A Closer Look At Neural Bellman-Ford Networks

Master Thesis

presented by
Simon Ernst Pressler
Matriculation Number 1449952

submitted to the
Data and Web Science Group
Prof. Dr. Rainer Gemulla
University of Mannheim

March 2023

Contents

List of Algorithms

List of Tables

Abstract

The Neural Bellman-Ford model (NBFNet) presented by Zhu et al. [43] has achieved impressive performance in link prediction in knowledge graphs. However, what drives this improvement over other approaches based on graph neural networks (GNNs) remains unclear. This thesis provides an independent replication of the reported performance metrics by creating an independent codebase in the JAX ecosystem. The reported results are reproduced, and the provided ablation study by the authors is extended by repeating the reported experiments on an additional dataset. The conducted ablation experiments support the reported findings in the initial publication. Additionally, the model is shown to be stable across different hyperparameter settings. Furthermore, experimental results indicate that the initialization of entity representations via an indicator function is critical for the strong performance of NBFNet. It is argued that the performance increase relates to the introduction of a direct dependence between the test subject and the predicate and between the test subject and the object. Neither alone is sufficient to produce the reported performance levels.

Chapter 1

Introduction

The last decades have seen an explosion in the accessibility of knowledge as the rise of the internet has made information and factual knowledge more readily available than ever. Therefore, organizing knowledge and making it accessible for automatic reasoning poses one of the central challenges for the further development of artificial intelligence.

While a lot of attention has been recently captured by huge language models trained on enormous corpora, they are currently not taking full advantage of the plethora of factual knowledge available, as they passively acquire knowledge while learning to reproduce language. In contrast to these models, knowledge graph-based approaches formalize facts by representing entities as embedded in complex relations to other entities. The result is a graph structure in which each contained object is related to other objects and factual information is stored in an intuitive and human-readable format. The focus on interconnected entities captures the human intuition that objects are not only defined by their characteristics but also by their relation to other objects. The resulting data structure describes each fact as a triple of a subject, a predicate, and an object. Together, these triples form a Knowledge Graph (KG)[1].

Alas, creating this highly structured way to store facts often involves substantive upfront costs, as knowledge has to be formalized and transformed into a set of entities and relations while ensuring the factual correctness of each fact. Furthermore, producing a complete knowledge graph poses a substantive problem as the number of potential relationships increases exponentially with the number of considered entities and relations connecting them. Therefore, augmenting incomplete knowledge graphs poses an important research problem.

This thesis focuses on the Neural Bellman-Ford network (NBFNet) model by

[1] For a glossary see in appendix Acronyms.

[43] that is proposed to solve the described link-prediction problem. Therefore, the first contribution of this thesis is the provision of an entirely independent reproduction of the research conducted by Zhu et al.. With this, it strengthens the results reported by the authors for link prediction in knowledge graphs, underlining the predictive strength of their proposed model. In doing so, this thesis provides the first implementation of the NBFNet model in the JAX ecosystem. It increases the set of possible framework choices for practitioners and researchers interested in the model and eases integration with existing JAX-based codebases.

Beyond reproducing the work by Zhu et al., additional insights into their proposed model are provided by re-training the model multiple times and computing the first estimates of the model variance. It is demonstrated that the model displays low variance comparable to most knowledge graph embedding methods. These additional statistics allow for a clearer distinction between meaningful experimental results and those which could be a product of model variance.

The fourth contribution is completing the ablation results provided by the authors. Their publication reports results on only one of two benchmark datasets. This opens the opportunity to provide the missing experiments for a fuller and more complete picture regarding model performance and the impact graph specific design choices. The conducted ablation experiments support the trends observed in the original publication.

Furthermore, additional ablation experiments have been conducted beyond those presented in the [43]. It is revealed that the skip connection added in the original publication provides no benefit for model performance. Furthermore, it is shown that query-dependent edge representations as part of the message-passing process are beneficial to model performance. In contrast, adding the predicate representation to the object representation during scoring provides no added value. It is demonstrated that the model is robust to reasonable changes in hyperparameters.

Last but not least, this thesis goes beyond the experiments conducted by the authors and provides a systematic investigation into the main drivers of the model's performance. To this end, the model is compared to alternative GNN-based approaches and subsequently adapted to study the influence of modelling decisions. It is demonstrated that using a multi-layer perceptron for scoring over a KGE scoring function is unnecessary to achieve the reported performance levels. The same is true for the use of message augmentation. However, the initialization of entity representations by an indicator function, creating strong dependencies between subject, predicate, and object representation, is demonstrated to be pivotal for model performance.

All necessary replication materials, including the codebase and the raw experimental data, can be found on the accompanying storage medium.

Chapter 2

Preliminaries

Before discussing the neural Bellman-Ford network model in more detail, this chapter is dedicated to laying the groundwork for this thesis. To this end, the following section 2.1 clarifies definitions of central terms and how they are used in this thesis. Section 2.2 provides a more formal definition of the link prediction problem to clarify the scope of the task the model under study should fulfil. Following section 2.3 discusses the related literature by first presenting relevant knowledge graph embedding models before moving on to present graph neural network-based approaches.

2.1 Terminology

Before moving on to the link prediction problem, several key terms need to be defined. First, the term ”knowledge graph” necessitates further clarification. Unfortunately, even though several different attempts have been made to define the concept, no agreed-upon standard definition exists. Nevertheless, different definitions frequently entail similar concepts. Usually, a knowledge graph is understood to contain facts codified as sets of nodes representing entities and edges which signify the relationship among them. Besides this minimal definition, some authors stress the nature of the encoded facts as covering multiple domains, while others extend it to include a temporal dimension or entity attributes [16].

For this thesis, it is sufficient to focus on the minimal definition of a knowledge graph. Therefore the term will be used to describe:

Definition 1 *A knowledge graph (KG) is a network composed of a set of nodes representing entities that are connected by edges signifying a form of relationship between these entities.*

This focus on the minimal definition of the term is motivated by the nature of the studied model, which understands KGs as a subset of (directed) edge-labelled graphs. This contrasts with more encompassing approaches, also paying attention to hierarchies of objects or defining additional knowledge graph schemes [16].

Given this understanding, a knowledge graph can be described as being composed of a set of facts $\mathcal{F}$, where an individual fact f_i is defined as:

Definition 2 *A fact is an ordered triple $[s_i, p_i, o_i]$, consisting of subject (s_i), a predicate (p_i), and object (o_i) . Where $s_i \in \mathcal{E}$, $o_i \in \mathcal{E}$, and $p_i \in \mathcal{R}$*

$\mathcal{E}$ signifies the set of all entities, while $\mathcal{R}$ denomiates the set of relations. From this definition, it is clear that when talking about an individual fact, s, p, and o can be used interchangeably with the terminology more typically found in the graph neural network literature describing an edge in a graph as composed of a head, relation, and tail. For a consistent use of terminology, this thesis will use the spo description whenever talking about a particular fact, which is being scored by the model. When discussing a general edge, or any other edge in the KG, the graph terminology will be used to clearly distinguish between the two cases. An overview table about the notation can be found in B.5 in appendix B.

2.2 Problem Statement and Metrics

After clarifying the terminology, this section proceeds to describe the problem under study in this thesis: link prediction in KGs. The link prediction problem in the specific context of KGs can be defined as:

Definition 3 *Given a KG $\mathcal{G}$, where $\mathcal{G}$ consists of n facts $\mathcal{G} = \{f_0, ..., f_n\} \in \mathcal{F}$, it needs to be decided if an unseen fact f_{n+1} from the set of all possible facts which can be constructed from $\mathcal{E}$ and $\mathcal{R}$ is valid given $\mathcal{G}$.*

In less formal terms: Link prediction is defined as accurately predicting the correctness of a new fact given the facts already contained in the KG. In case the statement is judged to be correct, it can be added to the KG. Therefore, the link prediction problem is usually discussed as a sub-problem of KG completion.

Common measures for the performance of models in link prediction tasks are based on ranking a set of test facts against all other possible facts in which either subject or object has been permuted. So for each fact in the test set $\mathcal{T}$, $2 * (|\mathcal{E}| - 1)$ permutations are scored. As the permutation of the subject or object can result in the creation of other existing facts, these should be excluded when assessing model performance. Subsequently, the predicted score of the true test fact can be ranked

against the filtered set of possible false facts. In case ties arise, different strategies are possible. This thesis follows the approach used in the NBFNet publication and resorts to a conservative tie resolution strategy which ranks the true fact worse than its permutations. The resulting ranking r serves as the basis for the computation of the following metrics:

$$\text{MRR} = \frac{1}{|\mathcal{T}|} \sum_{f \in \mathcal{T}} \frac{1}{r_f} \tag{2.1}$$

$$\text{MR} = \frac{1}{|\mathcal{T}|} \sum_{f \in \mathcal{T}} r_f \tag{2.2}$$

$$\text{Hits@}N = \frac{1}{|\mathcal{T}|} \sum_{f \in \mathcal{T}} \begin{cases} 1, & \text{if } r_f \leq N \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

Equation 2.1 presents the computation of the mean reciprocal rank (MRR), where $|\mathcal{T}|$ denotes the number of facts contained in the test set, and r_f the ranking of an individual fact. N is a constant. MRR serves as the main metric for comparison in the remainder of this thesis, as it provides a metric that is easily comparable across datasets. In contrast, mean rank (MR) is not scaled dataset independent and, therefore, less comparable but provides an absolute measure of predictive quality. The Hits@N metrics can be used to gain a more nuanced understanding of the model performance by providing the probability that the correct fact is among the top N answers produced by the model. Following the original publication, three thresholds are provided: Hits@1, Hits@3, and Hits@10.

Link prediction in KGs has been tackled using multiple different approaches. These include the inference of logical rules from a given set of facts and the subsequent application to fill gaps in the KG, e.g. [23]. While this kind of approach can use the facts contained in a KG as they are, alternative models applying machine learning algorithms run into an additional problem: *How can entire facts, individual entities and relations be translated into the euclidean space?* The resulting representations need to capture the topological structure of a KG meaningfully. This question, in the following, referred to as the translation problem, can be seen as the main challenge for these approaches. Therefore, the issue of link prediction is intimately connected to this translation problem. NBFNet provides a novel method for exactly this problem, by computing representations for entire facts, instead of representing s, p, and o separately, as will be discussed in chapter 3.

2.3 Related Work

As outlined in the last section, the main problem when applying machine learning techniques to the problem of graph completion lies in creating meaningful representations of entities and relations. Consequently, the proposed models for link prediction can be distinguished by their approach to the translation problem. The two families most important for this thesis are embedding-based approaches and graph neural network-based approaches, discussed in the following paragraphs.

2.3.1 Knowledge Graph Embedding Models

While many different embedding models exist, the fundamental idea is always to initialize embeddings for entities and relations, which are subsequently optimized to conform as well as possible to a scoring function. The scoring function enforces a specific assumed transformation relationship between s, p, and o in a given triple. The simplest example is the scoring function of the TransE model: $s + p \approx o$. The assumed interdependence between entities and relationships is a linear translation between s and o via p. Consequently, the distance between s and o after transforming s with p in the embedding space is minimized [4]. As this example demonstrates KGEs operate on learned entity representations and are therefore transductive, i.e. they can only be used to reason about entities that the model has seen during training.

Albeit many different forms of KGEs exist, they can be divided into two main approaches based on their definition of the scoring function. On the one hand, translation models use scoring functions that intend to project entities and relations into either one or multiple spaces, where their distance is minimized. Usually, the predicate is seen as a form of translation operation on the subject, the result of which should be as close as possible to the object representation [33].

For this thesis, two translational KGEs are of particular interest: The above-mentioned TransE model is one of the fundamental and oldest KGE models. Its scoring function minimizes the point-wise distance between s and o after s is translated by p. As a similarity measure, the authors deploy the L_1 or L_2 norm leading to the score function:

$$\|s + p - o\|_{1/2} \tag{2.4}$$

The simplicity of this function is one of the key advantages, as it makes TransE easy and fast to learn. Therefore, it is deployable on huge KGs. However, this relatively simple scoring function comes at the disadvantage of forcing entities that a common neighbour connects into relative proximity to each other in the vector space, as pointed out by [40, 33]. This lack of adaptability hinders the learning

of more complex relation structures. Therefore, TransE falls short of achieving state-of-the-art results [26].

Multiple models have been proposed based on TransE with the explicit aim of overcoming the outlined problems, among them the second embedding model directly relevant to this thesis. The RotatE model tries to solve the problems by considering the translation between subject and object as a rotation. More specifically, the authors provide a model which can accurately model symmetry, antisymmetry, inversion, and composition of relations. This is achieved by defining the entity and relation embeddings as vectors of complex numbers and defining the neighbourhoods of the relation embedding (p_i) as a rotation. $-\|s \circ p - o\|^2$ is the resulting scoring function where $|p_i| = 1$ and $\circ$ denotes the Hadamard product [29]. This approach can achieve state-of-the-art performance while remaining computationally affordable.

In contrast to translation models relying on distance-based scoring functions, the second approach to knowledge graph embeddings relevant to this thesis uses scoring functions that express similarities. This type of embedding model is described in the literature as semantic matching or tensor decomposition model [33, 40]. The oldest and most straightforward of these approaches exemplifies the underlying idea well: RESCAL uses the scoring function:

$$s^\intercal W_p t \tag{2.5}$$

W_p denotes a weight matrix dependent on the relation in a given triple. The authors use a bilinear mapping to capture threeway interaction effects between subject, predicate, and object [38].

DisMult, which is the third model interest for this thesis, can be understood as an evolution of RESCAL [33]. In contrast to RESCAL, DistMult simplifies the mapping by constraining the weight Matrix W_p to be a diagonal matrix [25]. This reduces the number of parameters substantively while limiting the captured interaction effects between subject and object to an element-wise product, thereby reducing time and space complexity to the same level as TransE but losing generality in the process [33].

As many more KGE models exist, a comprehensive discussion goes beyond the scope of this thesis. For a broader overview see [33, 40, 36]

2.3.2 Graph Neural Network Models

While KGE models are the dominant approach to obtaining algebraic representations of entities and relations, the fast development of graph neural network models has led to an increased research interest in how these can be adapted to the directed

and heterogeneous graphs that are KGs. As these models have been proven to capture graph structures in other domains, transferring them to KGs to compute representations that incorporate more of the local network topology surrounding entities has been an ongoing research effort. Underlying this trend is the intent to shift the representation learning away from primarily considering spo triples in isolation but rather to systematically considering local neighbourhoods of entities. This section explores this line of research by first briefly discussing general message-passing GNNs, before presenting a selected set of GNNs specifically designed for link prediction in KGs.

Message Passing GNN While the general field of GNNs is broader, the architectures applied to link prediction in KGs almost exclusively follow the same paradigm: Updating a central entity representation in a recurrent process by aggregating the representations of neighbouring entities dependent on the type of relation connecting them to the central entity. In other words, the entity representations are passed as messages along the edges of a given graph, and the central node obtains its updated representation based on the incoming messages. This iterative updating process incorporates information on the local network structure in a given L-hop neighbourhood into each node representation. L's size is the same as the number of message-passing iterations [39].

This model type can be described as a spatial graph convolution network (GCN) or message-passing graph neural network (MPGNN) [12]. While a consensus seems to exist that the former is a sub-group of the latter, the exact nature of this subsumption relation is not consistently defined, e.g. [41, 35]. The term GCN is used in the following paragraphs for consistency with the subsequently discussed literature, while the terms spatial graph convolution and message passing are used interchangeably.

A general description of this message passing process for a single vertex v is shown in algorithm 1 and adapted from [37]:

Algorithm 1

GENERAL MESSAGE-PASSING($h_v^{(0)}, \mathcal{N}_v$)

for $l \leftarrow 1$ to L **do**
  $a_v^{(l)} = \text{AGGREGATE}^{(l)}\left(\left\{h_u^{(l-1)} : u \in \mathcal{N}_v\right\}\right)$
  $h_v^{(l)} = \text{COMBINE}^{(l)}\left(h_v^{l-1}, a_v^{(l)}\right)$
end for

In this formulation, $\mathcal{N}_v$ denotes the set of neighbouring vertices to the vertex v for which a new representation h_v^l is computed during iteration l. The

AGGREGATE function in line two aggregates the representations of all neighbouring entities, while the COMBINE function in line three updates the entity representation based on the previous representation in step $l-1$.

For this thesis, it is helpful to conceptually split the AGGREGATE function into two stages:

Algorithm 2

MESSAGE FUNCTION AND AGGREGATION IN THE MESSAGE-PASSING ALGORYTHM($h_v^{(0)}, \mathcal{N}_v$)

1: **for** $l \leftarrow 1$ to L **do**
2: $\quad a_v^{(l)} = \text{Aggregation}^{(l)} \left(\text{Message}^{(l)} \left(\left\{ h_u^{(l-1)} : u \in \mathcal{N}_v \right\}, r_{u,v} \right) \right)$
3: $\quad h_v^{(l)} = \text{COMBINE}^{(l)} \left(h_v^{l-1}, a_v^{(l)} \right)$
4: **end for**

This formulation highlights that the AGGREGATE function contains two steps. First, the messages are calculated using a Message function before aggregating all incoming messages by an Aggregate function.

As GNNs are initially designed to be deployed on homogeneous and frequently undirected graphs, many models have been proposed to adapt them to heterogeneous and directed KGs. Beyond the commonality that all models resort to a form of message passing, the proposed models are diverse and numerous. This poses the problem that a comprehensive review covering all different approaches is clearly beyond the scope of this thesis. Therefore, the subsequently discussed models have been chosen because they share a central property with NBFNet. This allows for a helpful comparison among peers and avoids cluttering the comparison with only tangentially related models. Consequently, the discussion is limited to R-GCN, TransGCN, M-GNN, GRAIL, and INDIGO, as each model can highlight a specific aspect of NBFNet. For a more encompassing survey of GNNs for link prediction, see [39], which also informed the model selection for the following paragraphs.

R-GCN The first model proposed to tackle the problem of transferring a GNN to KGs is the R-GCN model by Schlichtkrull et al. [27]. The authors overcome the issues of graph heterogeneity and directionality by computing messages for each neighbour of a central entity. All entities are initialized with random representations. The messages are products between a relation and direction-dependent weight matrix $W_r^{(l)}$ and the entity representation of the respective neighbour. The new entity representation after one step of message passing $h_v^{(l+1)}$ can be described as:

$$h_v^{(l+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{j\in\mathcal{N}_i^r}\frac{1}{c_{i,r}}W_r^{(l)}h_j^{(l)} + W_0^{(l)}h_v^{(l)}\right) \tag{2.6}$$

While W_r signifies weight matrices dependent on the relation type r and direction. $W_0^{(l)}h_v^{(l)}$ adds a self-edge for each node to the network, thereby directly relating the new representation to the entity representation at $l-1$. This form of incorporating the prior entity representation into the updated entity representation is used by a number of the discussed models. Conceptually, the addition of self-edges can be understood as a special case of the COMBINE as presented in algorithm 1 and algorithm 2, in which the COMBINE function is equivalent with the Aggregation function. An architectural decision that is repeated across most of the subsequently discussed models.

The entity representations resulting from the iterative application of this message-passing scheme are subsequently scored using the DistMult scoring function. This general auto-encoder architecture of using the GNN as an encoder to obtain representations and subsequently scoring them with a separate decoder is still the dominant paradigm for GNN models designed for link prediction in KGs.

TransGCN R-GCN does not fully separate the encoder and decoder, as the relation representations are learned separately from the entity relations as part of the scoring function. Rather than learning the relation representations as part of the message passing, the model learns a set of relation- and direction-dependent weight matrices. This causes the model to increase substantively in parameter size and computational cost. In response to this problem, Cai et al. [7] propose the TransGCN model, which deviates from R-GCN by jointly learning entity and relation embeddings during the message passing process. To achieve this goal, they deviate from R-GCN in two ways: First, they replace the matrix mentioned above with the relation representation. A learned and layer-dependent transformation matrix updates this relation representation in each pass. Second, the relation representations are not directional, but rather the applied message computation function depends on the relation's direction. They provide two implementations of their model, one based on TransE and another on RotatE. Depending on the entity being head or tail, the original or the inverse of the message function is applied. This means, for the TransE model, that the message is either the sum or the difference between an entity and the connected relation, depending on the entity being head or tail. The RotatE models behave analogously. Again messages are aggregated by computing a weighted sum, and the resulting update is added with a message obtained from a self-edge. Depending on the model variations mentioned above,

the model is scored either using the TransE or the RotatE scoring function.

M-GNN Like TransGCN, M-GNN also improves upon R-GCN but chooses a different approach. The model proposed by Wang et al. [34] uses learned aggregate and message functions. The authors argue that the expressive capacity of R-GCN is not high enough, as, under the proposed updating scheme, two entities located in two non-isomorphic neighbourhoods in the KG can be assigned identical representations. This is related to using the weighted sum for aggregating the computed messages. The authors tackle this problem by computing new entity updates with the following function:

$$h_v^{(l)} = \text{MLP}^{(l)} \left(\left(1 + \epsilon^{(l)}\right) * h_{r_0,v}^{(l-1)} + \sum_{u \in \mathcal{N}_v^r} h_{r,u}^{l-1} \right) \tag{2.7}$$

Where, $\epsilon^{(k)}$ is a fixed constant and $h_{r,u}^{l-1} = W_r h_v$ with W_r again denoting a relation dependent weight matrix. $h_{r_0,v}^{(l-1)}$ denotes a self-edge. Beyond using learned functions, M-GNN can model hierarchical relations between entities by introducing graph and edge coarsening. The GNN is applied to multiple coarsened graphs. The resulting entity embeddings for high-level nodes resulting from combining entities during graph coarsening are then mapped back to these entities, and the representations are combined for scoring. As this aspect of the model is less relevant for the later comparison, please see the original publication for more details. The authors use DistMult and ComplEx for decoding.

GRAIL While the previously discussed models focus on fixing specific aspects of R-GCN, GRAIL [30] is concerned with the ability for inductive reasoning about unseen entities. In contrast to other discussed models, GRAIL extracts sub-graphs from the original KG, which contain only those entities which are part of direct paths from s to o. Furthermore, entity representations are not initialized randomly but rather encode the position of each entity to s and o in the respective sub-network. The fact being scored is added to the extracted sub-network. Computing all these sub-networks leads to a drastic increase in computational cost.

Following R-GCN, relation-specific weight matrices are used to compute messages that are combined as a weighted sum, where the weight is computed via attention. The resulting update is added to a self-edge.

Another idiosyncrasy of the model is the usage of not only the representations of s,p, and o to score a given fact but also of a representation of the fact's sub-graph. Like the previously discussed models, the concatenated representations are scored by a linear layer instead of a pre-defined scoring function.

INDIGO INDIGO by Liu et al. [22] also focuses on the ability of inductive reasoning about new entities. To this end, an entirely different approach is proposed: producing entity-pair representations instead of the individual entity and relation representations. This is achieved by not applying a GNN to a given KG directly but rather by encoding the KG into a new graph first. In this new graph, nodes are no longer entities but rather pairs of entities. An untyped and undirected edge connects these nodes if s or o are shared between them. Furthermore, each node is assigned a vector that encodes if a given relation, or its inverse, exists between s and o. E.g. $(0, 0, 1)$ expresses that only relation three connects the two entities in the KG. For message passing, INDIGO deploys a classic GCN [21], as the graph is non-longer directed or heterogeneous. Instead of a scoring function, a fact is judged for correctness by checking if the relevant so node's feature vector is above a threshold at the position corresponding to p.

Chapter 3

Neural Bellman-Ford Networks

After clarifying the basics in the last chapter, this chapter presents the NBFNet model in greater detail. This presentation will focus on the technical aspects of the model. The theoretical ideas underlying the model architecture will only briefly be covered. For further details on this issue, please see the NBFNet paper [43]. Subsequently, section 3.1.2 provides a detailed description of the model architecture. It highlights how the model architecture places NBFNet in the broader family of GNN-based models for link prediction by drawing comparisons to the GNN models described in section 2.3. This chapter concludes by considering questions regarding the implementation of the model in an independent codebase. This includes issues related to the chosen deep learning framework and deviations from the original implementation and the reasoning behind them.

3.1 The Model

Before discussing the details of the proposed mode architecture, it is necessary to understand the theoretical reasoning behind it. To this end, the following section briefly explains Zhu et al.'s reasoning underlying their design choices.

3.1.1 Theoretical Background

As Zhu et al. frame their theoretical considerations around the broader concept of link prediction not only in KGs, they adopt a graph-based terminology. The authors start their theoretical argument based on the assumption that link prediction necessitates the creation of a pair representation between s and o depending on p. Unfortunately, no further explanation is provided for this assumption, which differs from most other approaches as they compute separate representations for s and o,

as discussed in the literature section. Inspired by the inductive capabilities of path-based approaches to link prediction, the authors propose the computation of pair representations by capturing the sub-graph composed of all paths connecting s to o as previously discussed for the GRAIL model.

However, the authors point out that a simple implementation of this approach is, in practice, very costly, as all possible paths connecting s to o need to be computed. Therefore, the authors resort to the usage of the generalized Bellman-Ford algorithm. This algorithm is a generalization of the Bellman-Ford algorithm for the computation of single-source shortest paths in a weighted directed graph. The algorithm initializes the source node with 0 and all other nodes with $\inf$. Subsequently, the algorithm relaxes the edges in multiple iterations. More concretely, in each iteration, the values of a node are updated to be the minimum of the incoming messages from its neighbours. These messages are computed as the value of the neighbouring node plus the edge weight connecting both nodes. By repeatedly applying this algorithm until the node values no longer change, the shortest path between source and sink can be identified [8][chapter 24.1].

The generalized Bellman-Ford, algorithm generalizes this algorithm by replacing the addition operation with the generalized summation $\oplus$ and minimum with the generalized multiplication operator $\otimes$. The algorithm assumes that $\oplus$ and $\otimes$ form a semi-ring. Furthermore, they demonstrate that a large number of path-based algorithms can be understood as special cases of this approach, only applying different functions for the computation and the subsequent aggregation of the path representations. The authors present the following formulation of the Generalized Bellman-Ford two central functions in the algorithm as shown in equation 3.1 and 3.2. Please note that the notation has been adapted to the notation used in this thesis for consistency.

$$h_p^{(0)}(s,o) \leftarrow \mathbb{1}_p(s=o) \tag{3.1}$$

$$h_p^{(l)}(s,o) \leftarrow \left(\bigoplus_{(x,r,o)\in\mathcal{E}(o)} h_p^{l-1}(s,x) \oplus w_p(x,r,o) \right) \oplus h_p^{(0)}(s,o) \tag{3.2}$$

This approach closely resembles the general formulation of message-passing networks as described in equation 2, $h_q^{l-1}(s,x) \oplus w_q(x,r,o)$ is equivalent to the Message function, where $w_q(x,r,o)$ denotes and edge into o, and $h_q^{l-1}(s,x)$ the representation of the head entity at time $l-1$. $\bigoplus$ can be understood as analogous to the Aggregation function. As this formulation makes apparent, $h_q^{(0)}(s,o)$ is an addition introduced by the Generalized Bellman-Ford algorithm and has no

direct equivalent in the message passing formulation given in equation 2. However, the key difference is presented in equation 3.1, in which the indicator function presented in equation 3.1is used to initialize the representations of all nodes depending on whether they are s in a given query triple.

While the generalized Bellman-Ford algorithm provides a flexible way to compute representations of the sub-networks between u and v, it still relies on user-defined functions for $\oplus$, $\otimes$, and $\mathbb{1}$. Therefore these operators are replaced with learned neural functions, as shown below:

$$h_v^{(0)}(s, o) \leftarrow \text{INDICATOR}(s, o, p) \tag{3.3}$$

$$\begin{aligned} h_v^{(l)} \leftarrow & \text{ Aggregate} \\ & \left(\left\{ \text{Message} \left(h_x^{l-1}, w_p\left(x, r, o\right) \right) | (x, r, o) \in \mathcal{E}(o) \right\} \cup \left\{ h_o^0 \right\} \right) \end{aligned} \tag{3.4}$$

However, the parameterized and learned nature of these functions in their model provides no guarantee that the functions form a semi-ring as would be prescribed by the original Generalized Bellman-Ford algorithm.

As this thesis is focused on an empirical study of the model, the further theoretical details of the derivation process are not discussed in any more detail. Please, see the original publication for a full derivation and the associated proofs.

3.1.2 Model Architecture

The model resulting from this theoretical reasoning can best be explained in three steps: First, the preparation of the graph and the batches of training examples are described. Second, the message passing based on the Neural Bellman-Ford algorithm is discussed. Third, the scoring function and loss computation are presented.

Data Preparation As KGs only contain positive examples, i.e. existing relationships, negative examples need to be created. NBFNet relies on a negative sampling approach in which for each correct triple, n negative triples are created by perturbing s for half of a given training batch and o for the other half. These perturbations are random, but a filter ensures no valid triples are drawn. The filter removes true test-, validation-, and training triples. Therefore, the model operates under the closed world assumption, assuming that all drawn combinations not part of the original dataset are wrong.

Once the training examples are defined, the next step is to remove the positive examples from the graph to force the model to learn meaningful path representations between s and o. Additionally, the authors propose the removal of all one-hop

connections between s and o for each training example, including the negative examples. This step forces the model to learn longer path representations instead of focusing on direct one-hop connections. However, as is shown in table 4.2 in section 4.2, this procedure is only recommended for one of the two tested datasets. This approach's effect and the potential associated problems are discussed later in section 4.3, table 4.7 after introducing the datasets and their characteristics.

The remaining edges in the graph are augmented with the inverted graph edges, thereby effectively doubling the number of relations contained in the KG. This approach is equivalent to the way R-GCN deals with the directionality of relations, as the introduction of inverted relations allows for efficiently computing distinct representations of each relation depending on its direction.

Adding this set of inverted relations to the graph also allows for applying a computational trick, substantively reducing model run-time: The inversion of half of the training examples. As is explained in greater detail below, the representations of s are initialized dependent on p, whereas other non s entities are initialized differently. This means that for each s-p combination which is part of the training examples, the entire message-passing algorithm needs to be executed on a separate copy of the KG. As reported above, half of the negative training examples are produced by randomly perturbing s. In its naive form, this would force NBFNet to run the entire message-passing model over $b/2 * n + b$ individual graphs, where b is the batch size, and n is the number of negatives. However, this problem can be reduced to b independent graphs by inverting all training examples for which s has been perturbed to obtain negatives. As a result of this, substantive memory and computation cost is saved. However, NBFNet's computation of pair representations remains relatively expensive compared to more classic models such as TransGCN or R-GCN.

GNN Following the creation of a set of training examples and a batch-dependent version of the KG, the GNN model at the heart of NBFNet is applied. Before beginning the message-passing process, entity and relation representations need to be initialized. This step is the main divergence between NBFNet and other approaches. NBFNet relies on initializing random relation representations in an embedding layer from a normal distribution following the default variance scaling approach by TensorFlow.[1] Regarding entity representations, NBFNet distinguishes between the initial representation of s and the remaining non-s entities. Therefore the initialization of entity representations can be considered the application of an indicator function assigning all zero vectors to non-s entities and the previously

[1]For details, please see:
https://flax.readthedocs.io/en/latest/_modules/flax/linen/linear.html#Embed

initialized representation of p to s. The resulting initial entity representations are called the bounding condition and will be used multiple times in subsequent steps. The resulting relation representations will be referred to as general relation representations in contrast to the layer-specific relation representations discussed further below.

This step introduces two dependencies: First, the representation of s is directly dependent on p. Second, assigning a uniform all-zero vector to all non-s entities makes the representation of s the only source for meaningful updates in the graph. This approach is akin to the extraction of sub-networks proposed by GRAIL, where only edges that are part of paths connecting s to o are included in the message passing. In contrast to GRAIL, however, NBFNet's approach is less strict as other edges are not entirely removed. However, as the node representing s is the only source for a non-zero message, the impact of all edges not on a direct path between s and o is effectively minimized. Furthermore, the introduced dependency between the representation of s and the representation of o makes the representations of o effectively an s-o pair representation, an idea previously discussed as part of the INDIGO model even though INDIGO chooses an entirely different approach to obtain this form of representations. This highlights that ideas central to the architecture of NBFNet have been discussed previously, but the NBFNet architecture units them in a principled and consistent fashion.

After the initialization of representations, multiple iterations of message passing are applied to the KG. In the first step, layer-specific relation representations are generated. To this end, two approaches are discussed. On the one hand, layer-specific representations are created by learning independent relation embeddings for each layer. This approach follows R-GCN and M-GNN in as much as the edge representations during message passing are detached from relation representations later used for scoring. However, in contrast to both approaches, NBFNet learns only vectors instead of more complex transformation matrices, which reduces memory consumption.

Alternatively, an approach is discussed which creates layer-specific representations by applying a linear layer to the general representation of p. This approach is similar to the strategy deployed by TransGCN in that relation representations are generated from a given input and not created as random embeddings. However, this approach divers, as the relation representations at layer l in TransGCN are based on their representation in layer $l - 1$. In contrast, the representations in NBFNet are always depending on the representation of p at $i = 0$. This approach is subsequently referred to as query-dependent edge representations.

Given the layer-specific relation representations, the messages still need to be computed. To this end, the authors explore three different algorithms based on popular KGE models: DistMult, TransE, and RotatE. The authors adapt the scor-

ing functions of these models to fit a setting in which only head and edge are given. For example, the proposed DistMult message passing function computes the elementwise product of head and edge. Similarly, TransE is reduced to adding the head and edge representation. As the original RotatE also expects representations composed of complex numbers, the first half of the representation vector is treated as the real, and the other half as the imaginary part of the numbers. The pseudo-code for these message computation functions is presented in algorithm 3 where r is the type of the edge and h_u is the representation of the head of the edge.

Algorithm 3

ROTATE AS MESSAGE FUNCTION(h_u, r)

$h_{u,\text{real}}, h_{u,\text{img}} = \text{split}(h_u)$
$r_{u,\text{real}}, r_{\text{img}} = \text{split}(r)$
$\text{Message}_{\text{real}} = h_{u,\text{real}} * r_{\text{real}} - h_{u,\text{img}} * r_{\text{img}}$
$\text{Message}_{\text{img}} = h_{u,\text{real}} * r_{\text{img}} + h_{u,\text{img}} * r_{\text{real}}$
return $\text{concatenate}(\text{Message}_{\text{real}}, \text{Message}_{\text{img}})$

Also, this approach to message computation is not unprecedented. TransGCN also explores the usage of TransE and RotatE-derived message functions. However, while TransGCN proposes to adapt the message function depending on the direction of a connecting edge, NBFNet has already accounted for directionality by introducing inverted edges into the graph.

Following the message computation, the resulting set of messages is augmented with the bounding condition. This process is motivated by the theoretical formulation of the model, as outlined in the last section. It re-introduces the bounding condition and thereby reinforces the effect of the indicator function, making s the only source of a meaningful signal in the graph. Furthermore, it is noteworthy that message augmentation by using the bounding condition instead of the entity representations at $l-1$ introduces the same input across all layers of message passing. Message augmentation, as strictly motivated by the Generalized Bellman-Ford algorithm, is unique to NBFNet.

Given a set of messages for each entity in the KG, an update is computed by aggregating these messages. Four different aggregation functions are explored: Mean, Max, Sum, and Principle Neighbourhood Aggregation (PNA) as proposed by [9]. While the former three functions are self-explanatory, the latter merits a more detailed discussion, especially as it consistently outperforms the alternatives. The pseudo-code is provided in algorithm 4, where M denotes a vector containing all messages, O represents a vector holding index of the receiving node associated with each message, and D denominates a vector containing the degree of each

node. $\times$ denotes the Kronecker product. All functions with the prefix "Scatter", are computed separately for each message-receiving node.

Algorithm 4

PRINCIPEL NEIGHBOURHOOD AGGREGATION AS MESSAGE AGGREGATION FUNCTION($\mathcal{M}$,O, D)

MeanDegrees $=$ mean(D)
MeanMessages $=$ ScatterMean(M, O) $\triangleright$ *m*ean for each unique value in o
MeanSqrtMessages $=$ ScatterMean(M^2, O)
MinMessages $=$ ScatterMinimum(M, O)
MaxMessages $=$ ScatterMaximum(M, O)
STD $= \sqrt{\text{Clamp}\left(\text{MeanSqrtMessages} - \text{MeanMessages}^2, \text{min} = 1e-6\right)}$
Features $=$ Concatenate(MeanDegrees, MaxMessages, MinMessages, STD)
SCALE $= \log(D)/\text{mean}(\log(D))$
SCALES $=$ Concatenate$(\mathbf{1}, \text{scale}, \mathbf{1}/\text{clamp}(\text{scale}, \text{min} = 0.01))$
return Features $\times$ Scales

First, the number of neighbours is computed for each entity. It is worth noting that it is done for all entities, regardless if they only appear in test- or validation but not in training. Therefore, using PNA makes the model less inductive. It assumes that all entities part of the graph are known in advance, even though knowledge of their connection to the remainder of the KG is unnecessary. To account for the message augmentation, 1 is added to the count of neighbours for each entity. The resulting degree vector is used as the basis for the computation of a scaling vector. Next, for each entity, maximum, minimum, mean, and standard deviation are computed across its messages. These features are concatenated and subsequently multiplied in a Kronecker product with the scaling factor, leading to an output of size $12 * d$, where d is the dimensionality of the representations. Given the increase of output size by factor 12, this aggregation function substantively increases the memory footprint of the model if used.

While other models, such as R-GCN or Trans-GCN other GNNs, use weighted sums or mean aggregation, NBFNet uses a learned aggregation function like MGNN. To this end, the pre-aggregated messages are concatenated with the previous entity representations at $l - 1$, and subsequently, a linear layer is applied. This way for NBFNet to incorporate the entity representations at $t - 1$ into the computation of new representations at time t is also unusual. Most other GNN-based approaches utilize self-edges to incorporate the prior entity representation into the computation of the updated one, e.g. [27]. Using a linear layer to combine update and previous representations allows the model to learn which parts of the prior representation

should be updated in each layer. In contrast, using self-edges adds a message based on the old state to the set of messages. As this set is unordered, the aggregation function needs to be commutative, which does not allow it to distinguish between any incoming message and the prior entity representation. Therefore it cannot up or down-weight the previous representation against the other messages.

The resulting vector is normalized by applying layer normalization, and a rectified linear function (ReLU) is added to introduce non-linearity. The resulting new entity representations are added to the old node representations. With this, the authors introduce a skip-connection to ease model training, given the application of multiple message-passing layers.

The resulting entity representations serve as the basis for the next message-passing layer.

Scoring and Loss After multiple message passing layers, the final entity representations are computed. The next step is scoring the training examples. While other GNNs, such as R-GCN or TransGCN, need a full spo triple to score a given fact, NBFNet only scores a fact based on the representation of o concatenated with the representation of p. This is possible as NBFNet learns so pair representations. All necessary information regarding the pair of s and o should be contained in the representation of o. Regarding the concatenation of p to the representation of o, it can be argued that it shouldn't be strictly necessary to add further information as the initial representation of s, which is subsequently passed through the network and re-introduced to the network during each message passing iteration, is equivalent to the representation of the query relation p. This scoring based on the concatenated po vector is possible as NBFNet does not rely on traditional KGE scoring functions, like most other GNNs for link prediction. Rather than going down this conventional path, it follows the example of GRAIL and uses an MLP for decoding. The scoring is done via a 2-layer perceptron using the ReLU function as non-linearity.

After scoring the training examples, the loss is computed as binary cross-entropy loss (BCE). $P(f_{\text{postive}})$ denotes the probability assigned to the true training fact, while $P(f_i)$ denotes the probability associated with one of n negative examples.

$$\mathcal{L} = -\log P(f_{\text{postive}}) - \sum_{i=1}^{n} \frac{1}{n} \log(1 - P(f_i)) \tag{3.5}$$

Not all randomly sampled negatives are equally valuable during training, as some are easy cases. To combat this problem, the contribution of the negative

examples to the loss is scaled by using self-adversarial negative sampling as outlined in [29]. The scaling factor is obtained by applying a softmax function over the predicted logits of the negative examples and further divided by an adversarial temperature hyperparameter. This increases the importance of the negative examples, which are associated with high logit scores. Please note that, given a temperature of zero, all negative examples are weighted equally. The scaling factor becomes $1/n$ is the set of negative examples per positive. This is equivalent to not using self-adversarial sampling. The resulting loss is used to optimize the model parameters deploying the Adam optimizer [20]. The model is trained for 20 epochs, and early stopping is applied based on the performance on the validation data. During validation and test, the model follows the same procedure as outlined above. Message-passing is again conducted on a KG, containing only the training facts. However, one-hops are not removed during test and validation.

In summary, many of the architectural details of NBFNet are not new, as NBFNet follows a very classic GNN for link prediction architecture. However, the model deviates from other approaches by the uniform initialization of non s entities in the KG and by initializing s with the representation of p; this approach enforces the learning of entity pair representations in a novel way. Furthermore, the message augmentation continuously re-enforces the effect of the indicator function-based initialization of entity representations. Last, using an MLP as a decoder, while not entirely new, is unusual. All of the theses aspects are subject to experimentation in section 4.4.

3.2 The Implementation

After this rather technical model description, this section focuses on implementation-related questions starting from a discussion of the deployed deep learning framework before discussing the implementation differences between the replication codebase and the original. Zhu et al. provide two implementations of their model. The official implementation is based on TorchDrug [42], a PyTorch-based framework explicitly designed for drug discovery and developed by the Mila - Quebec AI Institute with which the authors are associated. Besides drug discovery, TorchDrug offers inbuilt capabilities for link prediction in KGs. Therefore, parts of the model, such as the scaling of the contribution of the negatives to the BCE loss by self-adversarial sampling, can be found in TorchDrug rather than in the provided replication repository. As this implementation is the official codebase, it was a reference point for my replication. An alternative re-implementation in Pytorch Geometric exists but has not been used.

3.2.1 Framework Choice

One of the stated goals of this thesis is to replicate the results reported in the original NBFNet publication. Therefore the choice of the framework has been primarily driven by considerations regarding the largest possible independence from the original codebase to strengthen the validity of the replicated results. Therefore, an implementation in TorchDrug or PyG would not have provided any added value. Given the current landscape of graph learning frameworks, the number of potential choices in Python is limited. While Microsoft's PyTorch GNN library lacked sufficient documentation, the TensorFlow-based Graph Nets library by DeepMind is, at the time of writing, in the process of being super-seeded by JRAPH. I abstained from using Spektral, given its implementation based on Keras, which is associated with a loss in computation speed. This left me with two potential choices: The Deep Graph Library (DGL) [32], or JRAPH by DeepMind [13].

While DGL is a well-known library, Jraph is a relatively new library building on JAX [5]. JAX itself can be seen as a direct competitor to PyTorch and TensorFlow, as it provides accelerated linear algebra compilation (XLA) combined with automatic gradient computation for arbitrary functions. JAX itself offers a high-level API mirroring NumPy. In contrast to PyTorch, it supplies mainly the basic NumPy operations. Functionalities such as neural-network layers are provided separately in libraries such as Haiku and FLAX, while different optimizers and loss functions are part of the Optax library. Therefore, the remainder of this thesis will refer to the JAX ecosystem rather than to JAX individually when talking about JAX. Jraph is developed as part of this ecosystem, adding graph-specific capabilities.

I decided to utilize JRAPH as it should provide two critical advantages over DGL: First, by using an entirely different framework not based on PyTorch, I achieved maximal independence between the original implementation and my codebase, thereby strengthening my replication. Second, HuggingFace reports substantive speedups for language models when comparing PyTorch-based implementations with JAX-based ones [19]. I could not locate benchmarks more specific to the graph domain, but I took the reported performance gains to indicate a general improvement in training speed. This would also align with the reported self-description of JAX provided by DeepMind. Therefore, I expected to achieve a decrease in training time, potentially opening up the possibility for more experimentation later on. The resulting stack of libraries included JAX, Flax for pre-defined neural network layers [15], and Optax [1] for loss computation and optimisation.

3.2.2 Differences to the Original

This framework choice also came with challenges that needed to be overcome. While some merely involved changes in coding style, others necessitated deviations from the original model. This section discusses the challenges experienced during the implementation and their influence on the resulting replication codebase.

Functional Programming The first set of challenges relates to characteristics of JAX[2]. JAX, as a framework, is designed following a purely functional programming paradigm. Consequently, it expects only pure functions as inputs and provides no guarantees for the correct behaviour of side effects after just-in-time compilation (JIT). Thereby, JAX forces its users to also work in the functional programming paradigm whenever their code interacts with JAX. Consequently, the provided codebase also follows this paradigm.

Padding JIT compilation is also at the heart of the second complication of using JAX. Functions are cached by the JIT compiler based on the type and shape of their inputs. Given a dynamically shaped input, the function is recompiled every time a previously unencountered shape is supplied, resulting in significant performance overhead. Furthermore, intermediate arrays during the computation are also expected to be statically sized, a requirement strictly enforced by the JIT compiler. While it is possible to declare the input on which intermediate array shapes are dependent as static, this causes re-compilation for each different value of the static input. Again, this can only be a solution given a small set of possible values; otherwise, the performance hit becomes substantial. Furthermore, this also means that intermediate arrays cannot be shaped on the fly to conform in their shape to a given input array, as is easily doable in PyTorch, e.g. `torch.ones_like`. This problem has been overcome by precomputing shapes as part of a generator function returning a partial function in which the shapes are fixed.

Regarding dynamically shaped input arrays, another solution has been applied: padding. The number of facts in the graph is dynamically changing based on the batch currently processed due to the removal of the training examples from the graph. Therefore, the graph must be padded to conform to its initial shape. This poses the problem that the added entity and relation must not assert any influence on the parameter optimization. Otherwise, the replication codebase would no longer be directly comparable to the original. For this purpose, a virtual entity

[2]For a complete list of potentially surprising behaviours, see
https://jax.readthedocs.io/en/latest/notebooks/Common_Gotchas_in_JAX.html

and relation are added to the graph. The graph is padded with facts constructed as self-relations of this virtual entity with itself, where the relation is the added virtual relation. The padded entities are excluded from message passing. Consequently, the added entity and relation assert no influence on the final representations of o and therefore have no impact on the computed loss. Due to this lack of effect on the loss, they do not influence the mode learning process.

Message Passing Algorithm and Checkpointing Besides the openly advertised complications associated with a JAX-based approach, I encountered another problem when implementing the message-passing algorithm. The initial implementation was based directly on the message passing in JRAPH. This worked well on the toy datasets used for development, but when applying it to the full WN18RR datasets, the memory consumption of this implementation became problematic. Even though the model was deployed on an NVIDIA A6000 GPU with 48 gigabytes of memory, the memory was not sufficient to compute gradients. Consequently, I implemented a second version of the message-passing algorithm. This second attempt was based on one of the two implementations for message passing provided in the official materials of NBFNet and reduced memory consumption slightly.

However, the original codebase mainly utilises a different approach to message passing which solves this memory issue by implementing the algorithm outlined in [17] in CUDA as part of the TorchDrug framework. This approach avoids the explicit computation of messages by unifying message computation and aggregation in a highly optimised fashion. Unfortunately, I could not locate an implementation of this algorithm in JAX, and an implementation from scratch to the same performance level was beyond this thesis's scope.

At last, this issue could be resolved by extensively limiting checkpointing. As described in the official documentation, all inputs to functions are stored during the differentiation of a function. In this context, this meant storing the number of iterations times the batch size times the number of edges times the representation dimensionality data points. This massively increased memory consumption. By turning off checkpointing, the model was forced not to store all the inputs but re-compute them during the backward pass. While this solution reduces memory cost, it is traded against computation speed. Fortunately, the overall increased computation speed of JAX compared to PyTorch led to comparable training times of ≈ 45 minutes per epoch on WN18RR, while increasing training time on FB15k-237 from ≈ 80 minutes to ≈ 100 minutes per epoch.

Data Parallelism The last issue which needs to be mentioned is the problem of data parallelism in the original model. The authors report training on four GPUs by splitting the batch into four equally sized sub-batches. They utilise the PyTorch distributed data-parallel functionality, which computes gradients for each part of the batch individually and subsequently back-propagates based on the average of these gradients. The updated model is redistributed across all GPUs [11].

Conforming to this approach was impossible, as the number of available GPUs with sufficient memory was limited to eight in a shared student server. As the resources of this server are highly contested, reliably accessing four GPUs was not possible. Using the entire batch instead of dividing it by four was also not an option due to the above-outlined memory constraints.

Consequently, I decided to emulate a data-parallel approach by breaking up the batch into four mini-batches and computing the gradients for each mini-batch sequentially on one GPU. As in the original, the resulting gradients are averaged to obtain the gradients for back-propagation.

Chapter 4

Replication and Experimentation

As stated during the introduction, the contributions of this thesis can be summarised under two over-arching goals: The first goal is the provision of an independent reproduction of NBFNet's performance metrics for link prediction, as reported in the original publication. The second goal is the provision of additional insights regarding the mechanisms driving model performance. Building on the above-described model architecture and implementation, this section tackles these two main goals by reporting the reproduction results in section 4.2 before studying how different architectural changes and varying hyperparameters affect model performance in section 4.3. The following section 4.4 goes beyond this experimentation to explore the differences setting NBFNet apart from other GNN-based approaches to link prediction. However, before discussing the results, the following section 4.1 needs to establish basic information about the used datasets.

4.1 Datasets

Before discussing concrete results, this section provides information about the benchmarking datasets. The original publication provides results on two classic link-prediction in KG benchmarks: WN18RR and FB15k-237. Table 4.1 presents essential characteristics of both datasets.

FB15k-237 The FB15k dataset builds upon the Freebase ([2]) general knowledge graph, which contains general facts about various topics, including but not limited to sports, movies, nationalities, and many more. The original FB15k dataset was published alongside the TransE model in 2015 [4] and has been constructed as a subset of Freebase to include entities that participated in more than 100 relations and were also mentioned in the Wikilinks database([28]).

	WN18RR	FB15k-237
Number of entities	40943	14541
Number of relations	11	237
Number of training examples	86835	272115
Number of validation examples	3034	17535
Number of test examples	3134	20466
Domain	semantic relations	general knowledge

Table 4.1: Essential dataset statistics for WN18RR and FB15k-237 as found in [6].

However, Toutanova and Chen [31] noticed that in FB15k, the test data is oftentimes a simple inversion of relations directly contained in the training data. This trivialises large parts of the validation, and test data as the model only needs to learn to invert a given query triple. To overcome this problem, they published the FB15k-237 dataset, in which they removed inverse relations from the dataset. Furthermore, any triples were removed from the validation and training dataset, which were one-hop distant from each other in the training set [24]. As demonstrated by [18], in all validation facts, s and o are either two or three hops distant from each other. This pattern is inconsistent with the training data in which one-hop connections exist even when removing the training batch from the graph.

This absence of one-hop relations in validation and training causes Meilicke et al. [24] to note that the learning of rules relying on direct neighbourhoods during training can only decrease model performance in validation and testing. Meilicke et al. describe this as an unrealistic bias. This thesis agrees with this assessment and empirically demonstrates the problem in section 4.3 of this thesis when discussing the effect of one-hop removal during training on overall model performance.

WN18RR WN18RR traces its roots back to the WordNet knowledge base [3]. The WordNet KG has been designed as a thesaurus. Therefore, it contains relationships between words, such as homonyms or hypernyms [4]. The publication that problematised the leakage by inverse relations in FB15k also uncovered the same issue in WN18RR [31]. However, Toutanova and Chen failed to provide an updated version of this dataset to combat this problem. This task was left to [10], who presented WN18RR as an updated version without this problem. The new dataset also displays a more varied distribution of shortest path lengths between s and o as 90% no longer are only removed. While a one-hop distance is still the most common case, distances up to 10 hops are observed [18].

	WN18RR	FB15k-237
Number of Message Passing Layers	6	6
Message Function	DistMult	DistMult
Aggregation Function	PNA	PNA
Learning Rate	5e-3	5e-3
Representation Dimensionality	32	32
Negative Examples per Positive	32	32
Batch size	128	256†
Remove One-Hops from Graph	False	True
Query Dependent Edge Representations	False	True
Adversarial Temperature	1	0.5

Table 4.2: The hyperparameter configurations reported by [43] for replicating their performance estimtaes on WN18RR, and FB15k-237. †Please note that a batch size of 256 should result in four mini-batches with 64 examples each. Due to memory constraints, the model has been reproduced using eight mini-batches with 32 examples each.

4.2 Replication

As stated above, this thesis's first goal is to provide an independent reproduction of the performance estimates for link prediction on KGs published alongside the NBFNet model. Given equivalent implementations, the same set of hyperparameters should be able to reproduce the original results closely. The original hyperparameters are reported in table 4.2. Alas, using the exact same set of hyperparameters on the FB15k-237 dataset was not entirely possible. As stated in section 3.2.2, the authors used four GPUs in a data-parallel fashion. Therefore, the reported batch size of 256 should result in four sub-batches of 64 examples distributed over the GPUs. However, given the larger number of training examples in FB15k-237, mini-batches of size 64 would have exceeded the memory of the available GPUs for reproduction ($>$ 48 GB). Therefore, it was necessary to divide each batch into eight mini-batches of size 32. While theoretically equivalent, minor differences resulting from this approach cannot be ruled out entirely.

The reproduction of the performance metrics provides the first opportunity to extend the original publication by adding estimates of the variance when re-training the model multiple times. To this end, each model has been trained three times, using the same seed and hyper-parameters to isolate only the variance caused by non-deterministic factors inherent in the training procedure. Table 4.3 presents the reproduced performance metrics as the mean of three training runs. The re-

		Original	Replication			
			Metric	Std	Lower	Upper
WN18RR	MRR	0.551	0.553	0.001	0.552	0.556
	MR	636	629	1.03	627	631
	H@1	0.497	0.499	0.001	0.496	0.502
	H@3	0.573	0.578	0.001	0.576	0.580
	H@10	0.666	0.663	0.000	0.662	0.664
FB15k-237	MRR	0.415	0.413	0.002	0.401	0.417
	MR	144	111	2.24	108	117
	H@1	0.321	0.320	0.003	0.315	0.325
	H@3	0.415	0.453	0.002	0.450	0.457
	H@10	0.599	0.596	0.002	0.450	0.457

Table 4.3: The replication results on both datasets using the hyperaparamters as show in tale 4.2. Reported replication results are the average of three training runs, based on which the standard deviations (Std) have been computed. Lower and Upper signify the lower and upper bounds of 95% confidence intervals based on the standard deviation. Original results taken from table 3 in [43].

implemented model can reproduce the reported results on both datasets. While small differences between the implementations exist, these are likely to reflect differences in parameter initialization or random number generation between PyTorch and Jax. Furthermore, these differences can be due to the non-deterministic nature of XLA when deployed on GPUs. Therefore, even a direct copy of the model parameters after training from one implementation to another is highly unlikely to produce exactly the same outputs.[1]

Going beyond the point estimates for model performance, table 4.3 also shows the standard deviation (std) as well as 95% confidence intervals of the reported metrics. The confidence intervals have been computed assuming the observed variance follows a normal distribution converging under the true performance metric. Generally, table 4.3 demonstrates only a small model variance when re-training the model. Even the relatively few training runs lead to small standard deviations comparable to the variation described in [26] for KGE models.[2]

[1] For a discussion of this issue, see https://github.com/google/flax/issues/963

[2] Please note that [26] report variance estimates on validation, while this table reports variance on the test data. Given the observed close resemblance between the validation and test results of NBFNet, this should not make a significant difference.

4.3 Ablation

After successfully reproducing the original results, this thesis's second goal is to provide additional insights into the model and its performance characteristics. A natural starting point for this investigation is the further ablation of the model beyond the experiments conducted in the original publication. This section not only provides the missing ablation on WN18RR but also demonstrates the effect of specific architectural choices as well as the general hyperparameter sensitivity of the model.

The subsequently discussed ablation experiments are all conducted by fully retraining the replication model. All parameters not explicitly subject to experimentation are fixed to the reproduction parameters as reported in table 4.3.

It can be argued that the changes presented below are consequential alterations to the model architecture. Consequently, an ideal approach would have been to individually conduct a full hyperparameter search for each tested setting. This approach could have strengthened the overall confidence that the presented differences are meaningfully and consistently related to architectural decisions instead of being the product of hyperparameters not optimally selected for the now-adapted optimization problem. Furthermore, this approach would have allowed for broader claims regarding NBFNet as a model family rather than being limited to statements about the two specific replication models. While this approach would have provided the strongest causal isolation, full searches for all the presented ablations and experiments would have been prohibitively costly. They would have by far exceeded the resources available to this thesis. Furthermore, the original publication does not indicate that the presented experiments were constructed using a search or multiple training runs. This makes the original ablation results directly comparable to the results reported below. In summary, this ablation provides insights regarding the specific NBFNet instances that are the replication models rather than the whole model family. A broader investigation needs to be left for future research.

Within this limitation, this approach allows for a direct comparison between different experimental settings on the same dataset. However, the comparison of effects across datasets is more complicated. It cannot be conclusively ruled out that potential interactions exist between those hyperparameters subject to experimentation and those already different across the datasets. While it cannot be ruled out that these interaction effects might influence model performance, the subsequently presented experiments show a generally low hyperparameter sensitivity. Even though this indicates that this problem, therefore, is likely of lesser impact, this caveat must be considered when discussing differences across the two datasets.

Please note that whenever the replication settings are reported alongside experimental results, the reported MRR is presented at the mean of the three retraining

processes reported in table 4.3 for consistency reasons.

A good starting point for further ablation is the extension of the experiments conducted in the initial publication. These were, unfortunately, limited to the FB15k-237 dataset. Given the differences in dataset characteristics as mentioned in section 4.1, the robustness of the drawn conclusions can be strengthened by replicating the conducted experiments on the second available dataset. Therefore, the experiments regarding the influence of path length, i.e. the number of message-passing iterations, and the effect of different aggregation and message-passing functions are reconsidered on the WN18RR dataset.

Message and Aggregation Function As outlined in section 3.1.1, the NBFNet model can be described by three neural functions: The message function, the aggregation function, and the Indicator function. While the number of possible alterations to the indicator function is relatively limited, a wide variety of choices exist for the message passing and aggregation function. To shed light on the effect these two functions have on model performance, the original publication provides performance metrics exploring 12 different combinations of message and aggregation functions on FB15k-237. The results of this effort are shown in table 4.4a. For better interpretability and to ease a later comparison to results on WN18RR, table 4.4c is provided in which the results are scaled to the performance of the model based on the replication hyperparameters.

Zhu et al. put forward the interpretation that more complex message and aggregation functions are associated with improvements in model performance. Furthermore, the comparatively stronger performance of two combinations, given their relative simplicity, is highlighted: The TransE message passing function in combination with the maximum as aggregation function and the DistMult function for message passing in combination with using summation for aggregation. Given these message-passing functions, no other simple aggregation function performs equally well in both cases. This is explained by the fact that these combinations fulfil the semi-ring assumption of the generalized Bellman-Ford algorithm.

This thesis adds to these results by conducting the same experiments on WN18RR using the replication model. The results and the results scaled to the model using the official reproduction setting can be seen in table 4.4b and table 4.4d. Table B.1 provides in appendix B provides additional metrics. The reported results are compatible with the interpretations put forward in the original publication. Again more complex aggregation and message functions seem to improve model performance, while the two simpler combinations discussed above outperform alternative combinations with equally simple aggregation functions. However, the differences between the combinations tend to be less pronounced. TransE still performs gen-

Message	Aggregate			
	Sum	Mean	Max	PNA
TransE	0.297	0.310	0.377	0.383
DistMult	0.388	0.384	0.374	0.415
RotatE	0.392	0.376	0.385	0.414

(a) MRR on FB15k-237

Message	Aggregate			
	Sum	Mean	Max	PNA
TransE	0.439†	0.414	0.520	0.536
DistMult	0.538	0.528	0.535	0.553
RotatE	0.534	0.531	0.539	0.549

(b) MRR on WN18RR

Message	Aggregate			
	Sum	Mean	Max	PNA
TransE	71.56%	74.70%	90.84%	92.28%
DistMult	93.49%	92.53%	90.12%	100%
RotatE	94.46%	90.60%	92.77%	99.76%

(c) MRR on FB15k-237 as % of the replication model

Message	Aggregate			
	Sum	Mean	Max	PNA
TransE	77.40%	74.86%	94.03%	96.92%
DistMult	97.29%	95.48%	96.75%	100%
RotatE	96.56%	96.02%	97.47%	99.27%

(d) MRR on WN18RR as % of the replication model

Table 4.4: Comparison of the effect of different combinations of message and aggregation functions on MRR. The results provided in sub-table 4.4a are originally presented in table 6 a in [43]. The results in table 4.4b are produced with the reproduction codebase. All other hyper-parameters are fixed at their reproduction values as displayed in 4.2. Table 4.4c and table 4.4d report the results scaled by results obtained from the reproduction parameters.

†Please note that this experiment has been accidentally conducted twice. The reported result is the mean of both results. Interestingly the variance between both runs is considerably larger than the variance reported for the replication setting. This could indicate larger variance for weaker models. However, this is not systematically explored.

erally worse than the two alternatively explored message functions. Differences between RotatE and DistMult are non-consistently favouring one message function over the other. This is an interesting finding as the RotatE model is generally more expressive than the DistMult model as outlined in chapter 2.3. While this could be due to the adaptation of the RotatE scoring function to work only based on relation and head entity for message passing, this could also point towards a limit of the usefulness of further increasing the expressive capacity of the message passing function beyond a certain threshold after which the capacity of the aggregation function dominates overall model performance. The observation that PNA consistently outperforms the simpler aggregation functions on both datasets can be theoretically explained by considering the findings presented in [37]. The authors demonstrate that all of the simpler functions used for aggregation are individually unable to represent specific graph topologies uniquely, i.e. they are not injective. PNA combines multiple simpler aggregation functions by concatenation. This increases the expressive power of PNA over simpler alternatives and makes more information accessible to the linear layer combining the update with the previous entity representation.

Maximum Path Length The second set of ablation experiments conducted in the original publication considers the effect of the number of iterations of message passing, i.e. the maximum path length which can be represented by the model. The results of these experiments are shown in table 4.5; again, results on FB15k-237 are taken from the original publication, while the results based on WN18RR are based on the reproduced model. For additional metrics please see table B.2 in appendix B

The emerging pattern is identical between both datasets. After six iterations of message passing, the additional gains in MRR become negligible, while the computation cost increases dramatically. The original publication, therefore, suggests that paths of length larger than six can only contribute very little additional information.

This explanation aligns with the results reported on the FB15k-237 dataset and the observation by Huurdeman [18] that all shortest paths connecting s and o in the FB15k-237 validation set are two or three hops removed from each other. However, Huurdeman [18] also shows that 5% of validation examples in WN18RR have shortest paths connecting s to o with lengths longer than six hops, the majority of which are seven or eight hops long. Given this observation, one could expect a more significant increase in model performance on WN18RR between six and eight iterations. However, this does not seem to be the case, supporting Xhu et al.'s interpretation of a saturation effect.

	Number of Message Passing Layers				
	1	2	4	6	8
FB15k-237	-	0.345	0.409	0.415	0.416
WN18RR	0.360	0.422	0.537	0.553	0.555

Table 4.5: The effect of the number message passing iterations on MRR. The FB15k-237 results are presented as taken from table 6.b in [43]. The results reported on WN18RR are produced using the reproduction code-base. All other hyperparameters are set to the reproduction settings presented in table 4.2.

This finding raises the question of whether the skip connection across layers introduced in the original model is strictly necessary. Skip connections have been developed in computer vision to mitigate the problem of vanishing gradients when training deeper networks. The model in which they were originally proposed (ResNet [14]) was substantively deeper than the six-layer message-passing network under study. Therefore, it should be tested if the introduced skip connection can contribute to model performance.

Due to time limitations, this test has been conducted only on the WN18RR dataset. The results are presented in table 4.6. Removing the skip connection is shown not to impact model performance significantly. While the test case performs slightly better, given the confidence intervals for model performance as reported in table 4.3, it cannot conclusively be ruled out that the observed performance difference is driven by random model variance. This test indicates that introducing a skip connection is unnecessary to achieve the reported performance metrics.

Skip Connection	True	False
MRR	0.553	0.555

Table 4.6: Impact of the skip connection between layers of message passing on MRR for WN18RR. All hyperparameters are set to the reproduction settings presented in table 4.2.

As demonstrated in table 4.5, a working albeit bad model can be constructed only considering paths of length one on WN18RR. The replication settings provided in the initial publication prescribe removing paths of length one between s and o from the training dataset in FB15k-237. The authors argue this is done to avoid over-fitting and encourage the model to learn longer path representations. However, this thesis argues that FB15k-237 is an unsuitable test case for this assumption. As outlined above, FB15k-237's validation set contains no directly connected s o pairs, while the training data does. Therefore, it cannot be distinguished if this form of emphasis

One-Hop Removal	True	False
WN18RR	0.483	0.553
FB15k-237	0.413	0.372

Table 4.7: Testing the effects of removing all edges from the KG connecting subjects to objects in a batch of training data on MRR. False indicates that only the exact training examples are removed from the graph. All other hyperparameters are set to the reproduction settings presented in table 4.2.

Query Dependent Edge Representations	True	False
FB15k-237	0.416	0.408
WN18RR	0.555	0.553

Table 4.8: Effect of query-dependent edge representations compared to independently learned edge embeddings during each iteration of message-passing on MRR. All other hyperparameters are set to the reproduction settings presented in table 4.2.

on longer paths is generally beneficial or if positive effects can only be observed on the FB15k-237 dataset due to its unique characteristics. Please note further that this procedure is only recommended for FB15k-237, not for WN18RR, where only the complete training facts are removed from the graph during training.

Both approaches have been tested on both datasets to provide more insights into this question. The results of this test are shown in table 4.7 and B.3 in appendix B. It demonstrates that the model removing one-hops performs significantly worse than the model, having seen short connections during training on WN18RR, while the opposite is true for FB15k-237. This finding undermines the reasoning for one-hop removal, which is put forward in the original publication and provides further empirical evidence for the argument in [24] that FB15k-237 contains an unrealistic bias. These findings highlight the benefits of designing model architectures having graph-specific characteristics in mind. Furthermore, they demonstrated that the use of the FB15k-237 dataset as a general benchmark is not without problems, as it is not representative of the majority of knowledge graphs.

Query Dependent Edge Representations The first experiment going entirely beyond the ablation results presented in the original publication, considers the impact query-dependent edge relations have on model performance. As with one-hop removal, the hyperparameter is used as a reproduction setting for FB15k-237 but not for WN18RR. Unfortunately, the authors provide no explanation why this is

Scoring Input			
subject	predicate	object	MRR
✓	✓	✓	0.553
	✓	✓	0.553
		✓	0.552

Table 4.9: The effect of different combinations of inputs into the MLP scoring function. The representations of subject, predicate, and object are supplied to scoring after applying the neural Bellman-Ford network. Inputs are concatenated. The results are reported on WN18RR at reproduction hyperparameters.

the case. This justifies an investigation into the effect this design choice may have on model performance. Therefore, the two non-reproduction combinations have been trained and are shown in table 4.8 alongside the recommended settings.

This experiment reveals that the computation of query-dependent relation representations seems to improve model performance, with a larger improvement being achieved on FB15k-237 than on WN18RR. However, both improvements fail to achieve statistical significance. Therefore, it cannot be confidently ruled out that the observed benefit is an artefact of model variance.

Scoring based on o As described in section 3.1.2, NBFNet scores a given triple based on the representation of o concatenated with the representation of p. Further, it has been argued that the concatenation of p to the representation of o should not be strictly necessary as the initial representation of s is equivalent to the representation of the query relation p and the representation of o is in turn an $s - o$ pair representation. Therefore, scoring a given fact based on the representation of o without losing information should be possible.

This assumption has been put to an empirical test in table 4.9. Due to limited resources, this and all following experiments have been conducted only on WN18RR. The difference between both scoring scenarios is small and statistically indistinguishable from random model variation. Therefore, all information necessary for scoring a complete spo triple can be inferred from the representation of o alone and the concatenation of p with o is indeed unnecessary. The negative test of adding s to allow scoring on the entire spo triple produced an MRR of 0.553, thereby proving that no further information is gained from adding the representation of s.

Adversarial Temperature	0	0.25	0.5	0.75	1	1.25	1.5
MRR	0.512	0.556	0.556	0.555	0.553	0.554	0.552

Table 4.10: The effect of varying the temperature hyperparameter of the self-adversarial sampling on MRR for WN18RR. 0 denotes that self-adversarial sampling has not been used. All other hyperparameters are set to the reproduction settings presented in table 4.2.

Batchsize	32	64	128	256
MRR	0.551	0.558	0.553	0.556

Table 4.11: The effect of varying the batch size on MRR, based on WN18RR. All other hyperparameters are set to the reproduction settings presented in table 4.2. Mini-batch sizes are kept constant at 32.

General Hyperparameter Sensitivity Another topic of interest which is unfortunately not addressed in the original publication, is the general hyperparameter sensitivity of the model. Several experiments have been conducted on the WN18RR dataset to investigate this issue.

First, table 4.10 reports the effect of varying the adversarial temperature parameter of the loss computation. As apparent, the differences between different values > 0 are relatively minor. However, the reported MRR for 0.25 and 0.5 of 0.556 significantly outperforms the reproduction setting using a temperature of 1, while 1.5 and 0 significantly underperform. These findings show that this form of weighting how the negative examples contribute to the loss function positively impacts model performance. However, all settings above 0 are within 1% of the reproduction setting.

A second test for model sensitivity is provided in table 4.11, which presents the effect varying batch sizes have on model performance on the WN18RR dataset. Interestingly, the resulting data seems to show no trend. While batches of size 32 underperform significantly, batches of size 64 and 256 significantly improve performance. All these experiments have been conducted using a fixed mini-batch size of 32. While significant differences exist, the magnitude of these effects, while non-negligible, is only within at maximum $\approx 1\%$ of the overall performance.

In addition to varying the batch size, the number of negative examples has been systematically explored. In contrast to batch size, a clear trend is visible, as increases in negative examples per positive training example are associated with increasing model performance. All of the explored configurations are significantly

Negative Examples per Positive	16	32	64	128
MRR	0.547	0.553	0.557	0.562

Table 4.12: Effect of a varying number of negative examples per positive examples during training on MRR for WN18RR. All other hyperparameters are set to the reproduction settings presented in table 4.2

Representation Dimensionality	16	32	64	128
MRR	0.542	0.553	0.561	0.563

Table 4.13: The influence of increasing the number of dimensions used to represent entities and relations on MRR for WN18RR. Please note that the increased memory consumption did not allow for constant mini-batch sizes. Therefore, for each increase beyond 32, the mini-batch size is halved. All other hyperparameters are set to the reproduction settings presented in table 4.2

different from the replication settings. However, the largest overall difference in model performance observed between the replication and the experimental settings is $\approx 1.6\%$ of total model performance.

Last, the effect of the number of dimensions used to represent the entities and relations has been put to an empirical test. A clear trend is visible: The more dimensions are used to represent entities and relations, the better the model performs. Again all experimental results are significantly different from the replication model. In contrast to the last discussed experiment increasing the number of dimensions is associated with a substantive increase in model run-time and most problematic memory consumption. Therefore the mini-batch size is not consistent across different settings. The dimensionalities of 16 and 32 are reported with a mini-batch size of 32, while 64 corresponds to a mini-batch size of 16 and 128 to a mini-batch size of only 8. The largest difference in MRR between reproduction and the experimental setting is $\approx 2\%$.

Overall, it has been demonstrated that the model is stable regarding changes in hyperparameters within a reasonable range. While significantly better or worse configurations exist, the effects sizes are usually $< 2\%$ of the overall model performance. A notable exception is the usage of 0 as the adversarial temperature parameter. However, this instance is equivalent to not using this part of the learning pipeline and can therefore be considered a larger change to the training process.

Last, the frequency with which the presented set of experiments produced models which significantly outperformed the replication model is noteworthy. This

indicates a sub-optimal hyper-parameter search strategy deployed in the original publications.[3]

4.4 An Investigation into the Performance Advantage of NBFNet

Where experiments in the last section are limited to the model architecture as originally provided, this section seeks to go beyond these confines and to understand why the model performs substantively better than other GNN models. Therefore this section studies three of the differences outlined in the model description provided in section 3.1.2 by systematically changing the model architecture to better understand which aspects of the model are necessary to obtain the reported performance levels. Three aspects are studied: First, using an MLP as a scoring function is considered. Subsequently, the two architectural decisions that set NBFNet apart from traditional message passing are investigated by looking at message augmentation before considering the effect of the indicator function.

Again due to resource constraints, all experiments are conducted on the WN18RR dataset. If not reported otherwise, all results are products of single re-training runs, using the hyperparameters as reported in table 4.2.

4.4.1 Scoring

As previously established, most other GNN-based models rely on pre-defined scoring functions frequently taken from KGE methods. All of the above-presented models follow this approach except for GRAIL and INDIGO. This section studies if MLP-based scoring is a necessary condition for the superior performance of NBFNet by replacing the MLP with traditional KGE scoring functions.

To investigate the impact of the usage of MLP-based scoring as compared to more traditional KGE scoring functions, the MLP layer has been exchanged for the three scoring functions discussed above for message passing. However, this necessitates a further change to the model as these functions expect complete spo triples as inputs, whereas NBFNet is only scored based on concatenated o-p pairs. Therefore, the codebase has been extended with the option to score based on complete spo triples.

Another problem that needs to be addressed is the representations of p. Given the original reproduction hyperparameters, these are not used to compute the edge representations during message passing. It has been observed in [7] that using

[3] For more information on this issue see section 4.4.3.

Scoring Function	Message Function	MRR
TransE	DistMult	0.425
TransE	TransE	0.392
RotatE	DistMult	0.425
RotatE	RotatE	0.417
DistMult	DistMult	0.551

Table 4.14: The effect of exchanging the MLP-based scoring function with traditional KGE scoring functions contingent on the message function. The reported metric is MRR on WN18RR. Query-dependent edge representations are used, and all other hyperparameters are set to the reproduction settings presented in table 4.2

the relation representations only during scoring is not optimal. Therefore, the experiments in table 4.14 use query-dependent edge representations to alleviate this issue.

Furthermore, two scenarios have been examined, one in which the message function remains fixed to the DistMult function as prescribed in the reproduction parameters and one in which the message function is the adapted version of the scoring function. While the former more closely resembles the reproduction hyperparameter setting, the latter can be expected to produce representations that are more closely aligned with the transformation relation between s, p, and o assumed by the scoring function.

This assumption was wrong, as shown in table 4.14, as DistMult as a message function consistently outperforms RotatE and TransE. Even when using matching scoring and message functions, this pattern holds. Furthermore, the performance gap between the DistMult scoring function and its alternatives is remarkable, as the all DistMult approach achieves nearly the same performance as the original model.

These results demonstrate that using an MLP as a scoring function is not necessary to achieve state-of-the-art performance. Still, the choice of scoring and message function has a substantive impact on overall performance. However, given the absence of a broader hyperparameter search, it cannot be established if the above-shown differences vanish given a better-tuned set of hyperparameters or if they reflect meaningful differences in model capacity. Under the assumption that the larger differences would remain after appropriate hyper-parameter tuning, it could be argued that several scoring functions should be part of the general ablation process of GNN-based methods to uncover the best-performing combination of message-passing and scoring functions. Furthermore, the conducted experiments would indicate that using an MLP layer instead of a scoring function

Message Augmentation	✓		✓
Traditional Self Edges		✓	
Linear Layer as COMBINE function	✓	✓	
MRR	0.553	0.554	0.551

Table 4.15: Exploring the effect of message augmentation and a linear layer as COMBINE function during message passing. Results are MRR on WN18RR. Traditional self-edges indicate that self-edges are computed and contribute to the updated node representation like all other messages. All other hyperparameters are set to the reproduction settings presented in table 4.2

can improve model performance substantively compared to a sub-optimally chosen scoring function. This could be explained by the above mention agnosticism regarding the transformation relation between s,p, and o inherent in using an MLP. MLPs are universal approximators and can adapt according to the message-passing function. In contrast, using a pre-defined scoring function provides stricter limits and less adaptability for the model. Given the scope of the investigation into the factors which improve NBFNet over alternative GNN models and the limited time available for this thesis, this enquiry has to be left for future research.

4.4.2 Message Augmentation and COMBINE function

Message augmentation and the usage of a linear layer as COMBINE function are the next candidates to explain increases in model performance compared to other GNNs. Both issues are grouped together as both related to incorporating information from previous layers in the model into the message-passing process at a given later layer.

To test the usage of a linear layer as COMBINE function substantively increasing model performance, the model has been trained without any form COMBINE function, i.e. the previous representation of an entity did not affect the new representation at all. The results of this experiment are presented in table 4.15 on the WN18RR dataset. Surprisingly, while this new model performs significantly worse than the original, the difference is $< 1\%$ of the total model performance. This demonstrates that the linear layer as COMBINE function cannot explain the outstanding model performance reported in the paper.

Understanding if message augmentation is a necessary condition for the model is a little more involved. Due to the model construction, the self-edges in NBFNet are added only after aggregating all of the messages. Furthermore, please recall that NBFNet assumes that the set of all possible entities is known, but some en-

tities' connections to the KG might be absent from the training data. This means that some of the entities are not connected to the network. Simply removing the message augmentation would break the model. Some entities would receive empty sets of messages, leading to non-sensical results due to JAX defaulting to extreme values when asked to compute, for example, the maximum of an empty set.

Therefore, the experimental test needs to be constructed differently: The essential feature of message augmentation is that the bounding condition is re-introduced unchanged in every layer. If a simple self-edge is used instead, this characteristic is no longer given, while no entity receives an empty set of messages. However, this poses the question of whether the self-relation should be shared across entities or if every entity is assigned its own self-relation. The former solution is implemented for two reasons: First, given the memory constraints outlined in section 3.2.2, allowing for parameter sharing is preferable as it has a substantively smaller memory footprint. Second, it can be argued that an identity relation should be the same across entities in the same way that other relation types are shared regardless of the s and o.

The result of this experiment is also presented in table 4.15. This approach seems to improve the model rather than reduce its performance. While this effect is not significant, it can be ruled out that message augmentation as derived from the generalized Bellman-Ford algorithm is necessary to achieve the reported model performance.

4.4.3 Initialization by Indicator Function

The last and most significant difference between NBFNet and other GNNs needs to be addressed: Using an indicator function to initialize the entity representations. As shown above, this form of initialization introduces two dependencies: First, the initial s depends on the representation of p. Second, the initial representation of o depends on s, as all non-s entities are initialized uniformly to an all-zero vector. To understand the distinct contribution of both dependencies to model performance, both need to be studied separately by breaking the respective dependency. The s,p dependency can be broken by initializing s with a randomly initialized embedding. The dependence between s and o can be removed by initializing all non-s entities with randomly initialized embeddings. These embeddings are non longer uniform for all non-s entities, thereby removing the dependence. In addition, a learned zero representation is tested, i.e. all non-s entities are initialized uniformly, not to an all zero-vector but to a learned embedding. The original publication mentions that they experimented with learned zero-vectors, but the approach could not improve model performance; the test is added to confirm their finding.

However, breaking the dependencies comes with the added complication that

		Initialization of non-s entities		
			uniform	non-uniform
		$\vec{0}$	embedding	embedding
Initialisation of s	embedding	0.291	0.297	0.051
	$s = p$	0.553	0.553	0.408

Table 4.16: Varying the initialization of s and non-s entities in the KG. Embedding denotes that the representation is initialized using an embedding layer, while $\vec{0}$ denotes the all-zero vector. Query-dependent edge representations are used, and scoring is based on the concatenated representations of s,p and o. All other hyper-parameters are set to the reproduction settings presented in table 4.2

the scoring function must also be adapted. As the representation of o is no longer an entity pair representation, the representation of s needs to be added to the scoring function. For comparability in effect size to the experiments regarding the effect of the scoring function. The same measures have been applied, i.e. the scoring function receives as input vector the concatenation of s,p, and o. p is used during message passing, as query-dependent edge representations are used.

All resulting combinations are presented in table 4.16. The experimental results demonstrate that using a learned zero vector instead of initializing all non s entities with an all-zero vector provides no added benefit as long as both dependencies are conserved. Model performance decreases if s is initialized by a separate embedding instead of using the representation of p. Interestingly, breaking the s-o dependence seems to have a smaller impact on overall model performance when compared to breaking the p-s dependence. While both models lose substantial parts of their overall performance, a random initialization of s seems to have a larger effect on the model than the non-uniform initialization of all non-s entities. This is surprising, as the uniform initialization of all non s entities is central to the generalized Bellman-Ford algorithm.

In contrast, the initialization of s with p is an adaptation to accommodate the heterogenous nature of knowledge graphs rather than a central part of the generalized Bellman-Ford algorithm. Initializing s and non-s entities with random nodes seems to break the model. To check if this model is entirely unfunctional or bad, it has been tested on the UMLS dataset [10] where it achieved evaluation MMR scores of ≈ 0.8. This indicates that the model is not broken but rather bad.

As stated above, all results not subject to a hyper-parameter search need to be considered only indicative of the specific replication model and cannot be generalized to the broader family of possible NBFNet models. However, the findings in table 4.16 are the most pronounced so-fare and deserve further investigation.

Thus, three hyper-parameter searches have been conducted for the following three settings: random s and non-uniform non-s, $s = p$ and non-uniform non-s, random s and uniform all-zero non-s.

Unfortunately, Zhu et al. provide little information about the search space used to find their best model and the deployed search strategy. While table 12 in appendix F of their publication lists the hyperparameters for each model, the explored space remains unclear. Furthermore, the authors mention only having tuned, learning rate, number of epochs, adversarial temperature, and the optimizer. The kind of conducted search is not documented. The paper's first author has been contacted to understand the search space better. In addition to the information presented in the publication, the author stated that the number of negative examples per positive, the batch size, the removal of one-hops, and the usage of query-dependent edge representations had been tuned. The mail also contained some albeit incomplete information regarding the covered search space, but unfortunately, no indication of the deployed search strategy.

As the searches are conducted to understand better the importance of initializations removing one-hops is excluded from the search space. Furthermore, only one optimizer has been named in the original publication and the mail by the author; therefore, only the adam optimizer is used. Consequently, batch size, the number of negatives, the learning rate, and the number of negative samples are considered. The entire search space is provided in table B.4 in appendix B.

Given this search space, 20 random Sobol trials have been constructed utilizing the LibKGE framework [6]. Please note that each trial has been executed for ten epochs instead of the 20 epochs used so far. This is due to time constraints during the final phase of the thesis. Fortunately, the performance gains between epochs level out relatively quickly so that after ten epochs, most models only achieve further marginal gains.

This is demonstrated by a search conducted on the original model to strengthen the overall confidence in this search space. Unfortunately, this search used RotatE as the message-passing function instead of DistMult. However, as the prior ablation has demonstrated, this should not substantively affect the model performance. The search could be stopped after only four trials, as all trials achieved validation MRR scores larger than 0.52, while the best-performing model on validation scored 0.550 on test data. This shows that ten epochs are sufficient to obtain more than 99% of the original model's performance and that the search space is likely to contain well-performing hyperparameter settings.

The first search tested if the performance of the scenario in which s and all non-s entities were initialized with non-uniform entity embeddings could be improved. None of the 20 trials could improve over the result reported in table 4.16. The second search tested the scenario where s is initialised with a random entity

embedding, but all other nodes are initialized with a zero vector. Indeed the best trial on validation produced an improvement boosting the MRR on test to 0.299. Last, a search probed models in which s was initialized with the representation of p, but all other entities were initialized with random embeddings. Unfortunately, a server update close to the hand-in date of this thesis cut this search short. Only 16 of 20 trials could be finished. Among those that were finished, the best-performing model on validation could not improve upon the result presented in table 4.16. These results indicate that the conclusions drawn from 4.16 are likely applicable to the wider NBFNet family. They suggest that using an indicator function for initialising entity representations is instrumental to the overall model performance. Furthermore, distinguishing between the two dependencies enforced by the indicator function makes it apparent that neither one alone is sufficient to produce a well-performing model. Both seem necessary to obtain the state-of-the-art results confirmed in this paper.

Chapter 5

Conclusion

5.1 Summary

This thesis set out to accomplish two goals: First, contribute an independent reproduction of NBFNet's reported performance in link prediction in KGs. Second, provide additional insights about the model and identify which part of the model is responsible for the substantive performance increase compared to other GNNs on the same task.

To achieve these goals, this thesis first provided an overview of the related literature by discussing a selected set of GNNs specifically designed for link prediction in KGs and the KGE models most relevant to this thesis. These models served as baselines against which NBFNet could subsequently be compared.

Subsequently, the theoretical foundations of the NBFNet model in the generalized Bellman-Ford algorithm have been highlighted. It has been shown how the derivation from the generalized Bellman-Ford algorithm has impacted the model architecture, and it has been established that the resulting entity initialization by an indicator function, as well as the message augmentation, are products of this derivation, with no direct precedent in the literature on GNNs. Furthermore, a comparison with other relevant GNN-based models for link prediction revealed that most of the ideas central to NBFNet had been explored in other models but not been combined in a principled fashion.

To test the impact of these innovations and reproduce the strong performance reported in the original publication, an independent codebase has been developed based on the JAX ecosystem. Using a different deep-learning ecosystem achieved the greatest possible independence between the original and replication codebase. In this process, several problems were overcome, among them the non-existence of a message-passing framework with an equal memory-efficient implementation

of a message-passing algorithm. While this problem could be circumvented, the memory constraints remained a limiting factor for the conducted experiments. Furthermore, the limited hardware resources were problematized as they did not allow for data-parallel training as in the original publication. A sequential emulation of data-parallel training has overcome this problem.

This new independent codebase has been used to reproduce the results initially reported in the paper. The exact same level of performance could be obtained. Furthermore, for the first time, an estimate for the model variance has been provided by re-training the models three times. The resulting estimates for confidence intervals around the true model performance have been subsequently used to distinguish meaningful differences in model performance from those which could be products of random model variation.

Subsequently, a set of ablation experiments was conducted. These augmented the results reported in the original publication by repeating the same experiments on the WN18RR dataset. The results displayed the same patterns as reported on the FB15k-237 dataset, thereby strengthening the conclusions drawn in the original publication. Additionally, several tests have been performed to test the impact of specific design choices. It has been highlighted that removing all one-hop connections between s and o is only improving model performance due to a particular idiosyncrasy of the FB15k-237 dataset but hurts performance on the WN18RR dataset. Furthermore, query-dependent edge representations have been found to increase model performance generally, while the model displayed general robustness to changes in hyperparameters.

Last, several experiments have been conducted to investigate which of the novel architectural design choices are necessary to obtain significant gains in performance against other models. Replacing the scoring MLP layer with KGE score functions revealed that MLP-based scoring is not strictly necessary to obtain the reported performance results. Furthermore, the inclusion of self-edges via a linear layer and message augmentation have been tested. Neither seems to be strictly necessary to obtain comparable performance levels.

The last set of experiments demonstrated that the model is sensitive to changes in the indicator function used initialise entity representations. In separate experiments, the dependencies introduced between predicate and subject, as well as subject and object, have been broken separately after adapting the model's scoring function. The results demonstrated the model performance substantively decreases if only one of the two dependencies is removed. A hyperparameter search space has been designed and validated on the original model to demonstrate that his effect is not an artefact of sub-optimal hyperparameter choices. Based on this search space, three random searches have been conducted, which support the conjecture that using the indicator function for entity representation initialization is the driving

factor of the observed model performance.

5.2 Future Work

While this thesis has been able to conduct several experiments, some questions have to be left for future research. First, future studies could explore which type of facts the model learns more efficiently than others. While the authors provide an analysis by relation cardinality, their analysis shows that NBFNet performs better than the baseline across all categories, making the experiment less insightful. Interesting factors might include the distance between s and o for each tested fact or the total degree of o to measure how well the entity is connected to the KG. Furthermore, it might be interesting to study which impact the inductive capacity of NBFNet has on the observed performance gap between NBFNet and other GNN models.

This thesis was limited by the computing resources available and especially the high VRAM consumption of the model. Creating a message-passing algorithm as efficient as in the original publication in JAX could allow for more extensive experimentation on FB15k-237 and larger datasets.

Additionally, this thesis suggests including more datasets to better gauge the performance of NBFNet against other models. However, this suggestion is not limited to the further study of NBFNet but applies more generally. As argued in section 4.1, FB15k-237 has some features which make it a hard test setup at the cost of not being very representative of the broader landscape of knowledge graphs. Using this dataset as one of the few widely applied benchmarks could incentivize researchers to develop models more closely tailored to a benchmark but less performant in reinitialised applications.

While studying the literature, it became clear that most models aiming for inductive capabilities focus on unseen entities, while only a minority considers inductive reasoning for relations. Therefore it would be interesting to explore possibilities for both types of inductive reasoning simultaneously. This, however, would necessitate the usage of relation and/or node attributes. This thesis suggests that using word embeddings learned by large language models might be a good starting point for this research.

Bibliography

[1] Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. The DeepMind JAX Ecosystem, 2020. http://github.com/deepmind.

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

[3] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data: Application to word-sense disambiguation. *Machine Learning*, 94:233–259, 2014.

[4] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

[5] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[6] Samuel Broscheit, Daniel Ruffinelli, Adrian Kochsiek, Patrick Betz, and Rainer Gemulla. LibKGE - A knowledge graph embedding library for reproducible research. In *Proceedings of the 2020 Conference on Empirical*

Methods in Natural Language Processing: System Demonstrations, pages 165–174, 2020.

[7] Ling Cai, Bo Yan, Gengchen Mai, Krzysztof Janowicz, and Rui Zhu. TransGCN:Coupling Transformation Assumptions with Graph Convolutional Networks for Link Prediction. In *Proceedings of the 10th International Conference on Knowledge Capture*, pages 131–138, September 2019.

[8] Thomas H Cormen. *Introduction to algorithms*. MIT Press, Cambridge, Massachusett, fourth edition edition, 2022.

[9] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020.

[10] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[11] PyTorch Foundation. Distributed data parallel. https://pytorch.org/docs/master/notes/ddp.html, last accessed: 2023-03-13.

[12] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[13] Jonathan Godwin*, Thomas Keck*, Peter Battaglia, Victor Bapst, Thomas Kipf, Yujia Li, Kimberly Stachenfeld, Petar Veličković, and Alvaro Sanchez-Gonzalez. Jraph: A library for graph neural networks in jax., 2020. http://github.com/deepmind/jraph.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for JAX, 2023. http://github.com/google/flax.

[16] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen

Staab, and Antoine Zimmermann. Knowledge Graphs. *ACM Computing Surveys*, 54(4):1–37, May 2022.

[17] Guyue Huang, Guohao Dai, Yu Wang, and Huazhong Yang. Ge-spmm: General-purpose sparse matrix-matrix multiplication on gpus for graph neural networks. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE, 2020.

[18] Moritz Valentin Huurdeman. Leveraging symbolic information for knowledge graph embeddings. Master's thesis, Data and Web Science Group University of Mannheim, 2022.

[19] Hugging Face Inc. Text classification examples. https://github.com/huggingface/transformers/tree/main/examples/flax/text-classification, last accessed: 2023-03-13.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. arXiv:1412.6980 [cs].

[21] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[22] Shuwen Liu, Bernardo Grau, Ian Horrocks, and Egor Kostylev. Indigo: Gnn-based inductive knowledge graph completion using pair-wise encoding. *Advances in Neural Information Processing Systems*, 34:2034–2045, 2021.

[23] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime Bottom-Up Rule Learning for Knowledge Graph Completion. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 3137–3143, Macao, China, August 2019. International Joint Conferences on Artificial Intelligence Organization.

[24] Christian Meilicke, Manuel Fink, Yanjie Wang, Daniel Ruffinelli, Rainer Gemulla, and Heiner Stuckenschmidt. Fine-grained evaluation of rule-and embedding-based systems for knowledge graph completion. In *The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part I 17*, pages 3–20. Springer, 2018.

[25] Maximilian Nickel, Volker Tresp, Hans-Peter Kriegel, et al. A three-way model for collective learning on multi-relational data. In *Icml*, volume 11, pages 3104482–3104584, 2011.

[26] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You can teach an old dog new tricks! on training knowledge graph embeddings. *ICLR*, 2020.

[27] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*, pages 593–607. Springer, 2018.

[28] Sameer Singh, Amarnag Subramanya, Fernando Pereira, and Andrew McCallum. Wikilinks: A large-scale cross-document coreference corpus labeled via links to wikipedia. *University of Massachusetts, Amherst, Tech. Rep. UM-CS-2012*, 15, 2012.

[29] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2019.

[30] Komal Teru, Etienne Denis, and Will Hamilton. Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*, pages 9448–9457. PMLR, 2020.

[31] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, Beijing, China, 2015. Association for Computational Linguistics.

[32] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, et al. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[33] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, December 2017.

[34] Zihan Wang, Zhaochun Ren, Chunyu He, Peng Zhang, and Yue Hu. Robust Embedding with Multi-Level Structures for Link Prediction. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 5240–5246, Macao, China, August 2019. International Joint Conferences on Artificial Intelligence Organization.

[35] Oliver Wieder, Stefan Kohlbacher, Mélaine Kuenemann, Arthur Garon, Pierre Ducrot, Thomas Seidel, and Thierry Langer. A compact review of molecular property prediction with graph neural networks. *Drug Discovery Today: Technologies*, 37:1–12, December 2020.

[36] Feng Xia, Ke Sun, Shuo Yu, Abdul Aziz, Liangtian Wan, Shirui Pan, and Huan Liu. Graph Learning: A Survey. *IEEE Transactions on Artificial Intelligence*, 2(2):109–127, April 2021.

[37] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

[38] Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the International Conference on Learning Representations (ICLR) 2015*, 2015.

[39] Zi Ye, Yogan Jaya Kumar, Goh Ong Sing, Fengyan Song, and Junsong Wang. A Comprehensive Survey of Graph Neural Networks for Knowledge Graphs. *IEEE Access*, 10:75729–75741, 2022.

[40] Mohamad Zamini, Hassan Reza, and Minou Rabiei. A Review of Knowledge Graph Completion. *Information*, 13(8):396, August 2022.

[41] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):11, December 2019.

[42] Zhaocheng Zhu, Chence Shi, Zuobai Zhang, Shengchao Liu, Minghao Xu, Xinyu Yuan, Yangtian Zhang, Junkun Chen, Huiyu Cai, Jiarui Lu, Chang Ma, Runcheng Liu, Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. Torchdrug: A powerful and flexible machine learning platform for drug discovery, 2022.

[43] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems*, 34:29476–29490, 2021.

Appendix A

Program Code / Resources

The source code, documentation, usage examples, raw data, and PDF version of this thesis are also contained on the CD-ROM attached to this thesis.

Appendix B

Further Materials

Message	AGGREGATE	MR	MRR	H@1	H@3	H@10
TransE	Sum	707	0.428	0.317	0.502	0.620
	Mean	748	0.415	0.302	0.489	0.610
	Max	702	0.520	0.465	0.543	0.6323
	PNA	673	0.536	0.482	0.560	0.644
DistMult	Sum	647	0.538	0.482	0.560	0.650
	Mean	692	0.528	0.474	0.548	0.640
	Max	717	0.535	0.482	0.556	0.642
	PNA	636	0.553	0.499	0.577	0.664
RotatE	Sum	655	0.542	0.489	0.559	0.648
	Mean	683	0.531	0.478	0.551	0.642
	Max	725	0.539	0.486	0.560	0.648
	PNA	631	0.548	0.494	0.572	0.660

Table B.1: Effect of varying the message and aggregation function in WN18RR. All other hyperparameters are set to the replication settings presented in table 4.2

#Layers	MR	MRR	H@1	H@3	H@10
1	8382	0.360	0.355	0.358	0.368
2	1753	0.422	0.401	0.430	0.460
4	889	0.537	0.491	0.557	0.637
6	636	0.553	0.499	0.577	0.664
8	530	0.555	0.498	0.580	0.670

Table B.2: The effect of varying the number of message passing iterations on WN18RR. All other hyperparameters are set to the replication settings presented in table 4.2

Dataset	One-Hop Removal	MR	MRR	H@1	H@3	H@10
WN18RR	True	0.483	0.483	0.327	0.515	0.657
WN18RR	False	636	0.553	0.499	0.578	0.663
FB15k-237	True	111	0.413	0.320	0.453	0.596
FB15k-237	False	116	0.372	0.271	0.412	0.576

Table B.3: Testing the effects of removing all edges from the KG connecting subjects to objects in a batch of training data on MRR. False indicates that only the exact training examples are removed from the graph. All other hyperparameters are set to the reproduction settings presented in table 4.2.

Hyperparameter	Type	Explored Options
batch size	choice	[32, 64, 128]
learning rate	range	(0.0005, 0.05)
adversarial temperature	range	(0.0, 2.0)
negative samples per positive	choice	[32, 64, 128]

Table B.4: The explored hyperparameter search space for the searches described in section 4.4 when discussing the effects of different initializations of s, and non-s entities.

Notation	Meaning
d	number of dimensions in the representation of relations and entities
f	an individual fact
h	representation of node during message-passing
l	counter for message passing iterations
n	number of negative sampels
o	object
p	predicate
r	edge type or the representation of an edge type
s	subject
u	placeholder of a node in a network
v	placeholder of a node in a network
x	placeholder of a node in a network
D	vector containing the number of neighbours for each entity in a KG
L	number of message passing layers
M	vector containing all messages
O	Vector holding all receiving nodes for a set of edges
P	probability
W	weight matrix
$\mathcal{E}$	set of all entities in a KG
$\mathcal{F}$	set of all facts in a KG
$\mathcal{G}$	an individual KG
$\mathcal{N}$	set of neighbours of a node
$\mathcal{R}$	set of all relations in a KG
$\mathcal{T}$	set of all facts in test or validation
$\otimes$	generalized multiplication
$\oplus$	generalized summation
$\otimes$	Kronecker product

Table B.5: An overview about the used notation in this thesis.

Acronyms

BCE Binary Cross-Entropy loss.

GCN Graph Convolutional Network.

GNN Graph Neural Network.

JIT Just In Time compilation.

KG Knowlege Graph.

KGE Knowledge Graph Embedding model.

MLP Multi-Layer Perceptron.

MPGNN Message-Passing Graph Neural Network.

MR Mean Rank.

MRR Mean reciprocal rank.

NBFNet Neural Bellman-Ford network.

PNA Principel Neighbourhood Aggregation.

ReLU Rectified Linear Unit.

XLA Accelerated Linear Algebra.

Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Masterarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Mannheim, den 15.03.2023 Unterschrift