7-29-2021

Reproducing and Explaining Entity and Relation Embeddings for Link Prediction in Knowledge Graphs

Narayanan Asuri Krishnan
nk1581@rit.edu

Reproducing and Explaining Entity and Relation Embeddings for Link Prediction in Knowledge Graphs

by

Narayanan Asuri Krishnan

A Thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in Computing and Information Sciences

B. Thomas Golisano College of Computing and Information Sciences
Rochester Institute of Technology

29th July 2021

Abstract

Embedding knowledge graphs is a common method used to encode information from the graph at hand projected in a low-dimensional space. There are two shortcomings in the field of knowledge graph embeddings for link prediction. The first shortcoming is that, as far as we know, current software libraries to compute knowledge graph embeddings differ from the original papers proposing these embeddings. Certain implementations are faithful to the original papers, while others range from minute differences to significant variations. Due to these implementation variations, it is difficult to compare the same algorithm from multiple libraries and also affects our ability to reproduce results. In this report, we describe a new framework, AugmentedKGE (aKGE), to embed knowledge graphs. The library features multiple knowledge graph embedding algorithms, a rank-based evaluator, and is developed completely using Python and PyTorch. The second shortcoming is that, during the evaluation process of link prediction, the goal is to rank based on scores a positive triple over a (typically large) number of negative triples. Accuracy metrics used in the evaluation of link prediction are aggregations of the ranks of the positive triples under evaluation and do not typically provide enough details as to why a number of negative triples are ranked higher than their positive counterparts. Providing explanations to these triples aids in understanding the results of the link predictions based on knowledge graph embeddings. Current approaches mainly focus on explaining embeddings rather than predictions and single predictions rather than all the link predictions made by the embeddings of a certain knowledge graph. In this report, we present an approach to explain all these predictions by providing two metrics that serve to quantify and compare the explainability of different embeddings.
From the results of evaluating aKGE, we observe that the accuracy metrics are better than the accuracy metrics obtained from the standard implementation of OpenKE. From the results of explainability, we observe that the horn rules obtained explain more than 50% of all the negative triples generated.

The thesis "Reproducing and Explaining Entity and Relation Embeddings for Link Prediction in Knowledge Graphs" by Narayanan Asuri Krishnan has been examined and approved by the following Examination Committee:

Dr. Carlos R Rivero
Assistant Professor
Thesis Committee chair

_____

Signature

_____

Date

Dr. Zack Butler
Professor

_____

Signature

_____

Date

Dr. Michael Mior
Assistant Professor

_____

Signature

_____

Date

Contents

List of Tables

Chapter 1

Introduction

Knowledge graphs are described as graphs of data where nodes represent entities of interest and edges represent the relationships between the nodes [18]. Knowledge graphs are represented as triples of the form (head entity, relation entity, tail entity), where the head and the tail are connected by a relation, e.g. (USA, hasCapital, Washington D.C.), which implies that the capital of USA is Washington D.C.

Knowledge graph embeddings are one of the common methods used to encode information from a knowledge graph [34, 11]. The goal of these embeddings is to project a knowledge graph into a low dimensional space while maintaining the structure of the graph [11]. Embedding graphs is not an easy task as each graph is unique and has a unique set of nodes and relations [11]. Knowledge graphs may contain data redundancy, which can bias the learning process of the embeddings towards the redundant part of a graph [3, 32].

Knowledge graph embeddings are used for various purposes like knowledge graph completion [9], relation extraction [36], entity classification [26], entity resolution [26] and link prediction [35, 9, 25, 31]. In this report, the focus is on using knowledge graph embeddings for link prediction. Link prediction is the task of predicting an entity that is connected to another entity by a given relation, i.e., predicting the head entity when relation (r) and tail (t) are given, or predicting the tail entity when head (h) and relation (r) are given [9]. When predicting the head entity, all entities e in the graph at hand are used, such that we evaluate triples (e, r, t). There (e, r, t) are considered positive if they exist in the graph, and negative if they are not present. A

similar process occurs when predicting the tail. A prediction is in the form of a score that measures the plausibility of each input triple, i.e., whether a given triple should exist in the given knowledge graph.

To convert entities and relations in a knowledge graph to embeddings, there exist a larger number of embedding algorithms [34, 11]. These embedding algortihms are either based on distance or based on similarity scoring functions. Distance-based approaches have scoring functions that are based on the distance between the embeddings of the two entities h and t. Some distance-based approaches perform translations based on relation r before distance is computed [9, 35, 19]. Similarity based algorithms compute scores by measuring the similarity between embeddings of h, r and t [25, 20, 33]. These embeddings are sub-symbolic representations of the entities and relations in the graph at hand, and are, therefore, not easy to interpret and explain [27].

When evaluating knowledge graph embeddings on link prediction, the goal is to rank based on score a positive triple over a (typically large) number of negative triples. It is possible that a positive triple may have a lower score, higher score or equal score to each negative triple. When the score of a positive triple is equal to that of a negative triple, it is called a tie [32]. There are three ways to calculate the rank of an individual triple [6]: optimistic, pessimistic and realistic. Optimistic rank assumes that the triple under evaluation is ranked first amongst triples that are tied. Pessimistic rank assumes that the triple under evaluation is ranked last amongst triples that are tried. Realistic rank is the mean of the optimistic and pessimistic ranks. These individual ranks of all positive triples are used to calculate metrics, such as Mean Rank (MR, the mean of the ranks), Mean Reciprocal Rank (MRR, the mean of the reciprocal of the ranks), Hits@K (H@K, the number of ranks that are less than or equal to K) [34] and Weighted Mean Rank (WMR, the weighted mean of all the ranks) [32]. These metrics measure the accuracy of embeddings and are helpful to compare them.

Problem Statement This thesis focuses on two shortcomings in the field of knowledge graph embeddings for link prediction.

The first shortcoming is that, as far as we know, current software frameworks to compute knowledge graph embeddings differ from the original papers proposing these embeddings. Certain implementations are faithful to the original papers, while others range from minute differences to significant variations.

For example, OpenKE [17] does not feature the same normalizations of embeddings as several of the original papers propose. Due to these variations, it is difficult to compare the implementations of the same embedding approaches available in multiple libraries. This also affects our ability to reproduce results as there are several factors that can change the results, such as normalization of embeddings, batch processing of data, and hyperparameters values. We develop a new framework AugmentedKGE (aKGE). aKGE features multiple knowledge graph embedding approaches that are implemented as close as possible, to the best of our efforts, to the original papers that describe them. The framework also features a rank-based evaluator. The evaluator calculates accuracy metrics (MR, MRR, H@K), computes realistic ranks of the positive triples taking ties into account, and stores the metrics, ranks and generated triples for further use. aKGE in its entirety is developed using Python and PyTorch. We provide a concise, Python-based summary in the form of a table of our implementations. This contributes to the reproducibility and extensibility of our library.

The second shortcoming is that the evaluation process of link prediction consists of ranking based on scores a positive triple, typically present in the test split, with respect to the negative counterparts derived from the positive triple. Accuracy metrics like MR, MRR, and H@K are aggregations of the ranks of the positive triples under evaluation. These accuracy metrics do not provide enough details as to why a number of negative triples are ranked higher than the positive triples. Providing explanations to these triples aids in understanding the results of the link predictions computed based on knowledge graph embeddings and, as far as we know, has not been addressed yet in the literatures. We mine Horn rules from the negative triples that are ranked higher than the positive triples. Using these rules, we provide metrics that can enhance the explainability of these incorrect triples. For example, one such metric suggests what percentage of negative triples that be explained using these Horn-like rules.

Experiments and results When comparing the implementations of aKGE with some of the existing embedding frameworks, we see that aKGE follows the implementations as close to the original papers as possible. OpenKE [17] features deviations like missing normalisations of embeddings and implementations differing from the original papers that contain these embedding algo-

rithms. PyKeen [4] and TorchKGE [10] follow closely to the original papers, but have some normalisations of embeddings missing.

Our experiments to evaluate aKGE uses eight datasets commonly used in the task of link prediction [17]: FB13, FB15K, FB15K-237, WN11, WN18, WN18RR, NELL-995 and YAGO [30]. FB13, FB15K and FB15K-237 are extracted from Freebase [8], which is a knowledge graph based on general facts. WN11, WN18 and WN18RR are extracted from WordNet [23] which is a large lexical knowledge graph of english. NELL-995 is extracted from NELL [12]. The embedding algorithms implemented in aKGE that are evaluated are as follows: ComplEx [33], DistMult [37], HolE [25], RotatE [31], SimplE [20], TransD [19], TransE [9] and TransH [35]. All the embedding approaches achieve very high WMR values with TransE achieving values very close to 1. When comparing the results with the results of the standard implementation of OpenKE [32], we observe that almost all our implementations have higher GMR and WMR values. We also observe that all our implementations perform better than AMIE [14], which is used as a baseline.

For the results of explainability in the process of link prediction, only five datasets were used: FB13, FB15K, FB15K-237, WN18 and WN18RR. This was done so that the differences in explainability could be highlighted when comparing the FreeBase datasets amongst themselves and the WordNet datasets amongst themselves. The embedding algorithms used were as follows: DistMult, HolE, RotatE, TransE and TransH. These embeddings were obtained from the experiments run on aKGE. The results show that except for FB15K, we are able to provide Horn rules that cover more than 50% of the negative triples that were deemed plausible by the model. TransE and RotatE consistently achieve the highest explainability scores. When comparing between the Freebase datasets, FB15K-237 has the highest percentage of negative triples covered. When comparing between the WordNet datasets, on average, WN18 has higher percentage of triples covered. Analysis of the rules generated and their corresponding explainability show that the rules that cover the largest number of negative triples are the ones that are easy to provide explanations for.

Report Organization The rest of the report is organized as follows. Chapter 2 explains training of embeddings, link prediction, generating negatives, and rule mining. Chapter 3 contains descriptions of exisiting libraries to em-

bed knowledge graphs and works done to explain embeddings and link prediction. Chapter 4 contains the different approaches used for embeddings, loss functions used in training and the implementation comparisons between aKGE and other libraries. Chapter 5 provides background into materialising triples in the evaluation process and the metrics used to provide the explainability. Chapter 6 discusses the results of link prediction for aKGE and the results of the explainability study on various embedding approaches. Chapter 7 discusses the conclusions from this study. Appendix A contains the various notations used throughout the report as well as some shorthands used for common functions in the report. Appendix B contains the results from the standard implementation of OpenKE.

Chapter 2

Related Works

This chapter contains descriptions of some libraries used to embed knowledge graphs. The second half of the chapter details the approaches taken in providing explanations to the field of knowledge graph embeddings and link prediction.

2.1 Embedding Libraries

This section discusses and describes five libraries used to embed knowledge graphs: OpenKE, AmpliGraph, GraphVite, PyKeen and TorchKGE. OpenKE, PyKeen, TorchKGE and aKGE are developed using Python and PyTorch whereas AmpliGraph uses another machine learning library called Tensorflow [1]. PyTorch [28] is a machine learning library written in Python that performs tensor computations and provides an automatic gradient interface for learning. This means that we do not need to explicitely calculate the gradients and they are calculcated automatically. PyTorch uses the GPU to accelerate these computations, making it faster and more memory efficient. PyTorch has extensive documentation and makes use of functions to perform required computations. It is also compatible with various mathematical libraries like NumPy, Pandas and figure plotting libraries like Matplotlib.

2.1.1 OpenKE

OpenKE [17] is an open toolkit for knowledge graph embedding written using Python and PyTorch [28]. The goal of OpenKE is to provide a unified interface that can be used by both academia and the industry. The library also contains pre trained embeddings of several well-known knowledge graphs. OpenKE provides an interface which encapsulates several data and memory processing functions, so that the user does not need to perform neither such data processing nor stitch various parts of the embeddings together.

OpenKE does not implement the validation step for early stopping but instead focuses on improving training time and reducing space complexity. OpenKE uses PyTorch whose API provides GPU acceleration and automatic derivation for machine learning applications. For large scale knowledge graphs, they also provide lightweight C/C++ versions which can train in a very fast time

Parallel learning is employed by OpenKE to speed up the time. It enables the use of multiple threads to perform the training. The triples are broken down into subsets and each subset is trained on a different thread. There are two ways for gradient updating, the first one is to have all the threads share the embedding space and update the gradients with no synchronization. The second way is to have each of the threads compute their own gradients and then sum all the gradients and then update the embeddings based on the total gradient.

2.1.2 AmpliGraph

AmpliGraph [13] is a library that uses Python and Tensorflow for knowledge graph embeddings. It has an intuitive API and is designed to reduce the amount of code written to train, test and learn embeddings. It provides GPU support for faster training and testing times, and is extendable, providing the necessary APIs to allow plugging in of new models with ease. The library provides interfaces to load the standard knowledge graphs used for link prediction but also allows importing custom knowledge graphs using functions, and provides functions to split the graph into training, test and validation sets. AmpliGraph follows a negative sampling strategy as described in [9]. AmpliGraph provides a validation interface while training.

2.1.3 GraphVite

GraphVite [39] is a library in Python that provides a high-performance CPU-GPU hybrid interface to embed graphs. The first step in node embeddings is to augment the network using random walks, i.e., choosing random paths in the graph where the size of the paths are within a specified distance. After augmenting, the size of the dataset is so large that it is not possible to load it into main memory. This is where the parallel online augmentation comes in. It generates samples on the fly without needing to store all the information. Since node embeddings are gradient exchangeable, i.e., changing the order on inputs has very little to no significant impact on the gradients, negative sampling can be parallelized by using the GPU. Each GPU updates the embeddings asynchronously. GraphVite introduces a new collaboration strategy was proposed where two pools are assigned to the CPU and CPU where each of them are working on different pools. This way both parallel augmentation and parallel negative sampling can both happen

2.1.4 PyKeen

PyKeen [4] is a graph embedding library written in Python using PyTorch. PyKeen comprises a number of modules implementing, for instance, training strategies, loss functions, and negative sampling. Through inheritance these modules are replaceable. This modular architecture aims to reduce the developer efforts to include new embedding approaches. PyKeen provides functionality to evaluate a set of embeddings. It supports multiple metrics as MR, MRR, Hits@k and some others like Adjusted Mean rank [6]. It also has an integrated hyperparameter optimization framework for tuning of hyperparameter optimization for producing the best results. Since different computers have different memory configurations, PyKeen provides automatic detection of training and batch size to prevent using more memory than available.

2.1.5 TorchKGE

TorchKGE is a graph embedding library developed using Python and PyTorch. TorchKGE provides a validation step during training for early stopping. For data loading TorchKGE allows the user to import their own datasets or use the standard datasets used for link prediction. It also supports splitting the graph

into training, test and validation sets. TorchKGE supports three negative sampling strategies: Uniform, Bernoulli and positional negative sampling.

2.1.6 Discussion

Each library contains different models, loss functions, corruption strategies and training methods. There are also differences between the libraries like the type of ranks they compute, availability of hyperparameter optimization within the library and the differentiation process.

When normalising parameters in the training process, it is unclear whether this normalisation should be included as part of the differentiation process or not. When comparing corruption strategies, certain libraries support corruption of both head and tail entities at the same time, and even the corruption of the relation. OpenKE and TorchKGE support corruption of head, tail and relations at the same time whereas PyKeen only supports corruption of either head, tail or relation at any given time. AmpliGraph supports corruption of head and tail entities at the same time but does not support the corruption of relations.

During the training process, the training can be stopped based on some early stopping criteria. This is done to prevent overfitting. OpenKE and TorchKGE do not contain early stopping, which is available in AmpliGraph and PyKeen. When comparing ranks computed during the evaluation process, OpenKE focuses on optimistic ranks. AmpliGraph, PyKeen and TorchKGE provide all three ranks which are optimistic, pessimistic and realistic ranks.

Regarding hyperparameter optimization, OpenKE suggests to use the same values as reported in the original papers. It also does not provide any hyperparameter optimization. PyKeen offers hyperparameter optimization through a framework called Optuna [2]. Ampligraph provides optimization through grid search and random search, and as far as we know, TorchKGE does not provide any hyperparameter optimization.

2.2 Explainability

To the best of our knowledge, explaining knowledge graph embeddings for link prediction has not been researched as thoroughly as other topics in the domain. Zhang et al. [38] focuses on explaining knowledge graphs through

closed-paths. Given a predicted triple (h, r, t), they aim to explain the triple by deriving a path between h and t that covers all the triples. For example, a prediction $meronym(hand, nail)$ is given by path $meronym(finger, nail)$ and $meronym(hand, finger)$ where $meronym$ between A and B implies B is a part of A. Support measures how many times $meronym(A, B)$ occur when $meronym(A, Z)$ and $meronym(Z, B)$ appear. The quality of these explanations is measured through recall and average support of the rules. Although this method provides explanation of single predictions, they fail to provide explanations as a whole.

Ruschel et al. [29] focuses on explaining knowledge graph embeddings and their predictions by training an interpretable classifier around them. For each triple, the closed-paths for these triples are used to train the classifier. After training, horn rules are derived and used to provide explanations to the predictions. In this approach, the extraction of paths is directly related to the algorithm they use (SFE [15]) and is not independent to be used with any knowledge graph embedding algorithms.

Chapter 3

Background

This chapter contains information on how embeddings are trained, link prediction, calculation of ranks and accuracy metrics, and some basics about rule mining.

3.1 Training knowledge graph embeddings

Given a knowledge graph, the goal is to encode the graph at hand into embeddings. An embedding can be described as a vectors of numbers. There are three steps for knowledge graph embedding techniques [34]: 1. Representing entities and relations, 2. Defining a scoring function, and 3. Learning the embeddings for entities and relations. The first step involves assigning a randomly initiated embedding to each entity and relation. Each embedding approach has a set of parameters that are used in the embedding process. For example, some approaches initialise embeddings with real numbers, whereas, other approaches initialise embeddings with complex numbers. Some approaches require additional embeddings apart from entity and relation embeddings. The different approaches and their parameters are described in section 4.1.

Once the required embeddings are initialised, we define a scoring function. Scoring functions are either distance-based or similarity-based. The scoring function returns a value that determines the plausibility of a triple. The plausiblity of a triple determines whether it should appear in the knowledge graph at hand. Using the scoring function, we calculate a score from every triple (h, r, t) that appears in the graph.

We then define a loss function that takes the score as input and run stochastic gradient descent to optimize the embeddings based on the loss function. When training the embeddings of triples that appear in the graph, we synthetically generate triples that are not in the graph, and use the scores of these negative triples to further optimize the embeddings.

3.2 Generating negative triples

To generate negative triples, we can either corrupt the head or the tail of a given triple by replacing it with a random entity such that the corrupted triple does not exist in the knowledge graph [9]. Given a triple $(h, r, t) \in \mathbb{D}^+$ where $\mathbb{D}^+$ is the set of all triples that are in the knowledge graph, C_h is the set of all triples with head corrupted and C_t is the set of all triples with the tail corrupted. $\mathbb{D}^-$ is the union of C_h and C_t, and set of entities E and relations R as follows:

$$C_h = \{(h', r, t) | (h, r, t) \in \mathbb{D}^+ \wedge h' \in E \wedge h' \neq h \wedge (h', r, t) \notin \mathbb{D}^+\}$$

$$C_t = \{(h, r, t') | (h, r, t) \in \mathbb{D}^+ \wedge t' \in E \wedge t' \neq t \wedge (h, r, t') \notin \mathbb{D}^+\}$$

$$\mathbb{D}^- = C_h \cup C_t$$

3.3 Link Prediction

Link prediction is the task of predicting the head entity of a triple when the predicate and tail entity are given and predicting the tail entity when the predicate and the head entity are given. After the embeddings have been trained for a given knowledge graph, we evaluate these embeddings for the process of link prediction. For each positive triple, we first generate corresponding negative triples using techniques illustrated in section 3.2. We compute the scores for the positive triple and all the negative triples. Using these scores, we rank the positive triple against the negative triples. Let C_h be the set of all corrupted head entities and C_t be the set of all corrupted tail entities for the positive triple under evaluation and $\mathbb{D}^+$ as the set of all positive triples. We

also define ρ_h as the rank of the positive triple under evaluation with respect to triples with corrupted head, ρ_t as the rank with respect to triples with corrupted tail and $f_r(h,t)$ as the scoring function. For a given triple (h,r,t), we define three types of ranks [6]:
Optimistic rank: The positive triple is ranked first amongst triples with equal scores

$$\rho_h^+(h,r,t) = |\{(h',r,t) \mid (h',r,t) \in C_h \wedge f_r(h',t) < f_r(h,t)\}| + 1$$

$$\rho_t^+(h,r,t) = |\{(h,r,t') \mid (h,r,t') \in C_t \wedge f_r(h,t') < f_r(h,t)\}| + 1$$

Pessimistic rank: The positive triple is ranked last amongst triples with equal scores

$$\rho_h^-(h,r,t) = |\{(h',r,t) \mid (h',r,t) \in C_h \wedge f_r(h',t) < f_r(h,t)\}|$$

$$\rho_t^-(h,r,t) = |\{(h,r,t') \mid (h,r,t') \in C_t \wedge f_r(h,t') < f_r(h,t)\}|$$

Realistic rank: This is the average of the optimistic and pessimistic rank

$$\rho_h(P) = \frac{1}{2}(\rho_h^+(P) + \rho_h^-(P))$$

$$\rho_t(P) = \frac{1}{2}(\rho_t^+(P) + \rho_t^-(P))$$

During the evaluation process, we compute arithematic mean rank [32] also called mean rank

$$AMR = \frac{1}{2\,|\mathbb{D}^+|}\left(\sum_{(h,r,t)\in\mathbb{D}^+} \rho_h + \rho_t\right)$$

The geometric mean [32] is defined as

$$GMR = exp\left(\frac{1}{2\,|\mathbb{D}^+|}\left(\sum_{(h,r,t)\in\mathbb{D}^+} ln(\rho_h) + ln(\rho_t)\right)\right)$$

and the weighted geometric mean [32] is defined as

$$WMR = exp\left(\frac{\sum_{(h,r,t)\in\mathbb{D}^+} |C_h|\, ln(\rho_h) + |C_t|\, ln(\rho_t)}{\sum_{(h,r,t)\in\mathbb{D}^+} |C_h| + |C_t|}\right)$$

3.4 Mining Rules

Given a knowledge graph, it has been shown that Horn rules can be mined from it that represent the triples in the graph [22]. An atom is a triple that has variables in the subject and/or object of a triple [14]. A triple (x, r, z) can be written as $r(x, z)$. A horn rule is defined as a rule with multiple body atoms but only one head atom [14]. For example, $isCapital(A, B) \Rightarrow inCountry(A, B)$ is an example of a Horn rule where $isCapital$ is the body atom, and $inCountry$ is the head atom and A and B are the variables. If the variables of the rule have been replaced by entities in the graph, the rule is called an instantiated rule. For example, $isCapital(paris, france) \Rightarrow inCountry(paris, france)$ where the entities in the graph are Paris and France.

Horn rules can be mined from a knowledge graph [14]. There are 4 parameters that can be used to judge a rule [14]: Support, Head Coverage, Confidence and PCA Confidence. Support is defined as the number of distinct pairs of subjects and objects that appear in the instantiations of a rule. Head coverage of a rule is the ratio of triples that are covered by the rule. Standard confidence of a rule is defined as the ratio of facts derived from the rule that are true in the knowledge graph. Standard confidence takes the triples that are not in the graph as negatives. PCA confidence only considers triples in the graph and considers the body as negatives.

Chapter 4

Embeddings

This chapter describe the various approaches that can be taken to obtain embeddings for a knowledge graph and the loss functions used in training. The second half of the chapter describes $aKGE$ and the implementations of these embedding approaches in Python and PyTorch.

4.1 Embedding Approaches

Table 4.1 describes the parameters used by each approach. Each parameter is an embedding of dimension k or dimension d as required by the approach. Each embedding can either contain real numbers or can contain complex numbers. Given entities h, t and relation r, we define their corresponding embeddings as $\mathbf{h}$, $\mathbf{t}$ and $\mathbf{r}$.

4.1.1 ComplEx

In ComplEx [33], the embeddings of entities and relations are in the complex number space $\mathbb{C}$. The use of complex numbers is to capture anti symmetric relations in the knowledge graph at hand. The scoring function for ComplEx is defined as

$$\begin{aligned} & f_r(h,t) = Re(\langle \mathbf{r}, \mathbf{h}, \bar{\mathbf{t}} \rangle) \\ & = \langle Re(\mathbf{r}), Re(\mathbf{h}), Re(\mathbf{t}) \rangle + \langle Re(\mathbf{r}), Im(\mathbf{h}), Im(\mathbf{t}) \rangle \\ & + \langle Im(\mathbf{r}), Re(\mathbf{h}), Im(\mathbf{t}) \rangle - \langle Im(\mathbf{r}), Im(\mathbf{h}), Re(\mathbf{t}) \rangle \end{aligned}$$

Table 4.1: Embedding approaches and their parameters

Model	Parameters
ComplEx	$\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{C}^d$
DistMult	$\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{R}^d$
HolE	$\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{R}^d$
RotatE	$\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{C}^d$
SimplE	$\mathbf{h}, \mathbf{r}, \mathbf{t}, \mathbf{r^{-1}} \in \mathbb{R}^d$
TransD	$\mathbf{h}, \mathbf{t}, \mathbf{w_h}, \mathbf{w_t} \in \mathbb{R}^k$ $\mathbf{r}, \mathbf{w_r} \in \mathbb{R}^d$
TransE	$\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{R}^d$
TransH	$\mathbf{h}, \mathbf{r}, \mathbf{t}, \mathbf{w_r} \in \mathbb{R}^d$

where $\langle x, y, z \rangle$ represents the dot product of x,y and z, Re and Im represent the real and imaginary parts of a complex number. $\bar{t}$ represents the conjugate of a complex number. ComplEx requires $||\mathbf{h}||_2 \leq 1$, $||\mathbf{t}||_2 \leq 1$, $||\mathbf{r}||_2 \leq 1$

4.1.2 DistMult

Instead of introducing full matrices for relations, DistMult [37] uses diagonal matrices M_r and introduces relational embedding $r \in \mathbb{R}^d$ where $M_r = diag(r)$. A diagonal matrix is defined as a matrix where all other elements except the main diagonal are zero. The scoring function is defined as

$$f_r(h,t) = h^T * diag(r) * t = \sum_{i=0}^{d-1} \mathbf{r_i} * \mathbf{h_i} * \mathbf{t_i}$$

$||\mathbf{h}||_2 = 1$, $||\mathbf{t}||_2 = 1$, $||\mathbf{r}||_2 \leq 1$

4.1.3 HolE

HolE [25] represents both entity and relation vectors in space $\mathcal{R}^d$. The scoring function is circular correlation as is defined as

$$f_r(h,t) = \mathbf{r^T}(\mathbf{h} \star \mathbf{t}) = \mathbf{r^T} * \mathbb{F}^{-1}(\bar{F(\mathbf{h})} * F(\mathbf{t}))$$

where $F(a)$ = Fourier transform of input a, $\bar{F(a)}$ = Conjugate of input a and F^{-1} is the inverse fourier transform. The conjugate of a complex number $a+bi$ is $a-bi$. HolE imposes constraints like $||\mathbf{h}||_2 \leq 1$, $||\mathbf{t}||_2 \leq 1$, $||\mathbf{r}||_2 \leq 1$

4.1.4 RotatE

RotatE [31] takes motivation from the Euler's identity that $e^{i\,\theta} = cos(\theta) + i\,sin(\theta)$, which indicates that a complex number can be mapped as a rotation in a complex plane. RotatE maps entities and relations to complex plane of dimension $\mathbb{C}$ and defines relation as a rotation from head to tail entity. The scoring function is given as

$$f_r(h,t) = -\,||\,\mathbf{h} \circ \mathbf{r} - \mathbf{t}\,||_1^2$$

where $|\mathbf{r_i}| = 1$ and $\circ$ is the element-wise product. The absolute value for a complex number is $|a + i\,b| = \sqrt{a^2 + b^2}$. Since $r_i = cos(\theta) + i\,sin(\theta)$, $|r_i| = \sqrt{cos^2(\theta) + sin^2(\theta)} = 1$.

4.1.5 SimplE

SimplE [21] is designed so that it can take advantages of inverse of relations using an inverse relation vector r^{-1}. Given $\mathbf{h}$ as the head entity embeddings and $\mathbf{t}$ as the tail entity embeddings, and h and t are the head and tail entities whose embeddings we wish to access. The scoring function is defined as

$$f_r(h,t) = \frac{1}{2} * (\langle \mathbf{h_h}, \mathbf{r}, \mathbf{t_t} \rangle + \langle \mathbf{h_t}, \mathbf{r^{-1}}, \mathbf{t_h} \rangle)$$

During evaluation and prediction process, SimplE ignores the r^{-1} and uses the scoring function

$$f_r(h,t) = \langle \mathbf{h_h}, \mathbf{r}, \mathbf{t_t} \rangle$$

4.1.6 TransD

TransD [19] is a translation-based embedding approach that introduces the concept that entity and relation embeddings are no longer represented in the same space. Entity embeddings are represented in space $\mathbb{R}^k$ and relation embeddings are represented in space $\mathbb{R}^d$ where $k \geq d$. TransD also introduces additional embeddings $\mathbf{w_h}, \mathbf{w_t} \in \mathbb{R}^k$ and $\mathbf{w_r} \in \mathbb{R}^{\mathbf{d}}$. I is the identity matrix. The scoring function for TransD is defined as

$$f_r(h,t) = -||h_\perp + \mathbf{r} - t_\perp||$$

$$h_\perp = (\mathbf{w_r w_h^T} + I^{d \times k})\,\mathbf{h}$$

$$t_\perp = (\mathbf{w_r}\mathbf{w_t^T} + I^{d\times k})\,\mathbf{t}$$

TransD imposes contraints like $||\mathbf{h}||_2 \leq 1$, $||\mathbf{t}||_2 \leq 1$, $||\mathbf{r}||_2 \leq 1$, $||h_\perp||_2 \leq 1$ and $||t_\perp||_2 \leq 1$

4.1.7 TransE

TransE [9] is a representative relational distance model. All the embeddings, entities and relations are both represented in the same space $\mathbb{R}^d$ where d is the dimension of the embedding. Given a triple (head, relation, tail), transE imposes the constraint that $h + r \approx t$. The scoring function for TransE is defined as

$$f_r(h,t) = -||\mathbf{h} + \mathbf{r} - \mathbf{t}||_{1/2}$$

TransE enforces additional constraints $||\mathbf{h}||_2 = 1$ and $||\mathbf{t}||_2 = 1$.

4.1.8 TransH

TransH [35] allows entities to have different representations for different relations by creating an additional embedding $\mathbf{w_r} \in \mathbb{R}^d$. This is done by projecting entities onto a hyperplane specific to the relation r and with normal vector $\mathbf{w_r}$.

$$f_r(h,t) = -||h_\perp + \mathbf{r} - t_\perp||_2^2$$

$$h_\perp = \mathbf{h} - \mathbf{w_r^T h w_r}$$

$$t_\perp = \mathbf{t} - \mathbf{w_r^T t w_r}$$

TransH imposes additional constraints $||\mathbf{h}||_2 \leq 1$, $||\mathbf{t}||_2 \leq 1$ and $||\mathbf{w_r}|| = 1$.

4.2 Loss functions

This section describes the various loss functions used in the process of training the embeddings. Given triple (h, r, t), $f_r(h,t)$ is the scoring function for the triple and returns a score. When $f_r(h,t)$ is calculated, the score is calculated using the embeddings corresponding to h, r and t.

4.2.1 Logistic Loss [34]

$$min \sum_{\tau \in \mathbb{D}^+ \cup \mathbb{D}^-} log(1 + exp(-y_{hrt}\, f_r(h,t)))$$

$\tau = (h,r,t)$ and $\mathbb{D}^+$ and $\mathbb{D}^-$ are the set of positive and negative triples. $y_{hrt} = +1$ if $\tau \in \mathbb{D}^+$ and $y_{hrt} = -1$ if $\tau \in \mathbb{D}^-$.

4.2.2 Margin Loss [34]

$$min \sum_{\tau^+ \in \mathbb{D}^+} \sum_{\tau^- \in \mathbb{D}^-} max(0, \gamma - f_r(h,t) + f_r(h',t'))$$

$\tau^+ = (h,r,t) \in \mathbb{D}^+$, $\tau^- = (h',r,t') \in \mathbb{D}^-$ and $\mathbb{D}^+$ and $\mathbb{D}^-$ are the set of positive and negative triples. γ is the margin parameter.

4.2.3 Sigmoid Loss [32]

$$min \sum_{\tau^+ \in \mathbb{D}^+} \sum_{\tau^- \in \mathbb{D}^-} sigmoid(-y_{hrt}\, f_r(h,t))$$

$\tau^+ = (h,r,t) \in \mathbb{D}^+$, $\tau^- = (h',r,t') \in \mathbb{D}^-$ and $\mathbb{D}^+$ and $\mathbb{D}^-$ are the set of positive and negative triples. $sigmoid(x) = \frac{1}{1+e^{-x}}$

4.2.4 Softplus Loss [32]

$$min \sum_{\tau^+ \in \mathbb{D}^+} \sum_{\tau^- \in \mathbb{D}^-} softplus(-y_{hrt}\, f_r(h,t))$$

$\tau^+ = (h,r,t) \in \mathbb{D}^+$, $\tau^- = (h',r,t') \in \mathbb{D}^-$ and $\mathbb{D}^+$ and $\mathbb{D}^-$ are the set of positive and negative triples. $softplus(x) = ln(1 + exp(x))$

4.2.5 Absolute Margin Loss [16]

$$min \sum_{\tau^+ \in D^+} \sum_{\tau^- \in D^-} f_r(h,t) + max(0, \gamma - f_r(h',t'))$$

$\tau^+ = (h,r,t) \in \mathbb{D}^+$, $\tau^- = (h',r,t') \in \mathbb{D}^-$ and $\mathbb{D}^+$ and $\mathbb{D}^-$ are the set of positive and negative triples. γ is the margin parameter

4.2.6 Negative Sampling Loss [31]

$$min \sum_{\tau^+ \in D^+} \sum_{\tau^- \in D^-} -log(sigmoid(\gamma - f_r(h,t)) - log(sigmoid(f_r(h',t') - \gamma)$$

$\tau^+ = (h,r,t) \in \mathbb{D}^+$, $\tau^- = (h',r,t') \in \mathbb{D}^+$ and $\mathbb{D}^+$ and $\mathbb{D}^-$ are the set of positive and negative triples.

4.3 Implementation

8 knowledge graph embedding algorithms have been implemented in *aKGE*: TransD, TransE, TransH, HolE, RotatE, DistMult, SimplE, ComplEx. The datasets used to train these models are illustrated in table 6.1. *aKGE* features a rank-based evaluator where the realistic ranks are taken into account when computing the final metrics. The library has been implemented using Python and PyTorch and the implementations of the embedding algorithms have been kept as close to the original papers as possible. Through our findings, we have discovered that current embedding libraries have minor implementation variations that do not allow for fair comparison between the same embedding algorithm in two different libraries. Due to these changes, it is also harder to reproduce the same results multiple times. Table 4.2 describes the implementation of the embeddings algorithms available in aKGE. We have provided this table so that it is easier to replicate our embedding algorithms and results. This also provides clarity on the approaches we have taken when defining the scoring functions and any implementational differences from OpenKE [17] for each model.

Similar to OpenKE and TorchKGE, normalization of parameters takes place in the differentiation process, while AmpliGraph, LibKGE and PyKEEN perform normalizations outside the differentation process at the beginning of each epoch. In PyKEEN the parameters are normalized in place, i.e., they are not part of the automated differentiation process. In the original paper, TransE proposes to normalize the parameters as presented in section 4.1.7, however, OpenKE also normalizes the relation parameter: $\|\mathbf{r}\|_2^2 = 1$, which the original paper does not propose. The rest of the frameworks have the same normalizations as presented in 4.1.7. Similar to TransE, in the implementation of TransH, OpenKE normalizes the $\mathbf{w_r}$ parameter twice rather, when the original paper proposes only a single normalization. aKGE, PyKeen

and TorchKGE normalize the parameters as described in the table; LibKGE normalizes $\mathbf{w_r}$ twice but misses out on normalizing any of the other parameters whereas AmpliGraph does not implement TransH. For HolE, the normalizations from the original paper are missing from OpenKE. While aKGE normalizes the parameters as described in section 4.1.3, PyKeen and TorchKGE only normalize entity embeddings. LibKGE does not implement HolE. Regarding TransD, OpenKE and TorchKGE do not normalize the transfers $\|\mathbf{h}_\perp\|_2$ and $\|\mathbf{t}_\perp\|_2$. aKGE and PyKeen normalize the parameters as described section 4.1.6; the other frameworks do not implement TransD.

When we compare algorithms like SimplE, the difference in implementations are much more apparent. The scoring function in table 4.2 is the scoring function mentioned in the original paper containing the SimplE algorithm [21]. But in the implementation of SimplE in OpenKE, the function is defined as:

$$f_r(h,t) = \frac{1}{2} * (< \mathbf{h_h}, \mathbf{r}, \mathbf{t_t} > + < \mathbf{h_h}, \mathbf{r^{-1}}, \mathbf{t_t} >)$$

These changes in implementation make it hard to compare results as we are no longer comparing the same models and also make it difficult to reproduce results.

Table 4.2: Scoring Functions and their corresponding PyTorch Implementations

Model	**Scoring Function**	**PyTorch Implementation**
ComplEx	$Re(\langle \mathbf{r}, \mathbf{h}, \bar{\mathbf{t}} \rangle)$	$sum^R(r(h) * r(r) * r(t) + i(h) * r(r) * i(t) + r(h) * i(r) * i(t) - i(h) * i(r) * r(t))$
DistMult	$\sum \mathbf{h} \circ \mathbf{r} \circ \mathbf{t}$	$sum^R(h * r * t)$
HolE	$\sum \mathbf{r}^T \left(\mathcal{F}^{-1}(\overline{\mathcal{F}(\mathbf{h})} \circ \mathcal{F}(\mathbf{t}))\right)$	$sum^R(r * ift^R(conj(ft^R(h)) * ft^R(t)))$
RotatE	$-\|\mathbf{h}\,\mathbf{r} - \mathbf{t}\|_2^2$	$-pow(norm^R(h * stack^R((cos(r), sin(r))) - t, p{=}2))$
SimplE	$\frac{1}{2}(\sum(\mathbf{h_h} \circ \mathbf{r} \circ \mathbf{t_t}) + \sum(\mathbf{h_t} \circ \mathbf{r^{-1}} \circ \mathbf{t_h}))$	$1/2 * (sum^R(hh * r * tt) + sum^R(ht * ri * th))$
TransD	$-\|(\mathbf{w_r}\,\mathbf{w_h}^T + I)\,\mathbf{h} + \mathbf{r} - (\mathbf{w_r}\,\mathbf{w_t}^T + I)\,\mathbf{t}\|_{1/2}^2$	$-pow(norm^R(sf(h, wh, wr) + r - sf(t, wt, wr), p{=}1/2), 2)$
TransE	$-\|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_{1/2}$	$-norm^R(h + r - t, p{=}1/2)$
TransH	$-\|(\mathbf{h} - \mathbf{w_r}^T\,\mathbf{h}\,\mathbf{w_r}) + \mathbf{r} - (\mathbf{t} - \mathbf{w_r}^T\,\mathbf{t}\,\mathbf{w_r})\|_{1/2}^2$	$-pow(norm^R((h - sum^R(h * wr) * wr) + r - (t - sum^R(t * wr) * wr), p{=}1/2), 2)$

Chapter 5

Explainability

5.1 Materialising Triples

During the evaluation process, there are two ranks calculated. When the head entity of a triple in test is corrupted, (h', r, t), we calculate r_h, which is rank with respect to corrupted head entities. When the tail entity is corrupted, (h, r, t'), we calculate r_t which is rank with respect to corrupted tail entities. Since (h, r, t) is the positive triple in test, if for a triple (h', r, t), $f(h', r, t) \leq f(h, r, t)$, then we say that the model has deemed the corrupted triple as plausible. Let N_h be the set of triples where the head entity is corrupted and let N_t be the set of triples where all the tail entity are corrupted. We define M as:

$$M = \bigcup_{(h,r,t) \in Test} N_{h\leq} \cup N_{t\leq}$$

M is dependent on the model under evaluation and thus generated different triples for different models.

When materialising M, there are several factors. The contents of M vary based on the corruption strategy being used [5]. The most common strategy used is Local Closed World Assumption (LCWA) [9]. Here the strategy is to generate all corrupted head and tail entities for each positive triple in test.

During the corruption process, it is possible to generate the same triple multiple times. Assuming three triple (h,r,t), (h,r,t') and (h',r,t). When corrupting t and t', the same triple will be generated because h and r are the same. Similarly, when corrupting h and h', the same triples will be generated since

r and t are the same. Even though the same triple may be generated multiple times, it is possible that each triple has different ranks during the evaluation process. This is due to the fact that the rank is dependent on the score of the positive triple under evaluation. For example, consider three triples t1, t2 and t3 with scores 0.2, 0.4 and 0.6. Let t2 be the corrupted triple that appears during the evaluations of t1 and t3 and let the triple with the lower score be considered the best. When t1 is evaluated, $f(t1) < f(t2)$ and therefore t1 will be ranked higher. But when t3 is being evaluated, $f(t2) < f(t3)$.

When creating M, we can choose to add all negative triples materialised in M or we can choose to add only those negative triples that have been materialised a given number of times. We can also choose to include the positive triples in test along with the negative triples synthesized. Since these positive triples are materialised only once, they have to be treated seperately if there is any filtering of triples based on counts.

Finally, we define a random model as a model where the rank of a given triple is equal to half the total number of corrupted triples [7]. Given positive triple (h,r,t),

$$r_h^e(h, r, t) = \frac{|N_h| + 1}{2}$$

$$r_t^e(h, r, t) = \frac{|N_t| + 1}{2}$$

During the creation of M, we have two options. We can choose to discard all the negatives of a positive triple where the positive triple is ranked worse than random, i.e., $r_h > r_h^e$ and $r_t > r_t^e$. This is done so that we can prevent M from having too many negative triples. We want to also discard these triples as we do not want to take the triples into account whose behavior is worse than that of a random ranking model.

5.2 Explaining Link Prediction

From M we wish to mine Horn Rules using AMIE [14]. Since AMIE only produces rules with one head atom, we can explain the rules predicate by predicate. For each predicate, we may have zero, one or many rules. In the case there are many rules, we need to have a method to select the rules. We can combine the rules by disunction, but this will lead to recomputing the coverage and confidence metrics for the rule. This may also lead to overlap

between the rules. Instead, we propose a selection criteria that picks the rule with a combination of high coverage and high PCA confidence. We use the head coverage of a rule as it determines how well a rule covers the triples in M and we use PCA Confidence as it has been shown that PCA provides better results than standard confidence [14]. Let $\mathcal{B}$ be the body of a rule and

Table 5.1: Example for Support, Head Coverage, Standard Confidence and PCA Confidence

Capital	Country
Capital(Paris, France)	Country(Paris, France)
Capital(Washington D.C., USA)	Country(Washington D.C., USA)
Capital(New Delhi, India)	Country(Rochester, USA)

$h(X, Y)$ be the head of a rule. Therefore, a rule is defined as $\mathcal{B} \Rightarrow h(X, Y)$. Let $inst(\mathcal{B} \Rightarrow h(X, Y))$ indicate the set of triple that come from instantiating the rule over the knowledge graph. Support of a rule is defined as

$$support(\mathcal{B} \Rightarrow h(X, Y)) = |\{(x, y)|(x, y) \in inst(\mathcal{B} \wedge h(X, Y))\}|$$

Support is an absolute number and therefore does not have much significance for a rule by itself. From table 5.1, $support(Capital(A, B) \Rightarrow Country(A, B))$ $= 2$ as there are two pairs of subjects and object that fit the rule when instanitated.
Head coverage of a rule is the ratio of triples that are covered by the rule.

$$hc(\mathcal{B} \Rightarrow h(X, Y)) = \frac{support(\mathcal{B} \Rightarrow h(X, Y))}{|inst(h(X, Y))|}$$

From table 5.1, $hc(Capital(A, B) \Rightarrow Country(A, B)) = 2/3$ since support of the rule is 2, and there are 3 instantiations for the head atom country.
Standard confidence of a rule is defined as the ratio of facts derived from the rule that are true in the knowledge graph

$$conf(\mathcal{B} \Rightarrow h(X, Y)) = \frac{\mathcal{B} \Rightarrow h(X, Y)}{|inst(\mathcal{B})|}$$

From table 5.1, $conf(Capital(A, B) \Rightarrow Country(A, B)) = 2/3$ since the support is 2 and there are 3 instantiations for the body atom.

Standard confidence considers triples that aren't in the graph as negatives. Therefore we defined PCA Confidence as:

$$pca(\mathcal{B} \Rightarrow h(X,Y)) = \frac{\mathcal{B} \Rightarrow h(X,Y)}{\mathcal{B} \Rightarrow h(X,Y) + |inst(h(X,Y'))|}$$

where $Y' \neq Y$. From table 5.1, $pca(Capital(A,B) \Rightarrow Country(A,B)) = 2$ since support is 2 and there is no Y' for Paris or Washington.

Given a rule $\mathcal{B} \Rightarrow h(X,Y)$ as R, we define selection critera $s_\beta(R)$ as:

$$s_\beta(R) = \frac{(1+\beta^2)\, pca(R)\, hc(R)}{\beta^2\, pca(R) + hc(R)}$$

where hc(R) is the head coverage of rule R and PCA(R) is the PCA Confidence of rule R. β is the paramter used to specify the importance of PCA over hc. $\beta \geq 1$ would imply we want give more importance to confidence and $\beta \leq 1$ implies we want to give more importance to coverage. $\beta = 1$ implies we give both the parameters equal importance. The value of s_β lies between 0 and 1, with 1 being the best and 0 being the worst.

Now that we have the s_β scores for each rule, for each predicate, we choose the rule with the highest s_β score. If there are two rules with the same s_β score, then we can choose any one of them. Given s_β scores for each predicate and P is the total number of predicates, we calculate the aggregated mean

$$\bar{s}_\beta = \frac{\sum_{p \in P} s_\beta(p)}{|P|}$$

Since there are cases where zero rules can be generated for a predicate, this can lead to $\bar{s}_\beta$ being skewed to either sides. Therefore we design a weighted arithematic mean where each predicate is weighed by the number of negative triples materialised for it.

$$\hat{s}_\beta = \frac{\sum_{p \in P} |T(M,p)| * s_\beta(p)}{\sum_{p \in P} |T(M,p)|}$$

where $T(M,p)$ is the number of triples in M for predicate p. If a predicate has a low s_β but a large number of triples in M, this is reflected in s_β.

Chapter 6

Experiments

This chapter contains the experiments done for aKGE and explainability. For aKGE, we provide the accuracy metrics calculated on some of the commonly used datasets in link prediction. For explainability, we provide the explainability metrics for 5 embedding approaches and also present a case study where the rules generated for the WordNet and FreeBase datasets are compared.

6.1 AugmentedKGE

6.1.1 Datasets

Table 6.1 describes the data that is commonly used in link prediction. Each dataset is split into training, validation and test sets. FB13, FB15K and FB15K-237 were extracted from Freebase [8]. Among the Freebase datasets, FB13 has the largest number of entities in training and FB15K has the largest number of triples in training. In both validations and test datasets, FB15K has the largest number of triples and entities. For FB15K, there are about 1100 relations missing from validation and test splits. For FB15K-237 there are only about 110 relations missing. WN11, WN18 and WN18RR were extracted from WordNet [23]. Among the WordNet datasets, WN18 has the largest number of entities, relations and triples among all the three splits. Unlike the Freebase datasets, the validation and test splits for WordNet datasets only have 1-2 relations missing. NELL-995 was extracted from NELL[24]. Although NELL-995 contains a large number of entities, it contains lesser triples than all three

Table 6.1: Analysis of the datasets and splits typically used to evaluate link prediction. $|E|$, $|P|$ and $|T|$ stand for the total number of entities, predicates and triples, respectively. $|E_{NT}|$ is the number of entities that are not present in the training split.

	Training			Validation			
	$\|E\|$	$\|P\|$	$\|T\|$	$\|E\|$	$\|E_{NT}\|$	$\|P\|$	$\|T\|$
FB13	75,043	13	336,784	5,667	0	13	4,541
FB15K	14,951	1,345	518,123	11,359	0	233	36,983
FB15K237	14,541	237	270,153	9,750	0	126	19,949
NELL-995	75,492	200	135,339	8,063	0	121	9,409
WN11	38,551	11	118,973	3,547	0	11	2,224
WN18	40,943	18	145,017	4,743	0	17	3,210
WN18RR	40,943	11	88,912	3,360	0	10	2,044
YAGO3-10	123,182	37	1,043,459	24,341	0	30	22,781

	Test			
	$\|E\|$	$\|E_{NT}\|$	$\|P\|$	$\|T\|$
FB13	5,707	0	13	4,548
FB15K	11,389	0	233	37,107
FB15K-237	9,775	0	126	20,014
NELL-995	8,053	0	121	9,465
WN11	3,576	0	11	2,232
WN18	4,724	0	17	3,215
WN18RR	3,339	0	10	2,047
YAGO3-10	24,441	0	30	22,800

Freebase datasets. Across the three splits, it can be observed that there are 79 relations missing from the validation and test splits. YAGO [30] is the largest dataset. It contains more than 1 million triples in training, making it very time consuming process to train embeddings on YAGO. Due to the size of YAGO, we also ran into several out of memory issues.

FB15K and WN18 contain relational anomalies like near-same predicates and near-inverse predicates [3] and these were removed in WN18RR and FB15K-237. From previous works, we know that these anomalies falsly inflate the accuracy of the embedding algorithms and must be accounted for.

6.1.2 Setup

We use the following intervals for the parameters. Learning rate of SGD is between $[1e-10, 1]$. Normalisation has values 1 and 2 where 1 is the manhattan norm and 2 is the euclidean norm. Number of negatives generated for a positive triples is between [1, 50]. For margin based models, the margin is between [0.01, 10]. The embedding dimensions are between [50, 250]. The number of batches the data is split into is between [1, 250]. The weight decay is between $[1e-10, 0.1]$ and the momentum is between [0.5, 1]. There are a total 2500 epochs that each model is run with a validation step happening every 100 epochs. We use early stopping where we evaluate the value of WMR over the validation set. To select the best hyperparameter values, we use the Ax platform [1]. While training, we employ the LCWA corruption strategy [9].

6.1.3 Results

Table 6.2: Global accuracy results for the Freebase-based datasets

	FB13'			FB15K'			FB15K-237'		
	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$
ComplEx	.743	.9984	.9986	.9843	.9989	.9989	.9603	.9985	.9986
DistMult	.7629	.9989	.9991	.8175	.9955	.9955	.8203	.9954	.9955
HolE	.7267	.9985	.9988	.8925	.9959	.9959	.8931	.9964	.9965
RotatE	.5872	.9961	.9967	.6146	.9917	.9917	.6091	.9925	.9927
SimplE	.5785	.9967	.9972	.0071	.986	.9861	.3391	.9909	.991
TransD	.2586	.994	.995	.4033	.9884	.9885	.4837	.9892	.9894
TransE	**.8991**	**.9992**	**.9994**	**.9947**	**.9996**	**.9996**	.9779	.9991	.9991
TransH	.8592	.9988	.999	.9893	.9992	.9992	**.9815**	**.9991**	**.9991**
AMIE	.1353	.7362	.7451	.2736	.9087	.9088	.1579	.7081	.7081

When comparing AMR, GMR and WMR values we can see a large difference between AMR and GMR and a very small difference between GMR

[1] https://ax.dev/

Table 6.3: Global accuracy results for the WordNet-based datasets

	WN11'			WN18'			WN18RR'		
	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$
ComplEx	.819	.9984	.9984	.9152	.9995	.9995	.3754	.9967	.9967
DistMult	.2245	.9917	.9917	.5231	.9966	.9966	.4768	.9974	.9974
HolE	.6172	.9962	.9962	.937	.9998	.9998	.5607	.9952	.9952
RotatE	.237	.9924	.9924	.382	.9934	.9934	.4042	.9935	.9935
SimplE	.4857	.9948	.9948	.2786	.9941	.9941	.3408	.996	.996
TransD	.2075	.9919	.9919	.2058	.9921	.9921	.1987	.9921	.9921
TransE	**.9661**	**.9998**	**.9998**	**.9849**	.9999	.9999	**.9272**	**.9996**	**.9996**
TransH	.9434	.9997	.9997	.9733	.9997	.9997	.8848	.9995	.9995
AMIE	.7678	.9994	.9994	.8908	**.9999**	**.9999**	.3674	.9739	.9739

and WMR values. The difference between AMR and GMR is due to the fact that we have a combination of low and high ranks and this causes AMR to be different. The differences in ranks is not penalised as much with GMR and WMR. When comparing the performance of all the models to AMIE which we use as the baseline, we can see that a large number of models perform better than AMIE. After checking the results of AMIE, we can conclude that this is due to the fact that AMIE does not generate many triples in test from rules generated with training and validation datasets [32]. Table 6.2 are the accuracy results for the Freebase datasets. We can see that the translational based models have the best accuracy for Freebase datasets. TransE and TransH consistently provide the highest AMR, GMR and WMR values. The other translational-based model, TransD, performs poorly when compared to TransE and TransH. From Tables 6.5 and 6.6, TransE and TransH have significantly low ties and ranks, especially for FB15K and FB15K-237. Among the other models, DistMult achieves the highest accuracy for FB13, ComplEx achieves the highest accuracy for FB15K and FB15K-237. Comparing the results of FB15K vs FB15K-237, we can see that AMIE sees a significant drop in accuracy, whereas the difference in accuracies for the other models are much less significant. HolE, RotatE and SimplE achieve higher WMR values for FB15K-237. This is also evident when looking at the ties and the ranks below for every model in Tables 6.5 and 6.6.

Table 6.4: Global accuracy results for NELL-995' and YAGO3-10'

	NELL-995'			YAGO3-10'		
	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$
ComplEx	.8207	.9993	.9993	.897	.999	.999
DistMult	.8662	.9996	.9996	.6253	.9978	.9979
HolE	.8342	.9987	.9987	.8562	.9986	.9986
RotatE	.7332	.9979	.9979	.459	.9973	.9974
SimplE	.4953	.9974	.9974	.3894	.9962	.9963
TransD	.3509	.9949	.9949	.401	.9961	.9961
TransE	**.9651**	**.9998**	**.9998**	**.9852**	**.9998**	.9998
TransH	.9577	.9997	.9997	.9768	.9997	**.9998**
AMIE	.9320	.9993	.9993	.4856	.9994	.9994

Table 6.3 are the accuracy results for the WordNet based datasets. The performance of AMIE is significantly better on the WordNet datasets than the FreeBase datasets. AMIE even outperforms all the models for WN18. TransE and TransH again achieve the best accuracy scores among all models with ComplEx being the next best model. We can see that between WN18 and WN18RR, the different in GMR and WMR values is not significant. Models like DistMult, RotatE and SimplE achieve better results for WN18RR and this can also be seen from their ranks tied and ranks below in Tables 6.5 and 6.6. RotatE and SimplE see reductions in the percentages of ranks below for WN18RR.

Table 6.4 shows the results for NELL-995 and YAGO. For NELL-995 and YAGO, AMIE has comparable performance to all the other models. Following the same trend, TransE and TransH achieve the highest accuracies closely followed by ComplEx and DistMult.

When comparing the results of aKGE with the standard implementation of OpenKE [32], we observe that all models perform better for aKGE except TransD. The results are the same for FB15K-237, but for FB15K, embeddings for HolE, SimplE and TransH perform worse in aKGE. For WN11, HolE, DistMult and TransD perform worse whereas for WN18 only TransD and TransH perform worse. When comparing aKGE with OpenKE for WN18RR and HolE, we can see that again only TransD performs worse. For NELL-995,

Table 6.5: Percentage of ranks in the test split that were tied and below expected for Freebase

	FB13'		FB15K'		FB15K-237'	
	Ties	Below	Ties	Below	Ties	Below
ComplEx	.0007	.0904	.0	.0014	.0	.0075
DistMult	.0008	.0882	.0209	.0304	.0131	.0343
HolE	.0008	.1039	.0002	.005	.0004	.01
RotatE	.0076	.192	.0024	.1434	.0015	.1523
SimplE	.0009	.1361	.0004	.372	.0009	.2782
TransD	.5	.365	.5	.2406	.5	.1975
TransE	.0044	.049	.0001	.0002	.0003	.0021
TransH	.0482	.073	.0002	.0001	.0005	.0011
AMIE	.8891	.8652	.7451	.7244	.9223	.8378

again only TransD performs worse.

These results indicate our hypothesis that changes in implementations such as normalisations and algorithms, impact the accuracy of the embedding approaches and therefore make the results harder to compare and harder to reproduce. SimplE is the best example of this hypothesis. Across multiple datasets, we can see huge differences in values obtained by OpenKE and aKGE. This can be due to the algorithmic differences detailed in section 4.3.

6.2 Explaining Link Prediction

6.2.1 Setup

We have trained 5 models to evaluate our measures on. These are: TransE [9], TransH [35], HolE [25] , RotatE [31] and DistMult [37]. The local closed world assumption [9] was used to generate negative triples. To train the models, we tried several hyperparamer combinations. We fed these hyperparameters combinations into a bayesian optimizer and chose the combination that has the best accuracy values over the validation split. We have evaluated our selectivity measues on the following datasets: FB13, FB15K, FB15K-237, WN18 and WN18RR. In the following sections, we will also present a case study between WN18 vs WN18RR and FB13 vs FB15K vs FB15K-237.

Table 6.6: Percentage of ranks in the test split that were tied and below expected for WordNet, NELL and YAGO

	WN11'		WN18'		WN18RR'		NELL-995'		YAGO3-10'	
ComplEx	.0	.0589	.0003	.0288	.0007	.2218	.0005	.0568	.0007	.0218
DistMult	.4935	.2623	.0	.172	.0	.1786	.0001	.0443	.4134	.1149
HolE	.0009	.1093	.0002	.0227	.0007	.1517	.0006	.0494	.0016	.0226
RotatE	.0271	.3333	.0407	.2843	.0056	.2584	.0059	.0891	.025	.2502
SimplE	.0	.2009	.0005	.2698	.0002	.2328	.0011	.1775	.0029	.2825
TransD	.5	.3374	.4997	.3257	.5	.3405	.5	.2161	.5	.2404
TransE	.0004	.0103	.0006	.0053	.0046	.0274	.0031	.0117	.0018	.0036
TransH	.0007	.0211	.0006	.0075	.0046	.0464	.0038	.0142	.0024	.0063
AMIE	.4263	.2321	.1244	.1092	.6363	.6326	.8945	.8227	.4845	.2575

Table 6.7: Explainability measures for the evaluated models

	FB13		FB15K		FB15K-237		WN18		WN18RR	
	$\hat{s}_\beta$	$\bar{s}_\beta$	$\hat{s}_\beta$	$\bar{s}_\beta$	$\hat{s}_\beta$	$\bar{s}_\beta$	$\hat{s}_\beta$	$\bar{s}_\beta$	$\hat{s}_\beta$	$\bar{s}_\beta$
DistMult	.113	.151	.545	.102	.603	.340	.264	.237	.420	.354
HolE	.470	.412	.597	.113	.563	.321	.335	.338	.320	.338
RotatE	**.576**	**.603**	.458	.091	.686	.400	.567	.609	**.548**	.540
TransE	.534	.513	**.832**	**.126**	**.874**	**.445**	**.597**	**.750**	.527	**.550**
TransH	.444	.412	.475	.091	.572	.378	.450	.515	.348	.349

When creating M, we chose to use triples that have been materialised more than zero times. We chose to not add positive triples to M because we wish to provide explanations to the negative triples and not the positives. We also choose to remove any negative triples that have been generated where the positive triple has been ranked worse than that of a random model as illustrated in section 5.1.

For the rule mining, we chose to collect all rules that have PCA Confidence ≥ 0.1. When selecting the rules, we use $\beta = 1$ to give equal importance to coverage and confidence.

Table 6.8: Ratio of positive triples that were ranked below the expected rank of a random model

	FB13	FB15K	FB15K-237	WN18	WN18RR
DistMult	.229	.044	.034	.191	.179
HolE	.108	.005	.010	.023	.152
RotatE	.213	.143	.152	.284	.260
TransE	.160	.000	.002	.005	.027
TransH	.163	.008	.001	.007	.336

6.2.2 Explainability

Table 6.7 shows the two explainability measures for the models under consideration. We can see that RotatE and HolE are consistently able to explain more than 40% of negative triples generated. When comparing $\hat{s}_\beta$ and $\bar{s}_\beta$ across all datasets, we observe that there is minor differences across FB13, WN18 and WN18RR. There is a larger difference between the two measures for FB15K and FB15K-237. This is due to the fact that both the FreeBase datasets have multiple predicates missing from the test datasets, and since $\bar{s}_\beta$ is an average, it gets skewed by the absense of multiple triples from test.

Due to the way we collect triples in M, we ignore negative triples generated when a positive triple is ranked worse than random. Table 6.8 shows the percentage of positive triples that are missing from materialisation process. We can see that across all datasets, RotatE disregards more than 14% of all positive triples. We can also see that across all models for FB13, there are more than 10% of triples being disregarded. FB15K and FB15K-237 achieves the lowest percentage of positive triples removed.

Table 6.9 contains the ratio of negatives that are in M to the total number of negatives generated. FB15K-237 has the highest ratios across all models. Both WN18 and WN18RR have low ratios, but WN18RR has the least. TransH for WN18RR has only 6.1% of all the negatives generated. HolE for WN18 only has 2.5% of all the negatives generated in M. From table 6.8 and table 6.9, although WN18 discards only 0.7% of positive triples for TransH, this leads to 51% of all the negative triples being removed from M.

When selecting rules we use their corresponding s_β scores. Table 6.10 shows the number of rules generated. Other than FB15K, RotatE generates the highest number of rules across all the datasets. WN18RR and FB13 has

Table 6.9: Ratio of negative triples in M with respect to all the negative triples ranked above positive triples

	FB13	FB15K	FB15K-237	WN18	WN18RR
DistMult	.313	.475	.541	.180	.125
HolE	.430	.876	.777	.025	.299
RotatE	.304	.340	.316	.183	.218
TransE	.296	.926	.843	.282	.246
TransH	.383	.707	.900	.490	.061

Table 6.10: Number of rules generated

	FB13	FB15K	FB15K-237	WN18	WN18RR
DistMult	124	180,153	32,156	230	164
HolE	222	160,894	18,826	332	215
RotatE	418	95,137	92,508	1,116	290
TransE	184	41,332	27,519	696	189
TransH	316	39,593	20,151	1,011	199

the lowest number of rules generated across all models.

6.2.3 WN18 vs WN18RR

As discussed above, WN18RR is a smaller dataset compared to WN18, and has various predicates from WN18 removed. This presents a problem as the predicates now have different identifiers in the dataset and have to be individually matched for comparison. Between WN18 and WN18RR, we can see that from table 6.8 only HolE and TransH have worse percentages of triples ranked below expected than the other models. An interesting observation from table 6.9, even though HolE has more number of positive triples discarded in WN18RR, there is a significant increase in the ratio of negative triples in M. But the opposite can be observed for TransH, where there is a significant decrease in the ratio of negative triples in M. From table 6.7, only DistMult has an increase in explainability. When comparing the rules generated, from table 6.10, we can see that WN18RR generates lower number of rules than WN18 across all models.

Table 6.11 contains the acronyms used for WordNet based datasets. For

Table 6.11: Acronyms for WN18 and WN18RR

Predicate	Acronym
_member_meronym	mero
_derivationally_related_form	related
_member_of_domain_region	region
_instance_hypernym	hyper
_member_of_domain_usage	usage
_synset_domain_topic_of	topic
_instance_hyponym	hypo
_has_part	part
_verb_group	vgroup
_also_see	see

WN18, TransE generates rules with the best s_β scores and DistMult generates rules with the worst s_β scores. For WN18RR, the same observations do not hold. The best scores are split between TransE and RotatE. DistMult is no longer consistently generating rules with the lowest s_β scores. To obtain tables 6.12 and 6.13, the s_β scores across all 5 models were averaged and the 5 best predicates were chosen. To explain the symantics of each predicate we rely on the definitions given by the Global WordNet Association[2].

[2]https://globalwordnet.github.io/gwadoc/

Table 6.12: Best rules for WN18

Head	TransE	s_β	RotatE	s_β	TransH	s_β	DistMult	s_β	HolE	s_β
mero(A,B)	mero(A,H), mero(H,B)	.905	mero(A,H), usage(H,B)	.587	mero(A,H), mero(H,B)	.635	mero(A,H), vgroup(H,B)	.224	mero(A,H), mero(H,B)	.394
region(A,B)	region(A,H), region(H,B)	.882	region(A,H), region(H,B)	.792	region(B,A)	.573	region(A,H), region(H,B)	.365	region(A,H), region(H,B)	.365
hyper(A,B)	hyper(A,H), hyper(H,B)	.989	hyper(A,H), region(H,B)	.855	hyper(A,H), hyper(H,B)	.760	hyper(A,H), hyper(H,B	.234	hypo(B,A)	.294
usage(A,B)	usage(A,H), region(H,B)	.972	usage(A,H), region(H,B)	.829	usage(A,H), usage(H,B)	.764	usage(A,H), usage(H,B)	.268	usage(A,H), region(H,B)	.499
topic(A,B)	topic(A,H), topic(H,B)	.952	topic(A,H), topic(H,B)	.641	topic(A,H), topic(H,B)	.775	topic(A,H), topic(H,B)	.280	topic(A,H), topic(H,B)	.403

Table 6.13: Best rules for WN18RR

Head	TransE	s_β	RotatE	s_β	TransH	s_β	DistMult	s_β	HolE	s_β
mero(A,B)	mero(A,H), mero(H,B)	.645	usage(H,A), mero(H,B)	.418	related(H,A), mero(H,B)	.434	mero(A,H), mero(H,B)	.495	topic(A,H), mero(H,B)	.297
region(A,B)	region(A,H), region(H,B)	.751	region(A,H), region(H,B)	.736	region(A,H), hyper(B,H)	.436	region(A,H), region(H,B)	.544	region(A,H), region(H,B)	.475
hyper(A,B)	hyper(A,H), hyper(H,B)	.943	hyper(A,H), hyper(H,B)	.819	hyper(A,H), usage(B,H)	.223	hyper(A,H), hyper(H,B)	.427	hyper(A,H), hyper(H,B)	.382
usage(A,B)	usage(A,H), part(B,H)	.572	usage(A,H), region(H,B)	.758	usage(A,H), see(H,B)	.340	usage(A,H), usage(H,B)	.544	usage(A,H), region(H,B)	.565
topic(A,B)	topic(A,H), topic(H,B)	.857	topic(A,H), topic(H,B)	.690	see(H,A), topic(H,B)	.393	topic(A,H), topic(H,B)	.408	topic(A,H), topic(H,B)	.444

A word B is a meronym of a word A if B is a part of A, e.g., finger is a meronym of hand. Meronyms can also be transitive. If word B is a mernoym of A, and word C is a meronym of B, then word C is a meronym of word A. For example, Finger is a meronym of Hand, and Nail is a meronym of finger. Therefore, nail is a meronym of hand. Rules of these kind have high s_β scores in tables 6.12 and 6.13. The rules with other predicates in them have lower s_β scores.

Region predicate between two entities A and B implies that B is a geographical or cultural pointer of A, e.g., Superbowl is a geographical pointer to United State of America. The region predicate can also be transitive. We can see that the rules for the region predicate is the same across all models execpt TransH in WN18.

The hypernym predicate between A and B implies that A is a more general concept than B. Hyponym is the reverse of the hypernym predicate. For example, sports is a hypernym of football and football is a hypnoym of sports. Hypernyms and hyponyms also exhibit the transitive property. We can see that the rules are mostly the same across all models with 3 variations. But these variations also symantically make sense, e.g., the region predicate in TransH for WN18 can be used in conjunction with hypernym. An important observation to note is the one of the variations is a rule with the hyponym predicate. This indicates that the models are learning the symantics of the language.

The predicate Usage has been renamed to Exemplify in the Global WordNet Association. Exemplify means to provide an example of. For example, a car is an example of a vehicle. This predicate also exhibits the transitive property. Using the same example as before, we can say Tesla Model S is an example of a car, therefore the Tesla Model S is an example of a vehicle. The usage predicate has the largest deviation in rules. We can see that there are different kind of rules. One of the predicates generated is the see predicate. According to the definition for the see predicate, it is defined as a loose relation between two entites, and therefore can be used in conjunction with the usage predicate. Another predicate generated is also region. For example, White House is in the region of Washington D.C., and Washington D.C. exemplifies USA, therefore the White House exemplifies USA. We can see that rules containing the region predicate are the rules with the highest s_β scores.

Lastly, we have the topic predicate. Topic is defined as connecting two

Table 6.14: Acronyms for FB13, FB15K and FB15K-237

Predicate	Acronym
producer_type	ptype
place_of_birth	pbirth
place_of_death	pdeath
nominated_for	nomfor
nationality	nation
type_of_union	tunion
influenced_by	infuby
country_of_origin	corigin
friendship/participant	fpart
olympic_athlete_affiliation/country	ocountry

scientifically related concepts. This predicate can also exhibit the transitive property. We can see that the rules generated for this predicate are all the same, except for TransH, where we have the see predicate used in conjunction with topic. From the definition above, we know the see predicate implies loose relation, and therefore can be used along with the topic predicate. We can see that for WN18RR, TransE for the topic predicate has the second highest s_β values.

6.2.4 FB13 vs FB15K vs FB15K-237

Between the three datasets in table 6.8, FB13 has the highester ratio of positive triples ranked worse than random, whereas both FB15K and FB15K-237 have the lowest ratios across all datasets. Only RotatE for both datasets have ≥ 0.05. TransE for FB15K and TransH for FB15K-237 have almost 0% of positive triples discarded indicating that the models are not wrongly ranking these triples as worse than random. From table refnegativesgm, FB15K and FB15K-237 have the highest ratios of negatives in M compared to all other datasets. For TransE and TransH these ratios are ≥ 0.7. RotatE has the lowest ratios among all three datasets. Comparing the measures in table 6.7, all three datasets have similar $\hat{s}_\beta$ scores with TransE having the highest scores in both FB15K and FB15K-237. Other than DistMult, FB13 has the highest $\bar{s}_\beta$ across the three datasets. As mentioned aboved, the $\bar{s}_\beta$ values for FB15K

are the lowest due the number of predicates that have 0 triples in test and 0 rules generated, therefore having their individual $s_\beta = 0$. Unlike WN18 vs WN18RR where we saw a decrease in $\hat{s}_\beta$ scores across WN18RR, we see an increase in $\hat{s}_\beta$ scores for FB15K-237 when compared to FB15.

Rules generated by FB15K and FB15K-237 are mostly of two types: 1. $r1(A, B) \wedge r2(B, C) => r1(A, C)$ and 2. $r1(A, B) \wedge r1(B, C) => r1(A, C)$. In most of the rules of these two types, $B = C$ and therefore the rule satisfies even if it does not make sense. For example, one of the rules generated were $film\_release\_region(A, GreatBritain) \wedge nationality(GreatBritain, GreatBritain) => film\_release\_region(A, GreatBritain)$. To illustrate some rules derived by FB15K and FB15K-237, we will be comparing them to 5 rules from FB13 with the highest average s_β values.

For FB15K, the model HolE has the predicate films in the body of every rule. This seems to be an error by the system as films is not related to any of the predicates under discussion.

For the predicate gender, on average, models perform better on FB15K and FB15K-237 than FB13. Between FB15K and FB15K-237, we can see that only RotatE has a higher s_β score in FB15K-237 while all other models have worse scores for FB15K-237. TransE for both the models covers 100% of the triples. All 3 datasets have atleast one rule of the form $gender(A, H) \wedge gender(H, B) => gender(A, B)$ and this rule has the highest s_β across all datasets.

For the predicate nationality, RotatE, DistMult and HolE perform better on FB15K-237 vs FB15K. FB13 has the highest s_β score for RotatE, while other models perform worse than their FB15K and FB15K-237 counterparts. DistMult on FB13 has the worst performance with $S_\beta = 0.05$. For DistMult in FB15 and FB15K-237, both ocountry and corigin indicate the country that the entities are affiliated to. Again, $nation(A, H) \wedge nation(H, B) => nation(A, B)$ has the highest s_β values in their respective datasets. In FB13, this rule also has the lowest score.

For the predicate place_of_death, models TransE, TransH and RotatE perform better for FB15K-237 than FB15K. Both FB15K and FB15K-237 have higher scores than FB13. The rules generated by the models on FB13 hold meaning when compared to that of FB15K. For models DistMult, FB15 has producer_type in the body. This does not make contextual sense, but this rule does have the highest s_β value amongst the 3 datasets. The transitive rule

Table 6.15: Best rules for FB13

Head	TransE	s_β	TransH	s_β	RotatE	s_β
gender(A,B)	spouse(G,A), gender(G,B)	.521	gender(A,H), institution(H,B)	.497	gender(A,H), gender(H,B)	.825
nation(A,B)	children(A,H), nation(H,B)	.594	nation(A,H), religion(H,B)	.6	nation(A,H), nation(H,B)	.737
pdeath(A,B)	children(A,H), pdeath(H,B)	.571	spouse(G,A), pdeath(G,B)	.473	pdeath(A,H), pdeath(H,B)	.737
pbirth(A,B)	children(A,H), pbirth(H,B)	.578	spouse(G,A), pbirth(G,B)	.459	pbirth(A,H), pbirth(H,B)	.665
children(A,B)	children(A,H), children(H,B)	.564	children(A,H), children(H,B)	.443	children(A,H), pdeath(H,B)	.450

Head	DistMult	s_β	HolE	s_β
gender(A,B)	gender(A,G), gender(G,B)	.253	gender(A,H), gender(H,B)	.574
nation(A,B)	nation(A,H), nation(H,B)	.05	nation(A,H), nation(H,B)	.340
pdeath(A,B)	pbirth(A,B), location(A,B)	.106	parents(A,H), pdeath(H,B)	.427
pbirth(A,B)	pdeath(A,B), location(A,B)	.103	parents(A,H), pbirth(H,B)	.460
children(A,B)	children(A,H), gender(H,B)	.433	children(A,H), gender(H,B)	.28

Table 6.16: Best rules for FB15K

Head	TransE	s_β	TransH	s_β	RotatE	s_β
gender(A,B)	gender(A,H), gender(H,B)	1	influby(A,H), gender(H,B)	.53	gender(A,H), gender(H,B)	.905
nation(A,B)	nation(A,H), nation(H,B)	.998	spouse(G,A), nation(G,B)	.503	nation(A,H), nation(H,B)	.734
pdeath(A,B)	pdeath(A,H), pdeath(H,B)	.812	spouse(G,A), pdeath(G,B)	.537	pdeath(A,H), pdeath(H,B)	.780
pbirth(A,B)	pbirth(A,H), pbirth(H,B)	.998	pbirth(G,B), spouse(G,A)	.678	pbirth(A,H), pbirth(H,B)	.842

Head	DistMult	s_β	HolE	s_β
gender(A,B)	spouse(A,H), gender(H,B)	.754	films(A,H), gender(H,B)	.574
nation(A,B)	nation(A,G), corigin(G,B)	.633	films(A,H), nation(H,B)	.531
pdeath(A,B)	ptype(A,H), pdeath(H,B)	.687	films(A,H), pdeath(H,B)	.728
pbirth(A,B)	ptype(A,H), pbirth(H,B)	.588	films(A,H), pbirth(H,B)	.645

$pdeath(A,H) \wedge pdeath(H,B) => pdeath(A,B)$ has the highest s_β value of 0.962.

For the predicate place_of_birth, a transitive rule appears twice for FB15K and FB15K-237, but only once in FB13. For FB13, the rule generated by model RotatE has the highest selectivity score. TransE on FB15K achieves the highest score for this predicate. Other than TransH, all the other models have higher scores on FB15K than on FB15K-237.

Table 6.17: Best rules for FB15K-237

Head	TransE	s_β	TransH	s_β	Rotate	s_β
gender(A,B)	gender(A,H), gender(H,B)	1	gender(A,H), gender(H,B)	.972	award(A,H), gender(H,B)	.843
nation(A,B)	nation(A,H), nation(H,B)	.997	fpart(G,A), nation(G,B)	.590	nation(G,B), films(G,A)	.810
pdeath(A,B)	pdeath(A,H), pdeath(H,B)	.962	pdeath(A,H), pdeath(H,B)	.820	pdeath(A,H), pdeath(H,B)	.850
pbirth(A,B)	pbirth(A,H), pbirth(H,B)	.925	pbirth(A,H), pbirth(H,B)	.790	state(A,H), pbirth(H,B)	.739

Head	DistMult	s_β	HolE	s_β
gender(A,B)	student(A,H), gender(H,B)	.732	tunion(A,H), gender(H,B)	.523
nation(A,B)	ocountry(A,H), nation(H,B)	.716	nation(G,B), nomfor(G,A)	.676
pdeath(A,B)	nation(A,H), pdeath(H,B)	.558	tunion(A,H), pdeath(H,B)	.671
pbirth(A,B)	pbirth(G,B), nomfor(G,A)	.501	pbirth(G,B), category(G,A)	.602

Chapter 7

Conclusion

This report contains AugmentedKGE, a new knowledge graph embedding library. The library features multiple embedding algorithms, features a rank-based evaluator, multiple corruption strategies and is developed using Python and PyTorch. The results of the library show that the library achieves more accurate models than OpenKE, which was the library under comparison. The results of the individual models are also compared and explained. The strategies used to implement each model are explained in this report along with their codes. This is done to keep the models as close as possible to the original papers containing them but also to improve reproducibility and extensibility of our framework. Although our framework produces better accuracies, it would not mean anything if the results of the embedding models could not be explained. In this report, we also present two newly designed measures to enhance the explainability of the model. These measures are obtained from the results of the link prediction process. We have also provided the best rules generated by each model and analysed the rule, so that the functioning of these embedding models can be explained further. From the results we can observe that the output of models can be explained by a small number of rules.

In the future, we aim to add further functionality to aKGE by adding new models, more corruption strategies, and rewriting sections of the code to improve the speed of the link prediction training and validation process. From our results, we can see that TransD performs poorly when looking at the accuracy measures and the percentages of ranks tied and below expected. We intend on performing an in depth study into these results, and if required,

reimplementation of the algorithm. When generating rules to calculate $\hat{s}_\beta$ and $\bar{s}_\beta$ measures, we are required to derive rules from the negative triples materialised. This rule generation process requires a large amount of space and is also very time consuming. We aim to search for more efficient rule generation algorithms, and if not available, we aim to extend AMIE with the focus being improvement in speed. We also aim to provide the explainability measures for all remaining models and datasets, and also devise new explainability measures that further enhance the explainability of the models.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[3] Farahnaz Akrami, Mohammed Samiul Saeef, Qingheng Zhang, Wei Hu, and Chengkai Li. Realistic re-evaluation of knowledge graph completion methods: An experimental study. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1995–2010, 2020.

[4] Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Sahand Sharifzadeh, Volker Tresp, and Jens Lehmann. Pykeen 1.0: A python library for training and evaluating knowledge graph emebddings. *arXiv preprint arXiv:2007.14175*, 2020.

[5] Iti Bansal, Sudhanshu Tiwari, and Carlos R. Rivero. The impact of negative triple generation strategies and anomalies on knowledge graph completion. In *CIKM*, pages 45–54, 2020.

[6] Max Berrendorf, Evgeniy Faerman, Laurent Vermue, and Volker Tresp. Interpretable and fair comparison of link prediction or entity alignment

methods with adjusted mean rank. *arXiv preprint arXiv:2002.06914*, 2020.

[7] Max Berrendorf, Evgeniy Faerman, Laurent Vermue, and Volker Tresp. Interpretable and fair comparison of link prediction or entity alignment methods with adjusted mean rank. *CoRR*, abs/2002.06914, 2020.

[8] 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.

[9] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems (NIPS)*, pages 1–9, 2013.

[10] Armand Boschin. Torchkge: Knowledge graph embedding in python and pytorch. In *International Workshop on Knowledge Graph: Mining Knowledge Graph for Deep Insights*, Aug 2020.

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

[12] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam Hruschka, and Tom Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 2010.

[13] Luca Costabello, Sumit Pai, Chan Le Van, Rory McGrath, Nicholas McCarthy, and Pedro Tabacof. AmpliGraph: a Library for Representation Learning on Knowledge Graphs, March 2019.

[14] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd international conference on World Wide Web*, pages 413–422, 2013.

[15] Matt Gardner and Tom M. Mitchell. Efficient and expressive knowledge base completion using subgraph feature extraction. In *EMNLP*, pages 1488–1498, 2015.

[16] Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach. *arXiv preprint arXiv:1706.05674*, 2017.

[17] Xu Han, Shulin Cao, Lv Xin, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. Openke: An open toolkit for knowledge embedding. In *Proceedings of EMNLP*, 2018.

[18] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, et al. Knowledge graphs. *arXiv preprint arXiv:2003.02320*, 2020.

[19] Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*, pages 687–696, 2015.

[20] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. *arXiv preprint arXiv:1802.04868*, 2018.

[21] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[22] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *IJCAI*, pages 3137–3143, 2019.

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

[24] Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Bishan Yang, Justin Betteridge, Andrew Carlson, Bhanava Dalvi, Matt Gardner, Bryan Kisiel, et al. Never-ending learning. *Communications of the ACM*, 61(5):103–115, 2018.

[25] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. Holographic embeddings of knowledge graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[26] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Icml*, 2011.

[27] Matteo PALMONARI and Pasquale MINERVINI. Knowledge graph embeddings and explainable ai. *Knowledge Graphs for Explainable Artificial Intelligence: Foundations, Applications and Challenges*, 47:49, 2020.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[29] Andrey Ruschel, Arthur Colombini Gusmão, Gustavo Padilha Polleti, and Fabio Gagliardi Cozman. Explaining completions produced by embeddings of knowledge graphs. In *European Conference on Symbolic and Quantitative Approaches with Uncertainty*, pages 324–335. Springer, 2019.

[30] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706, 2007.

[31] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*, 2019.

[32] Sudhanshu Tiwari, Iti Bansal, and Carlos R. Rivero. Revisiting the evaluation protocol of knowledge graph completion methods for link predictionn. In *TheWebConf*, pages 1–12, 2021.

[33] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In

International Conference on Machine Learning, pages 2071–2080. PMLR, 2016.

[34] 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, 2017.

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

[36] Jason Weston, Antoine Bordes, Oksana Yakhnenko, and Nicolas Usunier. Connecting language and knowledge bases with embedding models for relation extraction. *arXiv preprint arXiv:1307.7973*, 2013.

[37] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[38] Wen Zhang, Bibek Paudel, Wei Zhang, Abraham Bernstein, and Huajun Chen. Interaction embeddings for prediction and explanation in knowledge graphs. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 96–104, 2019.

[39] Zhaocheng Zhu, Shizhen Xu, Meng Qu, and Jian Tang. Graphvite: A high-performance cpu-gpu hybrid system for node embedding. In *The World Wide Web Conference*, pages 2494–2504. ACM, 2019.

Appendices

Appendix A

Notation

The notation followed for the rest of the document is presented. Torch is the reference used when indicating that a function is obtained from the PyTorch library. *torch.nn* is another package in PyTorch and is refered to as *nn*. $\mathbb{R}$ represents the set of Real Numbers and $\mathbb{C}$ represents the set of Complex numbers. A triple in the knowledge graph is represented as (h, r, t) where h is the head entity, t is the tail entity and r is the relation between them. Given a vector or a matrix x, x^T is the transpose of the vector or matrix. Given x and y where x and y are vectors or matrices, xy is the element-wise product between them. The element-wise product is also used in place of the symbol $\circ$. Given two matrices x and y, the matrix multiplication is defined as $x \times t$. $||x||_{1/2}$ is the Manhattan or Euclidean norm of a vector also known as L1 and L2 norms. The manhattan or L1 norm of a vector x is defined as

$$||x||_1 = \sum_{i=0}^{n-1} |x_i|$$

where n is the number of elements in the vector.
The euclidean or L2 norm of a vector x is defined as

$$||x||_2 = \sqrt{\sum_{i=0}^{n-1} x_i^2}$$

When norm is applied to a matrix we can specify which dimension we wish to compute the norm on. This can be either row or column or any other

dimension depending on the shape of the matrix. If no dimension is specified for a matrix, all elements are arranged into a vector and the norm is computed. For all functions defined below, dimension is the shape of the input. For example, if the shape of input x is $n \times m \times r$, then n is the 0th dimension, m is the 1st dimension and r is the 2nd dimension. Conversely, -1 corresponds to r, -2 to m and -3 to n.

PyTorch Function Shorthands

Function : torch.norm
Description : Computes L1 or L2 norm of the given vector or matrix along the desired dimension
Shorthand : *norm*

Function : torch.sum
Description : Computes the sum of the elements in the input along the given dimension. When no dimension is specified, computes sum of all elements in the input.
Shorthand : *sum*

Function : torch.pow
Description : Computes the required power of the given input. Given a vector or a matrix, the power is computed for each individual element.
Shorthand : *pow*

Function : torch.fft.rfft
Description : Computes the Fourier transform along the dimension for the given vector/matrix.
Shorthand : fft

Function : torch.conj
Description : Computes the conjugate of a given complex number as input. Given a vector or matrix of complex numbers, the output is the conjugate of each individual element.
Shorthand : *conj*

Function : torch.fft.irfft
Description : Computes the inverse Fourier transform for the given input/vector/matrix.
Shorthand : $ifft$

Function : torch.mul
Description : Computes the element-wise multiplication for the given vector/-matrix
Shorthand : *mul*

Function : (x).view
Description : Reshapes and creates a duplicate of the x it is called on. x can be a vector or a matrix. A vector can be reshaped to a matrix, a matrix can be reshaped to a matrix of a different size or can be reshaped into a vector.
Shorthand : $(x).view$

Function : torch.cos
Description : Computes the cosine value for the given input. If the input is a vector or matrix, element-wise cosine is computed.
Shorthand : *cos*

Function : torch.sin
Description : Computes the sine value for the given input. If the input is a vector or matrix, element-wise sine is computed.
Shorthand : *sin*

Function : torch.stack
Description : Concatenates a sequence of inputs based on the given dimension. The inputs must have the same size. Dimension 0 would imply concatenation on rows and dimension 1 would imply concatenation on columns.
Shorthand : *stack*

Function : nn.functional.normalize
Description : Computes the L_p normalisation over given inputs and dimension

where p > 0. Given a vector v, the L_p normalisation is defined as

$$v = \frac{v}{||v||_p}$$

Shorthand : nm

Function : (x).repeat
Description : Concatenates the same input x based on the number of concatenations specified and across the specified dimension.
Shorthand : $(x).repeat$

Function : torch.matmul
Description : Computes the matrix multiplication of two matrices
Shorthand : mm

Function : f(input, dim = -1)
Description : Given any function f and input of shape $n1 \times n2....nk$, this function operates on last dimension nk. Since there are many functions throughout this report that operate on this last dimension, we wish to develop a shorthand. Therefore any function of the form $f^R(input) = f(input, dim = -1)$. For example, $sum^R(x) = sum(x, dim = -1)$
Shorthand : $f^R(input)$

Function : x.shape
Description : Returns the shape of a vector or a matrix. Each individual dimension can be accessed in the form of x.shape[i] where i is the shape of the dimension we wish to access.
Shorthand : x.shape

PyTorch Code Shorthands

Vector to matrix reshape

```
1    def vecToMatrix(v, rows, cols):
2        return v.view(v.shape[0],rows,cols)
```

Description : Since the embedding class in PyTorch does not support matrices, we wish to reshape a vector/embedding into a matrix of a given size. Here v is the input vector/embedding we wish to reshape, and rows and cols are the size of the new matrix

Embedding to Complex

```
1    def embToComplex(emb):
2        return emb.view(emb.shape[0], emb.shape[1]/2, -1)
```

Description : The embedding class in PyTorch does not allow for defining embeddings with complex numbers as inputs. To go around this issue, we convert a vector into its complex form. For example, given a vector [1,2,3,4], this function would convert it into [[1,2], [3,4]]. This is the same as $[1 + 2i, 3 + 4i]$. Complex numbers are represented as vectors in PyTorch also because PyTorch currently does not support automatic gradient differentiation on complex numbers.

Transfer

```
1    def transfer(self, e, e_transfer, r_transfer):
2        e = F.normalize(e, p = 2, dim =-1)
3        wr = r_transfer.T
4        wh = e_transfer
5        et = e.T
6        mat = torch.matmul(torch.matmul(wr, wh), et)
7        return F.normalize(mat.T, p=2, dim=-1)
```

Description : Computes the Transfer function for TransD. This function is only used for TransD.

Appendix B

OpenKE Results

Table B.1: Global accuracy results for the Freebase-based datasets where $|M|$ is the model(s) selected and total models trained, $\overline{WMR}_{VA}$ is the adjusted accuracy of the best model in validation (for predicates in Q_2, Q_3 and Q_4), and $\overline{AMR}$, $\overline{GMR}$, $\overline{WMR}$ are the adjusted mean ranks obtained in the test splits using arithmetic, geometric, and weighted geometric means, respectively

	FB13'					FB15K'					FB15K-237'				
	$\|M\|$	$\overline{WMR}_{VA}$	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$	$\|M\|$	$\overline{WMR}_{VA}$	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$	$\|M\|$	$\overline{WMR}_{VA}$	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$
Analogy	1 (13)	.9372	.5298	.9227	.9244	1 (14)	.9918	.9736	.9937	.9936	2 (13)	.9945	.9438	.9948	.9949
ComplEx	1 (14)	.9570	.7657	.9541	.9552	1 (14)	.9876	.9585	.9894	.9894	1 (15)	.9931	.9611	.9935	.9936
DistMult	1 (15)	.9326	.4236	.9148	.9168	1 (15)	.9649	.7558	.9388	.9389	1 (15)	.9701	.8781	.9710	.9710
HolE	2 (13)	.9550	.6754	.9494	.9510	1 (13)	.9955	.9853	.9979	.9979	1 (12)	.9958	.9337	.9962	.9963
RotatE	2 (14)	.9616	.5660	.9476	.9509	1 (14)	.9070	.7349	.9194	.9196	1 (14)	.8662	.5716	.8669	.8673
SimplE	2 (14)	.9351	.4400	.9182	.9207	1 (15)	.9950	.9823	.9961	.9962	2 (15)	.9890	.9543	.9895	.9895
TransD	2 (22)	.9955	.8936	.9962	.9966	2 (22)	.9975	.9865	.9981	.9981	1 (22)	**.9981**	**.9847**	**.9984**	**.9984**
TransE	2 (24)	**.9978**	**.9119**	**.9979**	**.9982**	1 (21)	.9985	**.9942**	.9993	.9993	2 (21)	.9977	.9816	.9980	.9980
TransH	1 (20)	.9973	.8981	.9975	.9977	2 (20)	**.9987**	.9938	**.9995**	**.9995**	1 (21)	**.9981**	.9752	**.9984**	**.9984**
AMIE	1 (1)	-	.1353	.7362	.7451	1 (1)	-	.2736	.9087	.9088	1 (1)	-	.1579	.7081	.7081

Table B.2: Global accuracy results for the WordNet-based datasets

	WN11'					WN18'					WN18RR'				
	$\|M\|$	$\overline{WMR_{VA}}$	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$	$\|M\|$	$\overline{WMR_{VA}}$	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$	$\|M\|$	$\overline{WMR_{VA}}$	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$
Analogy	2 (14)	.8913	.8390	.9765	.9765	1 (15)	.9996	.8903	.9996	.9996	1 (15)	.9862	.4832	.9859	.9859
ComplEx	1 (14)	.9689	.8744	.9972	.9972	3 (15)	.7928	.3849	.7803	.7803	1 (10)	.9943	.6237	.9941	.9941
DistMult	3 (14)	.9788	.8930	.9988	.9988	2 (16)	.8620	.3778	.7807	.7806	4 (14)	.9366	.4249	.9370	.9370
HolE	3 (18)	.9800	.7988	.9988	.9988	2 (25)	.9996	.8796	.9994	.9994	1 (15)	.9966	.7507	.9961	.9961
RotatE	5 (15)	.9501	.6468	.9796	.9796	1 (14)	.9289	.6399	.9629	.9629	4 (14)	.7050	.3157	.7085	.7085
SimplE	1 (15)	.6279	.2201	.6338	.6338	1 (15)	.8508	.2869	.6998	.6998	4 (15)	.9042	.3707	.8985	.8985
TransD	5 (23)	**.9981**	**.9639**	**.9997**	**.9997**	1 (22)	**.9998**	.9726	.9998	.9998	1 (24)	.9988	.9058	.9986	.9986
TransE	3 (21)	.9980	.9502	**.9997**	**.9997**	3 (38)	**.9998**	**.9760**	.9998	.9998	3 (26)	**.9989**	.9458	**.9987**	**.9987**
TransH	3 (21)	**.9981**	.9566	.9996	.9996	1 (22)	.9997	.9601	.9998	.9998	2 (21)	.9987	**.9481**	.9986	.9986
AMIE	1 (1)	-	.7678	.9994	.9994	1 (1)	-	.8908	**.9999**	**.9999**	1 (1)	-	.3674	.9739	.9739

Table B.3: Global accuracy results for NELL-995'

	$\|M\|$	$\overline{WMR}_{VA}$	$\overline{AMR}$	$\overline{GMR}$	$\overline{WMR}$
Analogy	1 (15)	.9821	.7723	.9819	.9819
ComplEx	1 (15)	.9824	.7182	.9825	.9825
DistMult	1 (17)	.9895	.8195	.9893	.9893
HolE	1 (14)	.9957	.8090	.9957	.9957
RotatE	1 (14)	.9680	.6609	.9671	.9671
SimplE	1 (15)	.9941	.8022	.9940	.9940
TransD	1 (22)	**.9992**	.9708	.9993	.9993
TransE	2 (22)	**.9992**	**.9734**	.9993	.9993
TransH	1 (21)	.9991	.9566	**.9996**	**.9996**
AMIE	1 (1)	-	.9320	.9993	.9993

Table B.4: Percentage of ranks in the test split that were tied and below expected

	FB13'		FB15K'		FB15K-237'		WN11'		WN18'		WN18RR'		NELL-995'	
	Ties	Below	Ties	Below	Ties	Below	Ties	Below	Ties	Below	Ties	Below	Ties	Below
Analogy	.0010	.1938	.0001	.0008	.0001	.0126	.0002	.0567	.0	.0513	.0015	.2572	.0003	.0799
ComplEx	.0008	.0633	.0001	.0009	.0	.0047	.0	.0513	.0009	.2663	.0002	.1749	.0005	.1023
DistMult	.0013	.2702	.0003	.0419	.0001	.0151	.0	.0397	.0002	.2631	.0010	.2623	.0005	.0685
HolE	.0008	.1070	.0	.0016	.0001	.0207	.0002	.0914	.0	.0575	.0002	.1060	.0005	.0880
RotatE	.0113	.1877	.0013	.0415	.0017	.1510	.0020	.1456	.0030	.1505	.0046	.3036	.0036	.1415
SimplE	.0007	.2614	.0	.0010	.0001	.0032	.0016	.3591	.0003	.3194	.0002	.3075	.0008	.0876
TransD	.0049	.0269	.0001	.0001	.0001	.0007	.0011	.0108	.0005	.0100	.0012	.0347	.0017	.0086
TransE	.0136	.0222	.0001	.0003	.0001	.0012	.0013	.0164	.0012	.0092	.0015	.0156	.0020	.0070
TransH	.0245	.0292	.0001	.0005	.0002	.0033	.0004	.0141	.0022	.0145	.0029	.0169	.0033	.0277
AMIE	.8891	.8652	.7451	.7244	.9223	.8378	.4263	.2321	.1244	.1092	.6363	.6326	.8945	.8227