the influential instance. Removing the influential instance changes the fitted slope (weight/coefficient) drastically. From (Molnar, 2020).

viour by understanding the model as dynamic and a function of the training data. By identifying the influential instances, we can better interpret the models. Thanks to them, we can understand the model's behaviour globally and individually.

Identifying influential instances helps us find possible problems within the model and the training data. Furthermore, we can have an impression of the robustness of the whole model by looking at influential instances. For example, we should mistrust or at least investigate a model where a single instance significantly impacts the model's predictions.

Influence functions have several pros and applications:

- **Understanding model behaviour**: by using influence functions, we can analyse the model's behaviour globally and individually.

- **Debugging**: we can find possible errors or misbehaviours within our model.

- **Fixing training data**: we can identify wrong data and backtrack individual instances that need to be deleted or modified.

In summary, this chapter presents a scalable approach to applying Influence Functions to Knowledge Graph Embedding models. KGInfluence introduces an efficient method to increase interpretability in link prediction by providing the most influential data points for a given fact.

5.2 Procedure and Methodology

This section first introduces the background needed to present our approach. Then, it continues to describe our method, KGInfluence.

5.2.1 Knowledge Graph Embedding models

Suppose we are given a KG with a set of entities and relations $\mathcal{E}$ and $\mathcal{R}$, respectively. The collection of facts present in the KG being (s, r, o) where $s, o \in \mathcal{E}$ denote subject and object entities and $r \in yo\mathcal{R}$ is the relation between them.

For link prediction, models learn a scoring function in the form $s = \phi(e_s, r, e_o) \in \mathbb{R}$ which represents to what extent the model believes the fact is true. Such scoring functions can have different forms, e.g. a tensor factorisation method or a neural network. In this work, we focus on the RESCAL model (Nickel et al., 2011) which is one of the main approaches in the state-of-the-art. RESCAL is a tensor factorisation method with the scoring function:

$$\phi(e_s, r, e_o) = e_s^\top W_r e_o \tag{5.1}$$

where $e_s, e_o \in \mathbb{E}^{n_e \times d}$ are the subject and object embedding vectors and $W_r \in \mathbb{R}^{d \times d}$ is the relation embedding matrix, d is the embedding size.

5.2.2 Influence Functions

Consider a prediction problem from some input space $x \in \mathcal{X}$ for a set of output labels $y \in \mathcal{Y}$. Having a training set $z_i = (x_i, y_i) \in \mathcal{Z}$, we try to find the optimal model parameters on converge as $\hat{\theta} =^{def} \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta)$.

The goal of IFs is to identify the influence, i.e. the impact of the scoring function in the case of KGs, of training points in the model's prediction. To obtain such influence without having to retrain the whole model leaving a training point out, we look to up weight a training point z by an infinitesimal step ϵ. Then, we check the change of model parameters $\hat{\theta}_\epsilon - \hat{\theta}$:

$$\hat{\theta}_\epsilon \stackrel{\text{def}}{=} \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta) + \epsilon L\left(z, \theta\right) \tag{5.2}$$

From there, following (Cook and Weisberg, 1982) we have:

$$\left.\frac{d\hat{\theta}_\epsilon}{d\epsilon}\right|_{\epsilon=0} = -H_{\hat{\theta}}^{-1} \nabla_\theta L\left(z, \hat{\theta}\right) \tag{5.3}$$

where $H_{\hat{\theta}} \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta^2 L\left(z_i, \hat{\theta}\right)$ is the Hessian matrix and $\nabla_\theta L\left(z, \hat{\theta}\right)$ is the derivative of the loss at z with respect to θ. Refer to (Koh and Liang, 2017) to the whole derivation process for equation (5.3). Finally, if we follow (Koh and Liang, 2017), after applying the chain rule to study the change of model's prediction at a test point, x_t, we arrive at the influence of a training point z on such prediction:

$$INFL\left(z, x_t\right) = -\frac{1}{n} \nabla_\theta \hat{y}(x_t, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta}) \tag{5.4}$$

where $\hat{y}(x_t, \hat{\theta})$ is the prediction of the model on the test point x_t with parameters $\hat{\theta}$. We can use to approximate the influence of a training point without retraining the whole model.

An intuitive view of influence functions is shown in Figure 5.2, where we can check how do we perform a quadratic approximation of the actual loss to calculate the change in the model's parameter $\hat{\theta}_{-z}$ by upweighting $(-\frac{1}{n}\mathrm{I}_{\mathrm{up}\theta}(z))$ an instance z.

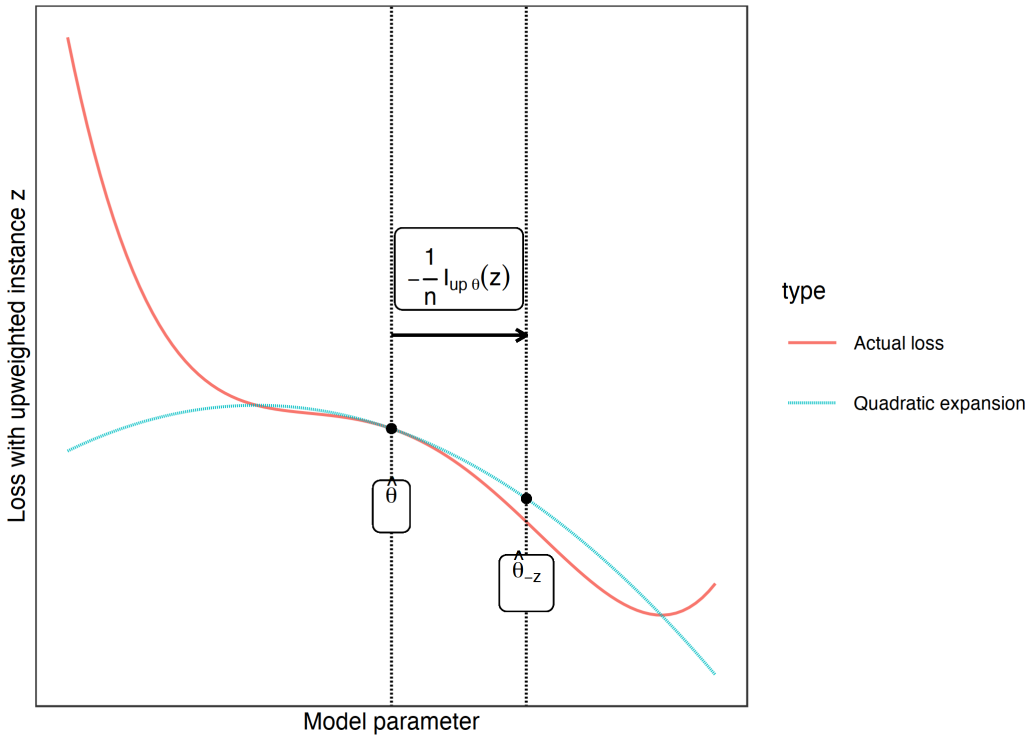


Figure 5.2: Updating the model parameter (x-axis) by forming a quadratic expansion of the loss around the current model parameter, and moving 1/n into the direction in which the loss with upweighted instance z (y-axis) improves most. This upweighting of instance z in the loss approximates the parameter changes if we delete z and train the model on the reduced data.. From (Molnar, 2020).

5.2.3 KGInfluence

5.2.3.1 Summary

We present the whole process for KGInfluence in Figure 5.3. The first step consists in training the KG embedding model. The training facts contained in the KG are fed to the embedding model, which is optimized to find the most optimal parameters. Then, once we have the optimal model trained, we can provide the model with a new test fact to predict. The model will return a score and predict whether the fact is true or not. To better understand such prediction, we apply the KGInfluence to obtain those training facts that were the most influential. We present the influence as the estimation of how much

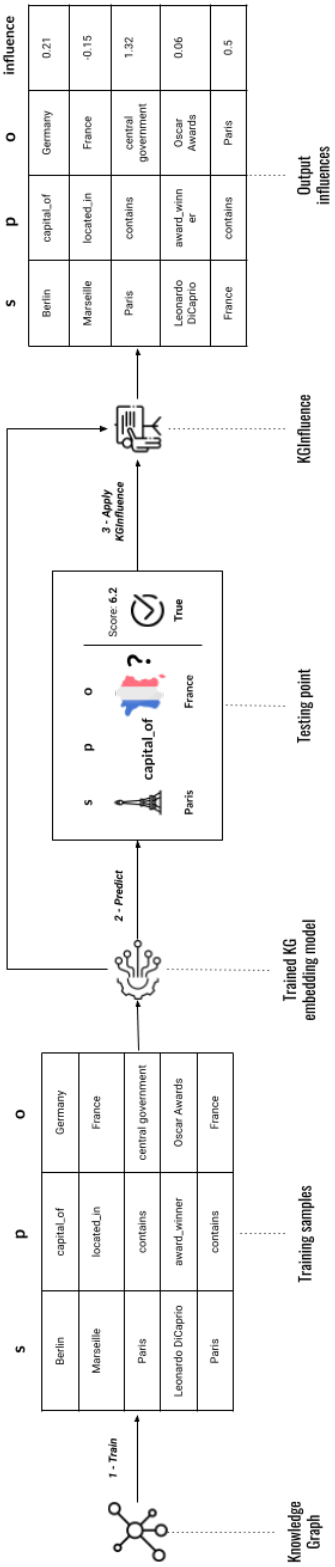


Figure 5.3: Summary and example for the KGInfluence process.

the scoring of the prediction fact would change if we removed a training fact.

We can also follow Figure 5.3 for an example, where the new fact to predict is (*Paris*, *capital_of*, *France*). The model returns a score of 6.2, which represents that the fact is *true*. After applying KGInfluence, we can approximate the influence of the training facts. In the example, we can observe the facts (*Paris*, *contains*, *central government*) and (*France*, *contains*, *Paris*) as the most important ones with an estimated influence of 1.32 and 0.5. However, we can also observe that the fact (*Marseille*, *located_in*, *France*) provides a negative influence value of -0.15. It makes sense for the model to identify that the central government is located in Paris is important to derive that Paris is, indeed, the capital of France. Furthermore, Marseille, which can be considered another important city of France, has a negative influence because it could make the model wrong.

5.2.3.2 **Basic approach**

In this section, we present the details of the KGInfluence approach. Inspired by the work (Cheng et al., 2019), we follow a similar idea and propose a scalable method for KG embeddings. In the training set $\mathcal{E}$, let $\hat{y}(s_t, p_t, o_t, \hat{\theta})$ be the model's scoring for a fact (s_t, p_t, o_t) under parameters $\hat{\theta}$. Our objective is to obtain $INFL(z, s_t, p_t, o_t)|z \in \mathcal{E}_t$, z being a training fact in the form of (s, p, o) and $\mathcal{E}_t \subseteq \mathcal{E}$ is the set of facts containing s, p or o, i.e. $s_t \vee p_t \vee o_t \in z, \forall z \in \mathcal{E}_t$. Following equation 5.4 we have

$$INFL(z, s_t, p_t, o_t) = -\frac{1}{n}\nabla_\theta \hat{y}(s_t, p_t, o_t, \hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta L(z, \hat{\theta}) \tag{5.5}$$

however, the equation above presents a challenge to overcome, the computation of $H_{\hat{\theta}}^{-1}$ where $H_{\hat{\theta}} \stackrel{\text{def}}{=} \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta^2 L\left(z_i, \hat{\theta}\right), z_i \in \mathcal{E}$. Computing the Hessian matrix is a costly operation, even more for KGs that can contain thousands or milions of elements. Computing $H_{\hat{\theta}}$ requires $\mathcal{O}(|z|p^2)$ where $|z|$ is the number of training points and $p = |z|n_e$ is the number of parameters in the model. Thus, the computation of $H_{\hat{\theta}}^{-1}$ requires $\mathcal{O}(|z|p^2 + p^3)$ which is infeasible for KGs.

Nevertheless, authors from (Koh and Liang, 2017) propose an efficient second order optimization technique for computing $H_{\hat{\theta}}^{-1}$. Based on the symmetry of $H_{\hat{\theta}}^{-1}$, we rewrite equation 5.5

$$INFL(z, s_t, p_t, o_t) = -\frac{1}{n}\nabla_\theta L(z, \hat{\theta})^\top H_{\hat{\theta}}^{-1}\nabla_\theta \hat{y}(s_t, p_t, o_t, \hat{\theta}) \tag{5.6}$$

now we can divide the equation 5.6 into two different parts: $H_{\hat{\theta}}^{-1}\nabla_\theta \hat{y}(s_t, p_t, o_t, \hat{\theta})$ and $\nabla_\theta L(z, \hat{\theta})$.

The first part, $H_{\hat{\theta}}^{-1}\nabla_\theta \hat{y}(s_t, p_t, o_t, \hat{\theta})$, is a Hessian-vector product that we transform to an optimization problem (we assume $H_{\hat{\theta}}$ is positive):

$$H_{\hat{\theta}}^{-1}\nabla_\theta \hat{y}(s_t, p_t, o_t, \hat{\theta}) = \arg\min_t \{\frac{1}{2}t^\top H_{\hat{\theta}} t - \nabla_\theta \hat{y}(s_t, p_t, o_t, \hat{\theta})^\top t\} \tag{5.7}$$

we can solve this by using Conjugate Gradients (CG). This approach does only require $\mathcal{O}(np)$ time (Martens et al., 2010), where n is the amount of training samples in the KG, and avoids computing and inverting the Hessian matrix.

The other part we divided from equation 5.6, $\nabla_\theta L(z, \hat{\theta})$, requires evaluating the gradient of the model for every training point z in $\mathcal{E}_t$ with parameters $\hat{\theta}$. This evaluation has complexity $\mathcal{O}(p|\mathcal{E}_t|)$.

Finally, we have to compute the influence of a training point in a given test fact $INFL(z, s_t, p_t, o_t)$. We follow equation 5.6 and perform the inner product between $H_{\hat{\theta}}^{-1}\nabla_\theta \hat{y}(s_t, p_t, o_t, \hat{\theta})$ and $\nabla_\theta L(z, \hat{\theta})$. After the analysis of both parts, for each training point we need $\mathcal{O}(p)$ operations, leading to a total complexity of $\mathcal{O}(p|\mathcal{E}_t|)$. $\mathcal{E}_t$ is the set of facts containing s,p or o, the amount of facts to compute is much smaller, i.e. $\mathcal{E}_t \lll \mathcal{E}$. Hence, the total computation time can noticeably be decreased.

5.2.3.3 **Reducing the complexity**

Nevertheless, the complexity $\mathcal{O}(np)$ derived from $H_{\hat{\theta}}^{-1}$ in the first part in equation 5.6 still remains a issue. The number of parameters p in a KG embedding model and all the facts n in the KG are very large. KGs like

FB15k (Bordes et al., 2013) contain up to 592,213 facts with 14,951 entities and 1,345 relations. Thus, we aim to reduce such complexity.

To do so, KGInfluence observes that KG embedding models grounded on tensor factorisation are only influenced by the latent parameters (the dimensions) of the embeddings. This is, for a given test point $x_t = (s_t, p_t, o_t)$ the parameters used to perform the scoring of the fact $\hat{y}(s_t, p_t, o_t)$ are $\theta_t = \{e_{s_t}, e_{p_t}, e_{o_t}\}$. The exact number of parameters depends on the KG embedding model, however, in this paper we use DistMult (Yang et al., 2014) which needs $3K$ parameters, K for the relation and each of the entities. Thus, we now update our modified version of equation 5.2 with θ_t

$$\hat{\theta}_{t,\epsilon} \stackrel{\text{def}}{=} \arg\min_{\theta_t} \frac{1}{n'} \sum_{j=1}^{n'} L(z_j, \theta_t) + \epsilon L\left(z, \theta\right) \tag{5.8}$$

where $z_j \in \mathcal{E}_t$ and $n' = |\mathcal{E}_t|$. The reason of this being that θ_t is only optimized by the facts containing s, p or o. If we follow the procedure substituing θ_t for equation 5.3 we arrive at the updated version of equation 5.6:

$$INFL(z, s_t, p_t, o_t) = -\frac{1}{n'} \nabla_{\theta_t} L(z, \hat{\theta})^\top H_{\hat{\theta}_t}^{-1} \nabla_{\theta_t} \hat{y}(s_t, p_t, o_t, \hat{\theta}) \tag{5.9}$$

Finally, we again divide equation 5.9 and review the computation complexity for each step and part:

Step 1. The computation complexity of $H_{\hat{\theta}_t}^{-1} \nabla_{\theta_t} \hat{y}(s_t, p_t, o_t, \hat{\theta})$ is now updated from $\mathcal{O}(np)$ to $\mathcal{O}(n'K)$. We only focus the facts containing s, p or o and their latent factors K.

Step 2. For computing $\nabla_{\theta_t} L(z, \hat{\theta})$ we need to evaluate all the training points in n' and their latent factors $3K$, thus the complexity is also $\mathcal{O}(n'K)$.

Step 3. The inner product over the steps 1 and 2 is needed to compute the final influence $INFL(z, s_t, p_t, o_t)$. Following those steps, the final influence is $\mathcal{O}(n'K)$.

Hence, we reduced the complexity of the basic approach from $\mathcal{O}(np)$ to $\mathcal{O}(n'K)$ allowing us to perform influence analysis for KGs.

Table 5.1: Datasets for link prediction and their number of entities and relations.

Dataset	$\mid \mathcal{E} \mid$	$\mid \mathcal{R} \mid$
FB15k-237	14,541	237
WN18RR	40,943	11

5.3 Experiments and discussion

In this section we present the performed experiments for KGInfluence. Before we present the experiments we provide an overview of the used datasets and implementation details.

5.3.1 Datasets

The datasets used for the evaluation are the following (see Table 5.1):

- **FB15k-237** (Toutanova et al., 2015) is a subset of the Freebase database which contains facts about the world. This is a better-suited version of the FB15k dataset where the inverse of many relations was removed to increase the difficulty.
- **WN18RR** (Dettmers et al., 2018) is a subset of WordNet, a hierarchical database containing lexical relations between words. Wn18RR, is a better-suited version of WN18 where the inverse of many relations was removed to increase the difficulty.

All the datasets were created for the link prediction task.

5.3.2 Implementation details

We open source the PyTorch implementation of the KGInfluence method on GitHub[1].

[1] https://github.com/unai-zulaika/KGInfluence

Table 5.2: Results obtained for KGInfluence in the FB15k-237 and WNRR datasets.

	FB15k-237			**WNRR**		
	Pearson´s R	Avg. Time (seconds)	Avg. diff. (in score)	Pearson´s R	Avg. Time (seconds)	Avg. diff. (in score)
KGInfluence	0.60	26	0.0038	0.17	4.87	0.015

To build and select the model we will apply KGInfluence on, we use the library (kge) provided by (Ruffinelli et al., 2019) which is considered a baseline work in the area. We choose DistMult (Yang et al., 2014) as our test model for KGInfluence due to its scalability. We search for the optimal hyper-parameters using SOBOL (pseudo-random) and Bayesian optimization. The final hyper-parameters for FB15k-237 are learning rate 0.15 using the Adagrad optimization method, embedding size of 256 and the Cross-Entropy loss. For WN18RR, we use learning rate of 0.33 using the Adagrad optimization method, embedding size of 512 and the Cross-Entropy loss. The exact hyper-parameter configuration can be found in the released code.

There were also two hyper-parameters to search for when applying KGInfluence. First, we applied a damping term of $1e-35$ and $1e-06$ for FB15k-237 and WN18RR, respectively, to avoid negative eigenvalues in Hessian. Then, we tried solving equation 5.7 using Conjugate Gradient and Trust-Region Conjugate-Gradient and found the standard Conjugate Gradient to be more optimal in every case.

5.3.3 **Efficiency**

To our knowledge, KGInfluence is the first scalable application of Influence Functions to KGs, even though CRIAGE (Pezeshkpour et al., 2019) proposes to compute the influence of a KG fact by using a first-order Taylor approximation to perform adversarial attacks. As CRIAGE only considers predicting influences on the object o of a fact s, p, o, while KGInfluence considers all the facts containing s, p or o, several aspects make the comparison between CRIAGE and KGInfluence unfair. Thus, we decided to leave a further analysis between the two methods for future work.

Recall that KGInfluence estimates the influence, $INFL(z, s_t, p_t, o_t)$, of the neighbours z of a fact , s_t, p_t, o_t on a trained KG embedding model. Such influence can be used to generate explanations of the model's predictions. To do so, we first evaluate the effectiveness of KGInfluence on its estimations. The most reliable approach to evaluate this effectiveness is to remove the training fact z and retrain the whole KG embedding model. We then compare the estimated influence with the real observed influence by leave-one-out retraining. We first select 30 test points from the test set to start the process. For each of those test points, we apply KGInfluence to get the influence $INFL(z, s_t, p_t, o_t)|z \in \mathcal{E}_t$, i.e. the set of training facts containing either s, p or o. After ordering by influence value, we retrain the KG embedding model without the most influential training fact and compare the observed influence with the estimated one. We use the Pearson correlation coefficient, Pearson's R, to provide a metric between the estimated influence and the observed one.

We conduct experiments for KGInfluence on both datasets, FB15k-237 and WNRR. Results are presented in table 5.2. As we expected, we obtained low Pearson's R values, especially in the case of WNRR. Link prediction with Knowledge Graph embeddings is a challenging task with models that do not only try to predict a score but rank such fact as highest as possible when compared with every other possible fact. Furthermore, the selected datasets are large, and the amount of training facts difficults estimating the influence. Additionally, we used the full expressive version of the selected model DistMult with embeddings dimensions of 256 and 512 for FB15k-237 and WNRR. Such dimensionality is considered large and increases the difficulty of the influence estimation. We select this dimensionality on purpose and avoid simplifying the model as this work aims to provide an influence estimation method for real settings and applications.

For FB15k-237 we obtain a Pearson's R of 0.60, which provides a good correlation value for the dataset. The distribution of the dataset allows performing the influence estimation on a smaller amount of facts than WNRR. Thus, we obtain better results on correlation. Besides, the average time to compute the influence of every training fact $z \in \mathcal{E}_t$ for a test fact$(s_t, p_t, o_)$ is

low, 26 seconds, which demonstrates the scalability of our approach. Furthermore, The average differences between the predicted and observed scores is low.

For WN18RR, however, KGInfluence achieves a low Pearson's R value, 0.17. The reasons presented above and the data distribution are the main factors. For each of the test points, the number of facts $z \in \mathcal{E}_t$ that need to be computed for the influence estimation consistently reaches a thousand instances. Nevertheless, the average time to compute the influence remains low, less than 5 seconds. This is due to the Conjugate Gradient algorithm not finding the best possible optimization and thus, finishing the algorithm quickly. This problems leads to the low Pearson's R value and to have a lower average time for estimating influence than FB15k-237, even though in WNRR many more train facts need to be evaluated.

Nonetheless, KGInfluence achieves convincing results in practice, as we show in the following subsection.

5.3.4 **Examples and conclusions**

In this subsection, we briefly overview a set of examples for the KGInfluence method. Table 5.3 presents the results provided by our method. For a target triple, we present the estimated three most influential facts in the dataset. Due to the nature of the datasets, the presented triples are related to different areas. Triples range from movies to location and verbs or hypernyms. Identifying why a model could select the most influential triples for each fact is relatively easy. For example, for the triple *Winnie the Pooh, film_release_region, France*, it is easy to identify that the most influential triples were also cartoon movies that were released in France. The interpretability level provided by these examples is powerful and relevant for end-users. We argue that in many cases, the use of machine learning tools in citizens' lives must be accompanied by interpretability tools that make them more trustable and user-friendly. Thus, we believe KGInfluence will end-users understand better the technology they are using.

Finally, several thesis objectives were addressed in this chapter. We accomplished (**O1**) by studying the state-of-the-art methods and techniques for Influence Functions. Objective (**O4**) was fulfilled as we presented a novel data point method applied to Knowledge Graphs. Additionally, objectives (**O5**) and (**O6**) were met by the efficiency analysis we provided (the link prediction task was not evaluated as the method does not impact the performance of the model).

Table 5.3: Example of the interpretability provided by KGInfluence. We provide the three most influential triples for a target triple.

	FB15k-237	**WNRR**
Target triple (s, p, o)	Winnie the Pooh, film_release_region, France	telephone, _verb_group, call
Most influential triples by order	Wreck-It Ralph, film_release_region, France	take_in, _verb_group , sack_up
	Brave, film_release_region, France	work, _verb_group , work
	Monsters, Inc., film_release_region, France	use, _verb_group , utilize
Target triple (s, p, o)	England, contains, Lancaster (United Kingdom)	breadfruit_tree, _hypernym, fruit_tree
Most influential triples by order	England, adjoins, United Kingdom	turbine, _hypernym , rotary_engine
	England, contains, Argyll and Bute	lysergic_acid_diethylamide, _hypernym , psychodelic_drug
	England, contains, Royal Academy	invocation, _hypernym , incantation
Target triple (s, p, o)	Autoharp, music/group_membership/group, Heart (Hard rock Artist)	take_a_breather, _derivationally_related_form, breathing_time
Most influential triples by order	Harmonica, music/group_membership/group, Pearl Jam	word, _derivationally_related_form , set_phrase
	Keyboard, music/group_membership/group, Fleetwood Mac	Keyboard, refine, _derivationally_related_form , refinement
	Lead vocalist, music/group_membership/group, Royal Genesis	packer, _derivationally_related_form , pack

"Esto no es nada sin ustedes."

- Dellafuente

CHAPTER

6

Conclusions and Future Work

This chapter presents a summary of the thesis and its contributions. We revisit the hypothesis and objectives to analyse to what extent they have been completed. Besides, we summarise the obtained contributions and publications during the thesis development. Finally, the chapter ends with a review of the remaining challenges and issues to overcome and proposes a line of future work.

The rest of this chapter is structured as follows: Section 6.1 summarises the work done and conclusions obtained from this thesis. Section 6.2 lists the main contributions of this dissertation. Section 6.3 analyses the objectives stated at the beginning of this dissertation and to what extent they have been fulfiled. Next, section 6.4 lists all scientific publications published during the development of this dissertation. Section 6.5 introduces possible future research. Finally, section 6.6 makes some final remarks.

6.1 Summary of work and conclusions

Knowledge Graphs are powerful data modelling tools widely used in many areas. The use of Machine Learning over those KGs is a common practice to offer new and enhanced services to citizens and end-users. Furthermore, interpretability is a crucial aspect of those KG-based Machine Learning models. It provides end-users with the needed methods to understand the decisions that machines make and impact their lives. Several scenarios, needs, and models require different interpretability techniques. However, we identify a lack of interpretability tools and approaches applied to KG-based models. So, the objective of this dissertation has been to develop a new set of tools that is flexible and applicable to multiple situations. In this manner, we can have filled in the gap in the research intersection between KGs and interpretability.

Intending to offer a set of tools that match different scenarios and needs, this thesis has followed the taxonomy presented in Chapter 1 and Figure 1.2. Such taxonomy classifies the interpretability techniques according to the type of outcome they produce. By providing different outcomes and results, we can suit the different scenarios that end-users might face and help them understand our models. Following the taxonomy, we have named each of the categories: **feature summary statistics**, **feature summary visualisation**, **model internals**, **data point** and **intrinsically interpretable model**.

We presented the proposed method for feature summary statistics and visualisation in chapter 3. To predict link weights in standard graphs, we developed a novel scalable approach that uses subgraph extraction and CNNs. We leverage the Weisfeiler-Lehman graph colouring technique to learn the importance of nodes in a subgraph. Then, we order the subgraph's nodes using such importance and feed the ordered nodes to a neural model which predicts link weights. The resulting importance value serves as an interpretable indicator of the role of a node in the subgraph. Additionally, the importance of every node can also be plotted to visualise and interpret how the whole subgraph behaves. We present the results on link weight prediction and the interpretability of the method within the chapter 3. This technique is appropriate for scenarios where the model needs to be interpreted at an overview

level, quickly understanding the role of each node in prediction time.

In chapter 4, we develop a method to have interpretable internal representations in KG-based models for the link prediction task. Grounding on cognitive arguments, we present a technique to have highly sparse and almost non-negative representations for the concepts within a KG, e.g. *Paris* or *Marie Curie*. Our technique uses the gRDA-l_1 optimisation algorithm and is applicable to different link prediction models. The obtained internal representations are semantically grouped and better interpreted with this method. The results presented in the Section 4.3 show that our method highly increases interpretability while maintaining competitive results on the link prediction task. This method is suitable for situations where high coherence and end to end interpretability is essential.

Finally, in chapter 5, we provide a data point method for link prediction in KGs. We leverage the power of influence functions to find the most important (influential) training samples for a given prediction. Furthermore, we present an approach to make influence estimation scalable and effective for large KGs. The presented results demonstrate the effectiveness of our approach and a set of use cases and examples applicable to the system. The method is suited for use cases where end-users have a low understanding of the system, and providing examples is useful and builds trust for them.

In summary, this PhD dissertation has presented a novel interpretability method applied to graphs and KGs for each of the categories within the taxonomy presented in Section 1.1 (except for the intrinsically interpretable model) that categorises methods by their output result. This set of interpretability tools allows researchers and developers to investigate further and enhance the explainability of their models.

6.2 Contributions

As illustrated in previous chapters of this dissertation, the knowledge and findings obtained from this thesis are diverse. As a consequence of those findings, the scientific and technical contributions generated from this dissertation are

presented in this section. Those contributions are linked to their respective objectives, described in Section 1.2:

(**C1**) *A novel link weight prediction framework that learns from graph structure features and achieves state-of-the-art results on various datasets.* This contribution is related to **O2** and corresponds to Section 3.2. We leverage the power of the Weisfeiler-Lehman graph colouring technique to improve the performance of our model and, at the same time, to find a feature statistic that can be used to improve interpretability.

(**C2**) *A feature summary statistic and visualisation method for the link weight prediction framework.* This contribution corresponds to **O2** and corresponds to 3.4. After developing the novel method for graph link weight prediction (the previous contribution), we leverage the node role statistic and provide an interpretability tool to understand better the predictions given by our model.

(**C3**) *We provide the results on interpretability and the link weight prediction task.* This contribution corresponds to **O5** and **O6**, and is presented in Section 3.3. We provide the results on the link weight prediction task compared to the other state-of-the-art methods and analyse how interpretability affects our model.

(**C4**) *We present an adaption of the optimisation algorithm named generalised Regularized Dual Averaging for Knowledge Graph and demonstrate its effectiveness.* Related to **O3** and presented in Section 4.2, we propose to use such adaption to learn sparse and non-negative model internals for KG-based Machine Learning models.

(**C5**) *We present the first sparse linear Knowledge Graph Embedding model, which increases interpretability and remains competitive with the state-of-the-art results for link prediction.* This contribution is directly connected to the **O3** and it also presented in Section 4.2. We use the generalised Regularized Dual Averaging method to develop semantically interpretable representations of the concepts within a Knowledge

Graph. With those representations being the core of every Knowledge Graph Embedding model, we provide a tool to learn interpretable internals for those models.

(**C6**) *An extensive evaluation and comparison between dense Knowledge Graph Embedding models and sRESCAL (our model).* This contribution is related to the objectives **O5** and **O6**, and corrresponds to Section 4.3. We present an in-depth evaluation of the differences between the non-sparse and the sparse model obtained after using the proposed optimisation method.

(**C7**) *We propose a novel approach to scale influence functions and make them efficient for Knowledge Graphs.* Connected with objective **O4** and corresponding to Section 5.2, this technique drastically reduces computation time and improves the performance of influence functions in KGs.

(**C8**) *To our knowledge, we propose the first use of Influence Functions for Knowledge Graphs, opening a new method for interpreting link prediction models.* Related to **O4** and presented in Section 5.2, we enhance the interpretability of the link prediction models for Knowledge Graphs. To make Influence Functions viable for Knowledge Graphs, we use the previous contribution. In this way, we can easily identify the most influential samples for a given prediction.

(**C9**) *We provide an extensive evaluation of the proposed technique to demonstrate the performance of KGInfluence and the enhanced interpretability.* Coupled with **O5** and **O6**, and presented in Section 5.3. The conducted experiments allow us to understand Knowledge Graph embedding methods better. However, Influence Functions do not affect a model's performance in prediction time.

A map for contributions, objectives and publications is presented in Figure 6.1.

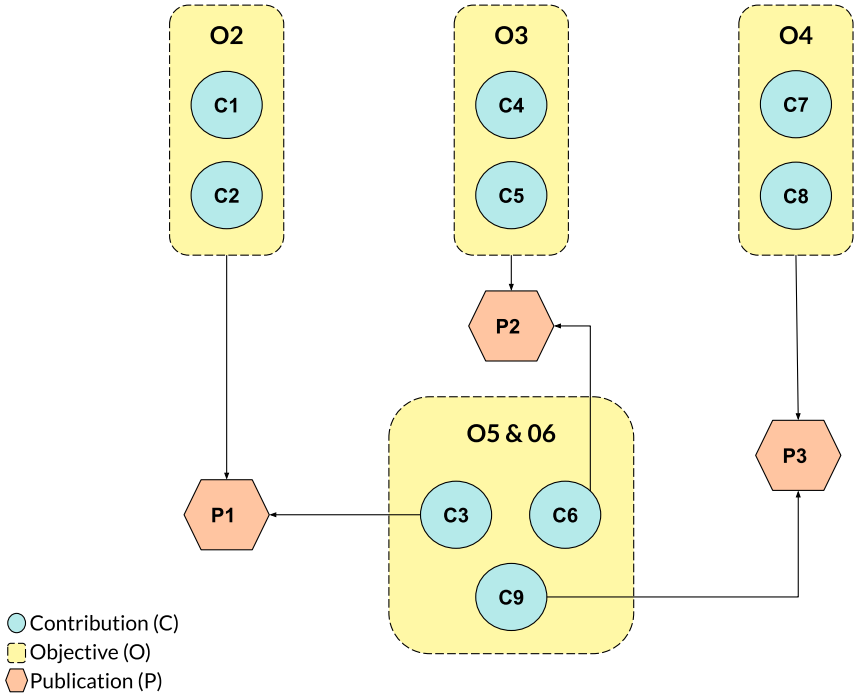


Figure 6.1: Contributions, objectives and publications map.

6.3 Hypothesis and objective validation

At the beginning of the dissertation, in Section 1.2, the following hypothesis was formulated:

> **Hypothesis.** For specific application needs and models applied to graphs and knowledge graphs, developing novel techniques focusing on output results makes it possible to increase users' interpretability at different levels.

Then, in order to validate it, the following goal was proposed:

Goal. To design and implement a set of interpretability tools for graphs and knowledge graphs that are suitable for specific scenarios and provide different output results.

To achieve this goal, a set of specific objectives were determined. We further provide a summary of how each specific sub-objective has been fulfilled:

(**O1**) *To study the current state of the art on machine learning applied to graphs and KGs and their interpretability techniques.* In Chapter 2 we present the most relevant works in KGs, interpretability and their intersection. We divide the chapter into those three sections. Additionally, each of the Chapters 3, 4 and 5 provides the needed theoretical background related to KG-based models and the interpretability method they present.

(**O2**) *To design and implement a feature summary statistics method applicable to graph or KG-based ML models.* In Section 3.2 we present a method for feature summary statistic and visualisation applied to our link weight prediction model. We have achieved state-of-the-art results on the task and provide an interpretability tool for the model.

(**O3**) *To design and implement a technique to make model internals (learned representations) more interpretable in KG-based ML models.* Section 4.2 introduces a novel optimisation method applied to KG link prediction models. This method develops sparse and non-negative representations (weights) of the entities and concepts within a KG. Those representations are semantically grouped and related and can make the models internally interpretable.

(**O4**) *To design and implement a data point method applicable to KG-based ML models.* Section 5.2 proposes a novel technique that applies Influence Functions to KG-based models. This technique allows identifying the most important (influential) training triples for a given prediction and offering end-users a data point explanation.

(**O5**) *To provide an appropriate evaluation methodology for each implemented method and their corresponding tasks.* Sections 3.3.3, 4.3.3 and 5.3 present the corresponding evaluation for each of the proposed methods from the taxonomy, in order: feature summary statistic, model internals and data point. Every method is evaluated on the task and interpretability wise (although some methods do not affect task performance).

(**O6**) *To analyse the impact of the implemented interpretability tools in the current state-of-the-art models performance-wise.* Sections 3.4, 4.3.4 and 5.3 provide an in-depth analysis of the impact of the methods on the corresponding tasks. It is common (although not always the case) for an interpretability method to affect task performance. Thus, we evaluate the tradeoff between task performance and enhancing interpretability with the proposed techniques.

Finally, by accomplishing these objectives and consequently the goal, the hypothesis of this dissertation has been confirmed. Results and evaluation for each of the methods in the followed taxonomy demonstrate that interpretability can be increased and further enhanced for graph and KG-based models. Furthermore, those methods are varied and suitable for different scenarios and use cases. Thus, we expect the work developed in this dissertation contribute to improving the use of KG-based models for citizens and users by laypeople and data scientists.

6.4 Relevant publications

During the development of this dissertation, several scientific manuscripts have been written, accepted and presented to the scientific community. In what follows, the list of published articles is shown.

6.4.1 International JCR Journals and Book Chapters

The work on the state-of-the-art link weight prediction model and the feature summary statistics and visualisation method was published.

(**P1**) Zulaika, Unai, Rubén Sánchez-Corcuera, Aitor Almeida, and Diego López-de-Ipiña. "LWP-WL: Link weight prediction based on CNNs and the Weisfeiler–Lehman algorithm." Applied Soft Computing 120 (2022): 108657. JCR Impact Factor (2021): 8.263, Q1.

Furthermore, the technique for developing an interpretable internal model that achieves sparse and non-negative KG representations was published in the following journal:

(**P2**) Zulaika, Unai, Aitor Almeida, and Diego López-de-Ipiña. "Regularized Online Tensor Factorization for Sparse Knowledge Graph Embeddings." Neural Computing and Applications (2022). JCR Impact Factor (2021): 5.102, Q1.

Finally, a parallel work about social networks and the role of features and nodes within those graphs was published:

- Sánchez-Corcuera, Rubén, Aritz Bilbao-Jayo, Unai Zulaika, and Aitor Almeida. "Analysing centralities for organisational role inference in online social networks." Engineering Applications of Artificial Intelligence 99 (2021): 104129. JCR Impact Factor (2021): 7.802, Q1.

6.4.2 **International Conferences**

The preliminary research that served as inspiration for this dissertation:

- Zulaika, Unai, Asier Gutiérrez, and Diego López-de-Ipiña. "Enhancing profile and context aware relevant food search through knowledge graphs." Multidisciplinary Digital Publishing Institute Proceedings 2.19 (2018): 1228.

Additionally, the data point method, grounded on applying Influence Functions to KGs, was published:

(**P3**) Zulaika, Unai, Aitor Almeida, and Diego López-de-Ipiña. "Influence Functions for Interpretable link prediction in Knowledge Graphs for Intelligent Environments" Proceedings of the 7th International Conference on Smart and Sustainable Technologies (Splitech 2022). p. 7.

6.4.3 Technical Contributions

- The source code of this dissertation is released in Github[1][2][3]. The code provides the three developed methods and a wide set of Knowledge Graph Embedding techniques.

- The sparse and almost non-negative version of the Rescal embeddings are accessible in the code repository[2].

6.5 Future work

Encouraged by the limitations and experiences learned throughout this dissertation, the following research lines and future work have been identified.

- *Towards explainable systems*: at the beginning of this dissertation, the difference between explainability and interpretability was explained. We focused on developing interpretable methods as a first step to bringing graph and Knowledge Graph models closer to end-users. However, we believe that to enhance understanding of models, they should explain their decision-making processes themselves. For example, the link weight prediction model could report why it predicted a weight by identifying the most critical nodes in a subgraph or explaining the topology. In the case of the interpretable weights for Knowledge Graphs, the model could output that the prediction of (*Paris, capital_of, ?*) is *France* comes from the words Paris and France being related or connected throughout other triples. Thus, we expect future work on developing explainable techniques and methods for graphs and Knowledge Graph based models.

- *To leverage the developed interpretability tools to research new interpretability methods in more challenging tasks*: throughout the thesis

[1] https://github.com/unai-zulaika/LWP-WL
[2] https://github.com/unai-zulaika/NNSKGE
[3] https://github.com/unai-zulaika/KGInfluence

work, we developed and presented various techniques that serve as fundamental tools for improving interpretability. Those techniques were developed for link prediction tasks and embedding models. Likewise, embeddings are used in many other tasks and areas. Thus, we consider that our work can be leveraged for more challenging tasks such as complex question answering or entity recognition.

- *Further extend the evaluation systems to real-world use cases with users*: in this work, we presented different evaluations for the methods we provide. It was out of the scope of this work to perform user-based evaluations. However, we consider that receiving user feedback would be beneficial in identifying possible gaps and limitations of the developed techniques. We leave for future work to test the methods in user-based pilots and testbeds.

6.6 Final remarks

This work was devised to contribute to interpretability and Knowledge Graphs research. We strongly believe that Machine Learning is a powerful tool that enhances people's lives and that Knowledge Graphs are flexible for managing real-world information. Thus, we contribute to increasing the interpretability of KGs, allowing humans to increase their control and understanding of the modern tools that empower their everyday routine. Furthermore, as we already explained in the dissertation, increasing interpretability also helps achieving properties such as trust, fairness, privacy and reliability.

Bibliography

Aicher, C., Jacobs, A. Z., and Clauset, A. (2014). Learning latent block structure in weighted networks. *Journal of Complex Networks*, 3(2):221–248.

Allen, C., Balazevic, I., and Hospedales, T. M. (2019). On understanding knowledge graph representation. *arXiv preprint arXiv:1909.11611*.

Balakrishnan, R. and Ranganathan, K. (2012). *A textbook of graph theory*. Springer Science & Business Media.

Balažević, I., Allen, C., and Hospedales, T. M. (2019). Tucker: Tensor factorization for knowledge graph completion. *arXiv preprint arXiv:1901.09590*.

Barbieri, N., Bonchi, F., and Manco, G. (2014). Who to follow and why: link prediction with explanations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1266–1275.

Batagelj, V. and Mrvar, A. (2014). Pajek. *Encyclopedia of Social Network Analysis and Mining*, pages 1245–1256.

Bau, D., Zhou, B., Khosla, A., Oliva, A., and Torralba, A. (2017). Network dissection: Quantifying interpretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6541–6549.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM.

Bordes, A., Glorot, X., Weston, J., and Bengio, Y. (2014). A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26.

Carvalho, D. V., Pereira, E. M., and Cardoso, J. S. (2019). Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832.

Chao, S.-K. and Cheng, G. (2019). A generalization of regularized dual averaging and its dynamics. *arXiv preprint arXiv:1909.10072*.

Cheng, W., Shen, Y., Huang, L., and Zhu, Y. (2019). Incorporating interpretability into latent factor models via fast influence analysis. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 885–893.

Colizza, V., Pastor-Satorras, R., and Vespignani, A. (2007). Reaction–diffusion processes and metapopulation models in heterogeneous networks. *Nature Physics*, 3(4):276.

Cook, R. D. and Weisberg, S. (1982). *Residuals and influence in regression*. New York: Chapman and Hall.

De Sá, H. R. and Prudêncio, R. B. (2011). Supervised link prediction in weighted networks. In *The 2011 international joint conference on neural networks*, pages 2281–2288. IEEE.

Dettmers, T., Minervini, P., Stenetorp, P., and Riedel, S. (2018). Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Doshi-Velez, F. and Kim, B. (2017). Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159.

Duma, M. and Twala, B. (2018). Optimising latent features using artificial immune system in collaborative filtering for recommender systems. *Applied Soft Computing*, 71:183–198.

Faruqui, M., Tsvetkov, Y., Yogatama, D., Dyer, C., and Smith, N. A. (2015). Sparse overcomplete word vector representations. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1491–1500, Beijing, China. Association for Computational Linguistics.

Fensel, D., Şimşek, U., Angele, K., Huaman, E., Kärle, E., Panasiuk, O., Toma, I., Umbrich, J., and Wahler, A. (2020). Introduction: what is a knowledge graph? In *Knowledge Graphs*, pages 1–10. Springer.

Fu, C., Zhao, M., Fan, L., Chen, X., Chen, J., Wu, Z., Xia, Y., and Xuan, Q. (2018). Link weight prediction using supervised learning methods and its application to yelp layered network. *IEEE Transactions on Knowledge and Data Engineering*, 30(8):1507–1518.

Ghosh, P., Saini, N., Davis, L. S., and Shrivastava, A. (2021). Learning graphs for knowledge transfer with limited labels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11151–11161.

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., and Kagal, L. (2018). Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE.

Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2015). Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *journal of Computational and Graphical Statistics*, 24(1):44–65.

Grover, A. and Leskovec, J. (2016). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864.

Guo, S., Wang, Q., Wang, B., Wang, L., and Guo, L. (2015). Semantically smooth knowledge graph embedding. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 84–94.

Gusmão, A. C., Correia, A. H. C., De Bona, G., and Cozman, F. G. (2018). Interpreting embedding models of knowledge bases: A pedagogical approach. In *2018 ICML Workshop on Human Interpretability in Machine Learning (WHI 2018)*, Stockholm, Sweden.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017a). Inductive representation learning on large graphs. *arXiv preprint arXiv:1706.02216*.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017b). Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

Harshman, R. A. (1978). Models for analysis of asymmetrical relationships among n objects or stimuli. In *First Joint Meeting of the Psychometric Society and the Society of Mathematical Psychology, Hamilton, Ontario, 1978*.

He, S., Liu, K., Ji, G., and Zhao, J. (2015). Learning to represent knowledge graphs with gaussian embedding. In *Proceedings of the 24th ACM international on conference on information and knowledge management*, pages 623–632.

Heider, F. and Simmel, M. (1944). An experimental study of apparent behavior. *The American journal of psychology*, 57(2):243–259.

Holland, P. W., Laskey, K. B., and Leinhardt, S. (1983). Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137.

Hou, Y. and Holder, L. B. (2017). Deep learning approach to link weight prediction. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1855–1862. IEEE.

Huang, Q., Yamada, M., Tian, Y., Singh, D., Yin, D., and Chang, Y. (2020). Graphlime: Local interpretable model explanations for graph neural networks. *arXiv preprint arXiv:2001.06216*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jeong, H., Mason, S. P., Barabási, A.-L., and Oltvai, Z. N. (2001). Lethality and centrality in protein networks. *Nature*, 411(6833):41.

Ji, S., Pan, S., Cambria, E., Marttinen, P., and Philip, S. Y. (2021). A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE Transactions on Neural Networks and Learning Systems*.

Kadlec, R., Bajgar, O., and Kleindienst, J. (2017). Knowledge base completion: Baselines strike back. *arXiv preprint arXiv:1705.10744*.

Kim, B., Khanna, R., and Koyejo, O. O. (2016). Examples are not enough, learn to criticize! criticism for interpretability. *Advances in neural information processing systems*, 29.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kipf, T. N. and Welling, M. (2016a). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kipf, T. N. and Welling, M. (2016b). Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*.

Koh, P. W. and Liang, P. (2017). Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR.

Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM review*, 51(3):455–500.

Kong, Z. and Chaudhuri, K. (2021). Understanding instance-based interpretability of variational auto-encoders. *Advances in Neural Information Processing Systems*, 34.

Lacroix, T., Obozinski, G., and Usunier, N. (2020). Tensor decompositions for temporal knowledge base completion. *arXiv preprint arXiv:2004.04926*.

Lee, C. Y. (1961). An algorithm for path connections and its applications. *IRE transactions on electronic computers*, (3):346–365.

Lee, D. D. and Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791.

Li, W. and Cai, X. (2004). Statistical analysis of airport network of china. *Physical Review E*, 69(4):046106.

Lin, Y., Liu, Z., Luan, H., Sun, M., Rao, S., and Liu, S. (2015). Modeling relation paths for representation learning of knowledge bases. *arXiv preprint arXiv:1506.00379*.

Liu, S., Ji, X., Liu, C., and Bai, Y. (2017). Similarity indices based on link weight assignment for link prediction of unweighted complex networks. *International Journal of Modern Physics B*, 31(2):1650254.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30.

Martens, J. et al. (2010). Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742.

Miller, K., Jordan, M. I., and Griffiths, T. L. (2009). Nonparametric latent feature models for link prediction. In *Advances in neural information processing systems*, pages 1276–1284.

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38.

Miotto, R., Wang, F., Wang, S., Jiang, X., and Dudley, J. T. (2018). Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics*, 19(6):1236–1246.

Molnar, C. (2020). *Interpretable machine learning*. Lulu. com.

Murata, T. and Moriyasu, S. (2007). Link prediction of social networks based on weighted proximity measures. In *Proceedings of the IEEE/WIC/ACM international conference on web intelligence*, pages 85–88. IEEE Computer Society.

Murphy, B., Talukdar, P., and Mitchell, T. (2012). Learning effective and interpretable semantic models using non-negative sparse embedding. In *Proceedings of COLING 2012*, pages 1933–1950.

Murphy, G. (2004). *The big book of concepts*. MIT press.

Newman, M. E. (2001). The structure of scientific collaboration networks. *Proceedings of the national academy of sciences*, 98(2):404–409.

Newman, M. E. (2003). The structure and function of complex networks. *SIAM review*, 45(2):167–256.

Nguyen, D. Q., Nguyen, T. D., Nguyen, D. Q., and Phung, D. (2017). A novel embedding model for knowledge base completion based on convolutional neural network. *arXiv preprint arXiv:1712.02121*.

Nickel, M. and Kiela, D. (2018). Learning continuous hierarchies in the lorentz model of hyperbolic geometry. In *International Conference on Machine Learning*, pages 3779–3788. PMLR.

Nickel, M., Murphy, K., Tresp, V., and Gabrilovich, E. (2015). A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33.

Nickel, M., Tresp, V., and Kriegel, H.-P. (2011). A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 809–816, Madison, WI, USA. Omnipress.

Nickel, M., Tresp, V., and Kriegel, H.-P. (2012). Factorizing yago: scalable machine learning for linked data. In *Proceedings of the 21st international conference on World Wide Web*, pages 271–280.

Niepert, M., Ahmed, M., and Kutzkov, K. (2016). Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023.

Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., and Mordvintsev, A. (2018). The building blocks of interpretability. *Distill*, 3(3):e10.

Opsahl, T. and Panzarasa, P. (2009). Clustering in weighted networks. *Social networks*, 31(2):155–163.

Osaba, E., Del Ser, J., Camacho, D., Bilbao, M. N., and Yang, X.-S. (2020). Community detection in networks using bio-inspired optimization: Latest developments, new results and perspectives with a selection of recent metaheuristics. *Applied Soft Computing*, 87:106010.

Otter, D. W., Medina, J. R., and Kalita, J. K. (2020). A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Palmonari, M. and Minervini, P. (2020). Knowledge graph embeddings and explainable ai. *Knowledge Graphs for Explainable Artificial Intelligence: Foundations, Applications and Challenges*, 47:49.

Pan, R. K., Kaski, K., and Fortunato, S. (2012). World citation and collaboration networks: uncovering the role of geography in science. *Scientific reports*, 2:902.

Panigrahi, A., Simhadri, H. V., and Bhattacharyya, C. (2019). Word2Sense: Sparse interpretable word embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5692–5705, Florence, Italy. Association for Computational Linguistics.

Parikh, N., Boyd, S., et al. (2014). Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239.

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710.

Pezeshkpour, P., Tian, Y., and Singh, S. (2019). Investigating robustness and interpretability of link prediction via adversarial modifications. *arXiv preprint arXiv:1905.00563*.

Porter, M. A., Mucha, P. J., Newman, M. E., and Warmbrand, C. M. (2005). A network analysis of committees in the us house of representatives. *Proceedings of the National Academy of Sciences*, 102(20):7057–7062.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

Riedel, S., Yao, L., McCallum, A., and Marlin, B. M. (2013). Relation extraction with matrix factorization and universal schemas. In *Proceedings of the 2013 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 74–84.

Ruffinelli, D., Broscheit, S., and Gemulla, R. (2019). You can teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*.

Schunn, C. D. (2020). The presence and absence of category knowledge in lsa. In *Proceedings of the Twenty First Annual Conference of the Cognitive Science Society*, pages 643–648. Psychology Press.

Sengupta, T., Pragadeesh, C., Talukdar, P. P., et al. (2017). Inducing interpretability in knowledge graph embeddings. *arXiv preprint arXiv:1712.03547*.

Shalev-Shwartz, S. et al. (2012). Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, 4(2):107–194.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9).

Sidiropoulos, N. D., De Lathauwer, L., Fu, X., Huang, K., Papalexakis, E. E., and Faloutsos, C. (2017). Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582.

Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

Socher, R., Chen, D., Manning, C. D., and Ng, A. (2013). Reasoning with neural tensor networks for knowledge base completion. *Advances in neural information processing systems*, 26.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Stanley, N., Bonacci, T., Kwitt, R., Niethammer, M., and Mucha, P. J. (2019). Stochastic block models with multiple continuous attributes. *Applied Network Science*, 4(1):1–22.

Subramanian, A., Pruthi, D., Jhamtani, H., Berg-Kirkpatrick, T., and Hovy, E. (2018). Spine: Sparse interpretable neural embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Sun, F., Guo, J., Lan, Y., Xu, J., and Cheng, X. (2016). Sparse word embeddings using l1 regularized online learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 2915–2921. AAAI Press.

Sun, Z., Deng, Z.-H., Nie, J.-Y., and Tang, J. (2019). Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*.

Toutanova, K., Chen, D., Pantel, P., Poon, H., Choudhury, P., and Gamon, M. (2015). Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Lisbon, Portugal. Association for Computational Linguistics.

Trouillon, T., Dance, C. R., Gaussier, É., Welbl, J., Riedel, S., and Bouchard, G. (2017). Knowledge graph completion via complex tensor factorization. *The Journal of Machine Learning Research*, 18(1):4735–4772.

Tucker, L. R. et al. (1964). The extension of factor analysis to three-dimensional matrices. *Contributions to mathematical psychology*, 110119.

van Engelen, J. E., Boekhout, H. D., and Takes, F. W. (2016). Explainable and efficient link prediction in real-world network data. In *International Symposium on Intelligent Data Analysis*, pages 295–307. Springer.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Voulodimos, A., Doulamis, N., Doulamis, A., and Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018.

Vrandečić, D. and Krötzsch, M. (2014). Wikidata: a free collaborative knowledge base.

Wang, Q., Mao, Z., Wang, B., and Guo, L. (2017). Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743.

Wang, Q., Wang, B., and Guo, L. (2015). Knowledge base completion using embeddings and rules. In *Twenty-fourth international joint conference on artificial intelligence*.

Wang, Y., Ruffinelli, D., Gemulla, R., Broscheit, S., and Meilicke, C. (2018). On evaluating embedding models for knowledge base completion. *arXiv preprint arXiv:1810.07180*.

Wang, Z., Zhang, J., Feng, J., and Chen, Z. (2014a). Knowledge graph and text jointly embedding. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1591–1601.

Wang, Z., Zhang, J., Feng, J., and Chen, Z. (2014b). Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.

Weisfeiler, B. and Lehman, A. A. (1968). A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia*, 2(9):12–16.

Weston, J., Bordes, A., Yakhnenko, O., and Usunier, N. (2013). Connecting language and knowledge bases with embedding models for relation extraction. *arXiv preprint arXiv:1307.7973*.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*.

Xiao, H., Huang, M., and Zhu, X. (2016). Ksr: A semantic representation of knowledge graph within a novel unsupervised paradigm. *arXiv preprint arXiv:1608.07685*.

Xiao, L. (2010). Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11(Oct):2543–2596.

Xie, Q., Ma, X., Dai, Z., and Hovy, E. (2017). An interpretable knowledge transfer model for knowledge base completion. *arXiv preprint arXiv:1704.05908*.

Yang, B., Yih, W.-t., He, X., Gao, J., and Deng, L. (2014). Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*.

Yang, X. and Wang, B. (2020). Local ranking and global fusion for personalized recommendation. *Applied Soft Computing*, 96:106636.

Ying, Z., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. (2019). Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32.

Zhang, M. and Chen, Y. (2017). Weisfeiler-lehman neural machine for link prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 575–583. ACM.

Zhang, M. and Chen, Y. (2018). Link prediction based on graph neural networks. In *Advances in Neural Information Processing Systems*, pages 5171–5181.

Zhao, J., Miao, L., Yang, J., Fang, H., Zhang, Q.-M., Nie, M., Holme, P., and Zhou, T. (2015). Prediction of links and weights in networks by reliable routes. *Scientific reports*, 5:12261.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81.

Declaration

I, Unai Zulaika Zurimendi, herewith declare that this dissertation is my own original work, carried out as a doctoral student at the University of Deusto. All assistance received and notions from other sources have been identified as such, acknowledging their correspondent contributions and citing them properly.

This work contains no material which has been presented in identical or similar form to any examination board, except where due acknowledgement is made in the dissertation.

This dissertation was finished writing on July 14$^{\text{th}}$, 2022.