Vrije Universiteit Amsterdam

Bachelor Thesis

Re-evaluating Knowledge Graph Embedding Models Performance on Domain Specific Datasets

Author: Victor Ciupec 2611565

1st supervisor: Dr. Peter Bloem
2nd reader: Dr. Erman Acar

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science

October 10, 2020

Abstract

Knowledge graph embedding (KGE) models have become popular solutions for the link prediction problem in knowledge graphs. In this work we compare the performance of several KGE models on domain specific datasets. KGE models learn algebraic representations of the entities and relations in a knowledge graph and use a scoring function to predict and rank new triples in the knowledge graph, thus separating correct from incorrect triples. KGE models vary not only in the parametrization of their relation operators and their scoring function, but also in the choice of hyperparameters, most notably loss function and training strategy. This thesis researches the question on whether KGE models perform differently on domain specific datasets compared to generic ones like Freebase and Wordnet, which are commonly used in most ML benchmarks. In our experiments we used two domain specific datasets (AIFB and MUTAG), both bound by strong ontological information. We aim to explain how hyperparameters affect KGE model performance on domain specific datasets by searching across a large hyperparameter space and then evaluating performance using the entity ranking protocol. We observed that the choice of loss function has by far the highest impact on the performance, followed by the training strategy. With some exceptions, cross entropy loss performed best across the board. Although DistMult outperformed the other models in our study, we observed that, when trained consistently, the relative performance gaps between models are small. Our results are further compared with the ones obtained by [Ruffinelli et al., 2020] on FB15K-237 and WNRR. Both experiments use their Pytorch-based `libkge` framework allowing for equivalent methodologies in both studies and thus commensurate results across the range of configurations. The models performed notably better in our experiment, which can be explained by the domain specific biases in the datasets we used.

1 Introduction

In a knowledge graph (KG) the information is structured as a large network of entities, their semantic types, properties and relationships between entities. In practice, knowledge graphs contain billions of entries and they play a pivotal role in many areas such as social network analysis, recommender systems, drugs discovery, question-answering bioinformatics or semantic searches to only name a few. The most common representation is in the form of triples *(subject, predicate, object)*, indicating that there is a relation expressed by the *predicate* between *subject* and *object*. It can also be viewed as a labelled directed graph, in which each vertex is an entity and each edge as a relation (link) between entities. In practice knowledge graphs contain only observed facts, which means they are not complete, i.e. they do not contain all the true facts (triples or links). Furthermore, they are often noisy, meaning some of the observations have less or no significance in machine learning. For these reasons KG completion methods are becoming increasingly popular and rely on link prediction, i.e. the ability to predict new triples in the KG, which in turn can be applied to a vast array of aforementioned real world problems. To solve the problem of link prediction, several approaches have been published: rule-based, embedding-based or hybrid.

Knowledge graph embedding (KGE) models infer algebraic representations of the entities and relations in a knowledge base, called *embeddings*. These models have successfully emerged as promising methods for link prediction and knowledge base completion. There is a vast number of KGE models published in the recent scientific litera-

ture. Some examples include RESCAL [Nickel et al., 2011], DistMult [Yang et al., 2015], TransE [Bordes et al., 2013], ComplEx [Trouillon et al., 2016], NTN [Socher et al., 2013], ConvE [Dettmers et al., 2018], TuckER [Balazevic et al., 2019], SACN [Shang et al., 2019] etc. The hyperparameters used in the original training have typically been chosen through grid search in a small paramemter space. As argued in [Ruffinelli et al., 2020], the diversity in the model training experimental setup makes it difficult to objectively compare the performance of the models. Therefore the authors proposed a PyTorch library called `libkge` for model training, hyperparameter search in a large space and model reproductions that facilitates consistent benchmarking of the KGE models.

Most of the KGE models in the literature are trained and evaluated using general purpose benchmark datasets such as Freebase and Wordnet. Freebase is a generic knowledge graph encapsulating triples from various domains, such as film, sports, events, geographical locations etc. FB15K-237 [Toutanova and Chen, 2015] contains a restricted number of entities and relations. The sets of triples used in training or evaluation do not include semantic information about entities and relations. The Wordnet dataset is a thesaurus of terms and their semantic relations with other terms, for example synonyms or hyponyms. WNRR [Dettmers et al., 2018] is a restricted subset of Wordnet in which the number of terms have been restricted. Given that these datasets attempt to model a range of domains, their ontological schema does not contain many constraints and they potentially have a degree of sparsity or noise. Thus KGE models can suffer from problems of generalization, reflected in poorer performance.

Compared to generic benchmark datasets such as Freebase or Wordnet, domain specific knowledge graphs do not just contain entities, since many of the nodes are typed according to an ontology, resulting in a richer, more structured data set. This in turn results in more redundancy in expressing facts, less noise and less sparsity than their generic counterparts. Because KGE can suffer from poor performance when the KG is either too sparse or noisy we expect that KGE would perform better on domain specific datasets.

The research question of this thesis is whether using domain specific datasets in training impacts performance and which hyperparameters impact performance the most for these datasets. We investigate how certain properties of domain specific datasets such as ontological structure and redundancy in expressing facts through triples can influence the performance and the selection of hyperparameters.

This thesis is organized as follows: Section 2 offers an overview of the existing KGE models, highlighting the main differences in terms of the entity / relation representation and scoring function. In Section 3 we take a closer look at our domain specific datasets and present our experimental methods and procedures as well as metrics for evaluating the performance of the KGE models, The experiment results follow in Section 4. Section 5 describes related work on which this study is based. We draw conclusion in Section 6.

2 Knowledge Graph Embedding Models

We define a knowledge graph $\mathcal{K} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ as a set of triples (i, k, j), with $i, j \in \mathcal{E}$, $k \in \mathcal{R}$ where $\mathcal{E}$ is a set of entities and $\mathcal{R}$ is a set of relations. For each entity i and relation k, an *embedding model* attaches the *embeddings* $e_i \in \mathbb{R}^d$ and $r_k \in \mathbb{R}^{d_R}$ in a low dimensional vector space. d and d_R are model *hyperparameters* called the *size* of the embedding. The

model learns embeddings that capture the graph structure of the knowledge base, so that new facts can be inferred.

Embedding models work by predicting new triples and ranking them according to a scoring function $s : \mathcal{E} \times \mathcal{R} \times \mathcal{E} \mapsto \mathbb{R}$. The scoring function $s(i, k, j)$ depends on the triple (i, k, j) only through their corresponding embeddings e_i, r_k and e_j. The scoring function measures the plausibility of the triple to be part of the KG, resulting in a separation of the correct triples from the incorrect ones: the triples with high scores are considered more likely to be correct than triples with low score.

Entity and Relation Representations

The embedding model architectures differ in the way entity and relations are represented and combined to generate a prediction: *factorization models* (RESCAL, DistMult), *translational models* (TransE) or *deep learning models* (ConvE or NtN). Existing embedding models in the literature also differ in parametrization of relation operators, which combines the representations of the two entities.

Embedding models represent entities as low dimensional vectors, however different approaches exist with most representing entities as unit vectors. NtN on the other hand represents each entity as an averages of associated word vectors which are initialized with pre-trained vectors such as word2vec [Mikolov et al., 2013].

One of the early KGE models is RESCAL([Nickel et al., 2011]). It uses a tensor factorization model which relies on encoding the knowledge graph into a three-dimensional tensor. Tensors, as opposed to graphical approaches provide greater simplicity as multiple relations can be described as higher order tensors. Furthermore no prior knowledge about the data is needed and latent-components can be inferred.

DistMult ([Yang et al., 2015]) is a simplified version of RESCAL, in which the interaction between the subject and object entities is captured via a diagonal matrix. Due to this simplification, DistMult can only model symmetric relations.

ComplEx([Trouillon et al., 2016]) can be seen as a DistMult extension with complex embeddings: nodes and relations are modeled by multi-dimensional vectors with a real and imaginary part. It is able to handle both symmetric and asymmetric relations.

TransE ([Bordes et al., 2013]) is an energy based model that represents relations as translations in the embedding space: for a triple (i, k, j), e_j is close to e_i plus a vector r_k. The model needs a reduced set of parameters as it learns only one vector for each entity and relation.

Scoring Functions

Relation representation within the various embedding models reflects in the scoring functions. The model architectures have varying scoring functions, however the introduction of new models generally brings new training types, loss functions and new forms of regularization.

The scoring function can generally be expressed as a function of the embeddings:
$s(i, k, j) = f(e_i, r_k, e_j)$ where f can be a fixed or parametrized function. The greater the score of the triple, the more likely it is to be considered true by the model. Depending on whether arbitrary interactions between subject and object embeddings are allowed during

computation of the scoring function, KGE models can be classified as decomposable or monolithic. Decomposable scoring functions allows only element-wise operations and use multiplicative composition of the entity embeddings and the relation type embeddings. Monolithic models allow arbitrary interactions, but in consequence they are more costly to train and use. In this thesis we only consider decomposable models.

RESCAL's scoring function is bilinear in the entity embeddings $s(i,k,j) = e_i^T R_k e_j$ where $R_k \in \mathbb{R}^{d \times d}$ is a matrix constructed from the entries of r_k. DistMult is a constrained version of RESCAL, in which R_k is a diagonal matrix that uses smaller number of relation parameters ($r = d$). ComplEx' scoring function uses the dot product in the complex domain: $s(i,k,j) = Re(< e_i, r_k, \bar{e}_j >)$. TransE is a translation-based model that represents each relation as a single vector that linearly interacts with entity vectors. It uses the negative distance $-\|e_i + r_k - e_j\|$. In ConvE the score is defined by a convolution over 2D shaped embeddings.

3 Methods and Procedures

3.1 Datasets

We trained and evaluated the KGE models in this study using two datasets: AIFB and MUTAGENESIS (in short, MUTAG). Both dataset schemas are very detailed, containing full hierarchies of classes and sub-classes using the RDF model [Klyne and Carroll, 2004]. The RDF model itself has other restrictions, such as no literal entity (e.g. string or numeric) can be the subject of a triple, only the object (i.e. literals are always attributes). In our training and evaluation we augmented the dataset with ontological information, i.e the *type* relations between an instance and its class, for example *(id34, type, Publication)* in AIFB or *(bond54, type, Bond-1)* in MUTAG. Although the special semantics of the *type* and *subClassOf* relations are not considered when training and evaluating the KGE models (one special behavior is that type relations are always transitive), these relations could contribute to clustering of the instances of the same type in the embedding space and influencing the model to associate similar relations to these instances.

The *AIFB* dataset [Bloehdorn and Sure, 2007] is a real world dataset that records the organizational structure and research activity at the Institute of Applied Informatics and Formal Description Methods (AIFB) at the University of Karlsruhe. It is based on the SWRC ontology, which generically models key entities relevant for typical research communities and the relations between them. The AIFB dataset has 8248 entities and 47 relations with semantics specified using OWL [Bechhofer et al., 2004]. The dataset contains approximately 37K relation triples, for example $(Group, publishes, Publication)$ or $(Project, member, Person)$. This dataset is a good fit for our experiments because it is small enough to efficiently train given the available computational power, but large enough to be able to avoid the risk of sensitivity to noise and overfitting. As opposed to FB15K-237 and WNRR that were used in [Ruffinelli et al., 2020], AIFB has a much lower average number of triples per relation (less relation reuse) at 640 in AIFB, 1300 in FB15K-237 and 8300 in WNRR, which can be explained by the argument that AIFB is more domain-specific and avails of a diverse array of relations, whereas FB15K-237 and WNRR are more generic and simply structured. A number of relations are symmetrical, such as *member* relation between $Project$ and $Person$ is the inverse of $worksAtProject$ relation. Moreover,

there is redundancy in expressing facts: for example $(Organization, carriesOut, Project)$ and $(Person, worksAtProject, Project)$ then the triple $(Organization, employs, Person)$ is more likely and we expect the KGE models to be able to learn such relations.

The *MUTAG* dataset [Bühmann et al., 2018] is a medium sized OWL dataset, but larger than AIFB, which describes structure of organic chemical compounds that was mostly used in the original paper to determine their mutagenicity. The dataset contains approx 62K triples across 15k entities and 14 relations. Out of the 14 relations, 11 are attributes, i.e. relations between entities and literals. Examples of triples are $(Compound, hasAtom, Atom)$ or $(Atom, charge, double)$. Unlike AIFB, the triples per relation stands at 2660, suggesting more relation reuse across the dataset and less relation redundancy. There are no symmetrical relations in the MUTAG dataset. However, the only 3 relations that link entity instances form a indirect domain relation redundancy: if $(Compound, hasAtom, Atom)$ and $(Compound, hasBond, Bond)$ then $(Bond, inBond, Atom)$. We expect the KGE models to be able to easily predict any of the three relations given the other two.

	Entities	Relations	Train Size	Test Size	Valid Size
AIFB	8280	47	17534	5845	5632
MUTAG	15260	14	37239	12413	12413

Table 1: *AIFB* and *MUTAG* Dataset statistics.

3.2 Experimental Setup

We use the `libkge` library for training and hyperparameter search. Initially, the AIFB data is in `.rdf` format, and MUTAG data is in `.owl` format. Our script parsed through the triples, cleaned up the empty or malformed entities and output the remaining triples into train, test and validation text files that are accepted by the `libkge` library. We used a train, test and validation split of 60%, 20% and 20% respectively.

To ensure a uniform distribution of entities and relation across the train, test and validation files, the triples were shuffled using before splitting.

During the pre-processing stage, `libkge` assigns indices to each entity and each relation. KGE model implementations use these indices instead of the original entities and relations when learning the embeddings.

To perform hyperparameter search, each model needs to be configured with the search type, the number of trials, the search ranges for each applicable hyperparameter, the validation metric, maximum number of epochs, the split to use etc.

Our entire experimental setup including search configuration, all the trials with their hyperparamenter values and the best model config files are published for further research and analysis [1].

[1] https://github.com/Vixci/bachelor

3.3 The Search Method

`libkge` performs selection of the best model during the search by comparing filtered MRR on validation data amongst all the trials. It implements Quasi-random hyperparameter search via a SOBOL sequence [Sobol, 1967] using the AX framework (`https://ax.dev/`) to identify optimal discrete hyperparameters. SOBOL sequence was used as it fills the hyperparameter space of possibilities more evenly and thus allows for faster convergence. All models were trained for a maximum 400 epochs, validating models every 5 epochs using filtered MRR. Early stopping was implemented with a patience of 50 epochs, meaning the training would stop prematurely if no improvement was found within 50 epochs. For each dataset and model we generated 30 different configurations for each combination of training type and loss function. There were 7 different combinations for RESCAL, DistMult, ConvE and Complex; whereas there were only 2 different combinations for TransE. TransE was an exception as it only supports the negative sampling training type. A Bayesian optimization stage was added to tune the numerical hyperparameters. The Bayesian phase included the best configuration so far for the model and an additional 20 new trials. The final five model configurations for each model architecture were trained, and the best-performing model was selected using filtered MRR as the validation metric (see Section Evaluation Protocols).

3.4 Hyperparameters

Our experiment searches a large hyperparameter space to ensure that no significant hyperparameters were excluded.

We applied all major training strategies (1vsAll, Negative Sampling and KvsAll). The training types differ mainly in the way negative samples are generated and used to calculate the scoring function. In *negative sampling* [Bordes et al., 2013], for a positive (i, k, j) triple, pseudo-negative triples are sampled by mutating either of the entities and optionally excluding those part of the KG. *1vsAll* [Lacroix et al., 2018] omits sampling and takes all triples achieved by changing the subject and object positions for (i, k, j). *KvsAll* [Dettmers et al., 2018], focuses on labeling entire rows from a relation's matrix, for example $(i, k, *)$ or $(*, j, j)$ and the positive or negative label is attached simultaneously depending on whether the resulting triples are in the KG.

We used the following loss functions: cross entropy (log-loss), binary cross entropy (sigmoid log loss) and pairwise margin ranking (between a positive and a negative triple using margin as a hyperparameter).

Reciprocal relations (boolean) [Lacroix et al., 2018] is a hyperparameter in which KGE models assign separate scores to $(i, k, ?)$ and $(?, k, j)$ queries, resulting in the use of two scoring functions s_{sub} and s_{ob}. The use of reciprocal relations dramatically decreases computational cost of ConvE, therefore `libkge` only trains ConvE with reciprocal relations.

Other considered hyperparameters are: regularization technique (whether to apply penalty term with either of L1, L3 or L3 norms for the loss function), embedding sizes (the dimensions of the embeddings in vector space, which can be 128, 256 or 512), batch sizes (number of training samples to use at each epoch before optimizing parameters and can be 128, 512 or 1024), optimizer algorithm for loss minimization (Adam [Kingma and Ba, 2014] or Adagrad [Duchi et al., 2011]) etc. Tables 6 and 7 in the Appendix contains all the hyperparameters for additional information.

According to [Ruffinelli et al., 2020] at the date of their original experiment no prior study had used such a large hyperparameter search space.

3.5 Evaluation Protocols

The KGE models are evaluated in their ability to correctly predict unseen triples using the *entity ranking* protocol. For each (i, k, j) triple in the test data, the model computes all scores for the unseen triples $(?, k, j)$ and $(i, k, ?)$, filtering out the observed matching triples from the training or validation datasets. The answers are ranked in descending order by their score.

For each entity in test triple that is being answered, the rank is calculated (measuring how high it scores across all test triples). *Filtered Mean reciprocal rank (MRR)* calculates the average of the reciprocal of the ranks for all entities. Filtered MRR is a measure of model precision of predicting entities. A complementary metric is *filtered HITS@K*, which measures the proportion of test triples ranking in the top K answers. We are reporting with Hits@10.

3.6 Hardware Setup

For this study we employed a wide range of hardware:

- 6 GPU-equipped compute nodes on the Distributed ASCI Supercomputer (DAS5) [Bal et al., 2016]
- 4 AI-optimised VMs on Google Compute Engine each with: 8 vCPUs, 52GB RAM, 1 Nvidia Tesla Volta100 GPU
- To supplement the computing capacity, we also used 1 desktop workstation with Nvidia GTX 1080 GPU.

We used a Google Cloud Platform Storage Bucket to store the search trials and results. The search and training data produced by the experiments occupied up to 600GB. Hyperparameter search took of approximately 300 hours of computations for both datasets and each search took between 4h and 18h, depending on the model and the hyperparameter or machine configuration.

4 Results

In Table 2 we list the filtered MRR and Hits@10 on test data of the *best models* obtained during the hyperparameter search. In our study, DistMult has outperformed the other models on both domain specific datasets, followed closely by ComplEx. This is not necessarily surprising, as DistMult and ComplEx scored high on standard benchmark datasets in the previous study by [Ruffinelli et al., 2020], whose results are listed in table 2 as well for reference.

Table 3 shows the mean and standard deviation of the MRR and Hits@10 obtained during the validation phase when trained from scratch 5 times. These metrics are from the best hyperparameter configuration for each model and dataset. DistMult was again the best performing model, followed closely by ComplEx.

			RESCAL	TransE	DistMult	ComplEx	ConvE
Our study	*AIFB*	MRR	42.1	46.01	**49.2**	48.7	47.2
		Hits@10	56.9	59.8	**60.1**	60.0	58.7
	MUTAG	MRR	35.63	26.82	**48.07**	38.68	31.63
		Hits@10	46.65	47.39	**60.32**	50.49	47.39
[Ruffinelli et al., 2020]	*FB15K237*	MRR	**35.7**	31.3	34.3	34.8	33.9
		Hits@10	**54.1**	49.7	53.1	53.6	52.1
	WNRR	MRR	46.7	22.8	45.2	**47.5**	44.2
		Hits@10	51.7	52.0	53.1	**54.7**	50.4

Table 2: Performance on test data of the best performing models. Comparison between results using the *AIFB* and *MUTAG* datasets (our experiment) and *FB15K237* and *WNRR* [Ruffinelli et al., 2020]

We also noted that that the MRR and Hits@10 scores are remarkably higher for all models on AIFB than for MUTAG. This can be explained by the fact that AIFB contains symmetrical relations (see section Datasets for example), whereas MUTAG does not. As the experiments in [Akrami et al., 2018] demonstrated, on datasets with symmetrical relations, KGE models tend to perform considerably better, suggesting that in this case the performance excess on AIFB is justified by the fact that the models are good at predicting the inverse relations.

We compare our results on validation and test splits to the results obtained using the same methodology on FB15K-237 and WNRR by [Ruffinelli et al., 2020]. For AIFB our results are notably better compared with FB15K-237 by an average of 30% for MRR but only 10% for Hits@10. This suggests that while predictions on domain specific datasets are consistently better (higher precision), the number of correct predictions in the top 10 (the recall) is more similar with the number obtained for FB15K-237. Compared to WNRR, AIFB performed 12% better on average across both evaluation metrics. On the other hand, the MUTAG dataset didn't have such an increase in performance associated with it, apart from when using the DistMult model where it performed very similarly to AIFB. Across both MUTAG and AIFB datasets, DistMult was the best performing model.

In line with [Ruffinelli et al., 2020], we also observed in our experiment that the relative performance gaps between different best model configurations on any of the datasets were vastly reduced compared to the original publications; this is perhaps due to the consistent

			RESCAL	TransE	DistMult	ComplEx	ConvE
Our study	*AIFB*	MRR	41.6±0.2	45.6±0.1	48.6±0.4	48.6±0.1	47.5±1.0
		Hits@10	56.0±0.2	59.2±0.1	60.0±0.4	59.7±0.2	58.5±0.5
	MUTAG	MRR	36.7±0.8	30.0±0.2	47.5±0.5	38.0±0.4	30.8±0.8
		Hits@10	47.0±0.8	43.9±0.6	59.6±0.5	49.7±0.2	42.7±2.3
[Ruffinelli et al., 2020]	*FB15K237*	MRR	36.1±0.3	31.5±0.1	35.0±0.0	35.3±0.1	34.3±0.1
		Hits@10	54.3±0.5	49.8±0.2	53.5±0.1	53.9±0.2	52.4±0.2
	WNRR	MRR	46.8±0.2	22.6±0.0	45.4±0.1	47.6±0.1	44.3±0.1
		Hits@10	51.8±0.2	51.5±0.1	52.4±0.3	54.1±0.4	50.4±0.2

Table 3: Mean and standard deviation of the performance on validation data of the best performing models. Comparison between results using the *AIFB* and *MUTAG* datasets (our experiment) and *FB15K237* and *WNRR* datasets [Ruffinelli et al., 2020]

methodology during the hyperparameter search, as well as the importance of datasets used in training.

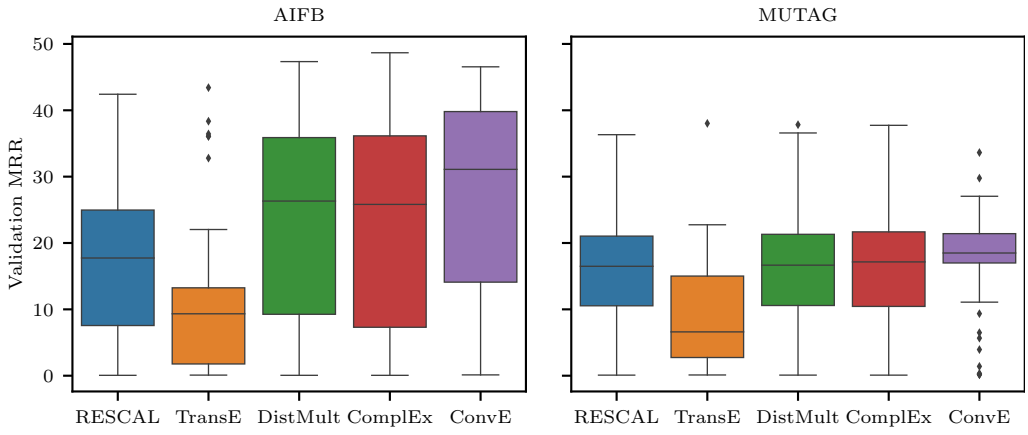

Figure 1: Distribution of filtered MRR (%) on validation data over each of the models during the quasi-random hyperparameter search for AIFB and MUTAG dataset respectively.

Figure 1 shows the distribution of filtered Validation MRR across the quasi-random

hyperparameter configuration space. For each of the models there were 210 configurations, except for TransE, which had 60 (2 train type/loss combinations with 30 trials each). From this we can see that the median ConvE configuration is the best amongst tested models for both datasets, whereas TransE is the worst. From this we can infer that ConvE may be the best model to train if no hyperparameter search is conducted as it's median validation MRR is higher than the other models.

Furthermore it can be seen that the impact of most hyperparameter choice across all models is higher on AIFB than on MUTAG (there is higher variance in the MRR across the space), meaning models are more sensitive to hyperparameter changes. The MUTAG plot shows about the same variance as FB15K-237 in [Ruffinelli et al., 2020].

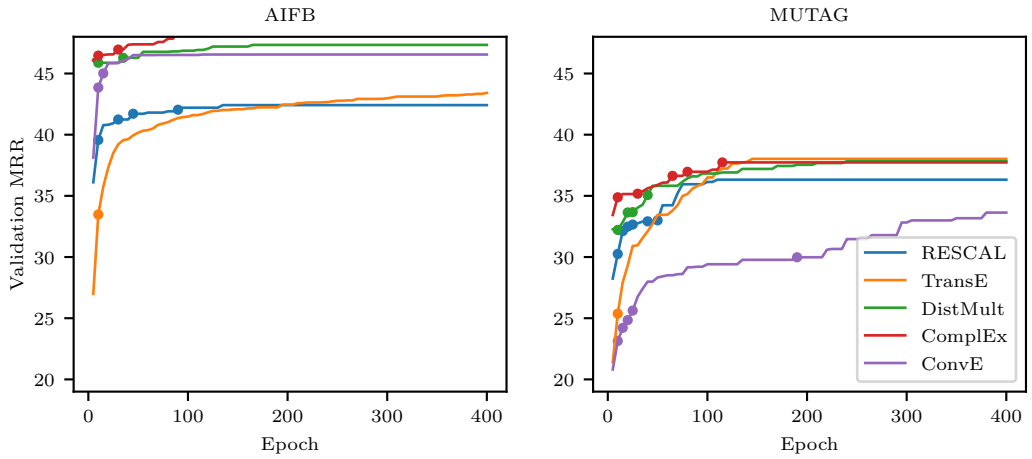

Figure 2: Best filtered MRR % on validation data achieved during quasi-random hyperparameter search, expressed as a function of the number of training epochs. Changes in the corresponding temporary best hyperparameter configuration are marked with a dot.

Figure 2 shows the various model behaviours by plotting the validation MRR across all hyperparameter configurations over the course of the training. The dots indicate a changing best hyperparameter configuration. As can be seen from the graph, many of the models reach a plateau at around 100 epochs showing diminishing returns in performance over the course of the training. Good model performance can be achieved with a smaller number of epochs. This is in line with the results found in [Ruffinelli et al., 2020] showing a similar trend. However longer training can be beneficial to some models in search of the best performance.

4.1 Impact of Hyperparameters

To explore the impact of loss function and training types Figure 3 shows the distribution of Validation MRR for every valid train type/loss function combination for each model. Across all best performing models, Cross Entropy (CE) was the one hyperparameter they had in common, having a significant impact on model performance. This is consistent with [Ruffinelli et al., 2020] where CE performed best on the FB15K-237 and WNRR datasets for all models. In our study the only exception was ConvE, on which binary cross entropy

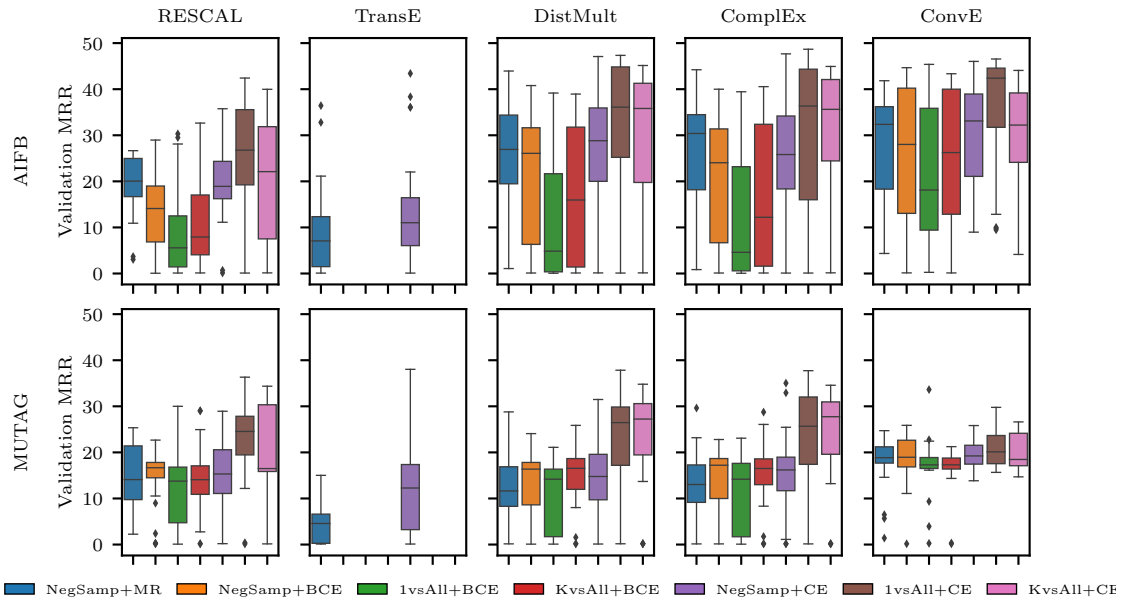

Figure 3: Distribution of filtered MRR (%) on validation data over the train type/ loss function combinations for each of the models during quasi-random hyperparameter search using AIFB and MUTAG dataset respectively.

performed best. This is quite interesting, as BCE loss function was initially employed in a KGE model by [Dettmers et al., 2018] as part of ConvE.

Table 4 shows the selected hyperparameters of the best performing models during the quasi-random search stage, as well as the relative loss in performance if the hyperparameter was not used. As can be seen from the results, the best choice of hyperparameter is often model and dataset specific. This is further reinforced in [Ruffinelli et al., 2020] which surfaced the same pattern.

Across both datasets, the hyperparameter with the highest impact was the loss function, incurring significant drops in performance if not used. This observation confirms the analysis in [Mohamed et al., 2019], who argues for the substantial contribution of the choice of loss function on KGE models performance.

Across both datasets, the best training type for all models except TransE was 1vsAll (TransE only accepts negative sampling as a training strategy). This is different from the results in the previous study in which there was more diversity across the training types for best performing models.

During the Bayesian optimisation stage on the best performing models selected during quasi-random search, there were only marginal improvements to be achieved. This can be seen in Table 5 in the Appendix.

Apart from training strategy and loss function, the impact of specific hyperparameters was difficult to evaluate individually. This is because the impact of hyperparameters cannot be observed on an individual basis due to the limited number of 30 configurations. All that can be inferred from Table 4 is that a specific hyperparameter value was part of the best model and the possible drop in MRR value if the next best set of hyperparameters was selected. Since the other configurations did not have the same combination of hyper-

		RESCAL	TransE	DistMult	ComplEx	ConvE
AIFB	Mean MRR	*41.6*	*45.6*	*48.6*	*48.6*	*47.5*
	Emb. size	256 *(-1.0)*	512 *(-7.2)*	128 *(-1.5)*	512 *(-0.2)*	288 *(-1.5)*
	Batch size	128 *(0.3)*	128 *(-9.2)*	512 *(-1.5)*	1024 *(-0.2)*	256 *(-1.5)*
	Train type	1vsAll *(-1.6)*	NegSamp –	1vsAll *(-1.5)*	1vsAll *(-0.9)*	1vsAll *(-1.5)*
	Loss	CE *(-9.0)*	CE *(-9.2)*	CE *(-4.7)*	CE *(-4.4)*	CE *(-2.1)*
	Optimizer	Adagrad *(-0.7)*	Adagrad *(-7.2)*	Adagrad *(-1.9)*	Adam *(-0.2)*	Adagrad *(-1.7)*
	Initializer	Normal *(0.3)*	XvNorm *(-7.2)*	Unif. *(-1.9)*	XvNorm *(-0.2)*	XvNorm *(-2.0)*
	Regularizer	L3 *(0.3)*	L2 *(-9.2)*	L2 *(-1.5)*	L2 *(-0.9)*	L1 *(-1.7)*
	Reciprocal	Yes *(-0.7)*	Yes *(-9.5)*	No *(-1.5)*	Yes *(-0.2)*	Yes –
MUTAG	Mean MRR	*36.7*	*37.9*	*47.5*	*38.0*	*30.8*
	Emb. size	256 *(-2.1)*	512 *(-15.2)*	128 *(-10.9)*	128 *(-0.4)*	128 *(-1.0)*
	Batch size	512 *(-2.1)*	128 *(-15.2)*	512 *(-10.9)*	1024 *(-0.4)*	128 *(-1.0)*
	Train type	1vsAll *(-2.3)*	NegSamp –	1vsAll *(-12.7)*	1vsAll *(-3.0)*	1vsAll *(-4.2)*
	Loss	CE *(-6.7)*	CE *(-22.9)*	CE *(-18.7)*	CE *(-8.4)*	BCE *(-1.0)*
	Optimizer	Adam *(-2.3)*	Adagrad *(-19.6)*	Adagrad *(-11.2)*	Adam *(-0.4)*	Adam *(-4.0)*
	Initializer	XvNorm *(-2.3)*	XvNorm *(-15.2)*	Unif. *(-10.9)*	Unif. *(-0.4)*	XvUnif *(-1.0)*
	Regularizer	L3 *(-2.1)*	L2 *(-15.2)*	L2 *(-12.8)*	L2 *(-2.9)*	L2 *(-1.0)*
	Reciprocal	No *(-2.3)*	Yes *(-15.2)*	No *(-12.7)*	No *(-1.6)*	Yes –

Table 4: Hyperparameters of best performing models w.r.t. filtered MRR (on validation data) after quasi-random hyperparameter search. For each hyperparameter, we also give the reduction in filtered MRR for the best configuration that does not use this value of the hyperparameter (in parenthesis).

parameters, each individual one cannot be simply categorised as the best possible choice. Hyperparameters themselves are highly dependent on the choice of model, dataset and also on the synergy with other hyperparameters. For a complete list on the hyperparameter values selected for best models in both datasets, see table 6 and 7.

5 Related Work

[Ruffinelli et al., 2020] conducts an experimental study and compares the performance of 5 KGE models trained with 2 public datasets (FB15K-237 and WNRR). The experiment searches through a large hyperparameter space: major training types, reciprocal relations, loss functions, regularization techniques, optimizers and initialization methods, embedding size and batch size. The study shows that when trained using a consistent method, the relative performance of the models shrinks, enabling less sophisticated models to perform comparably with more expressive ones. The authors developed the `libkge` library, which is also used in this study, enabling a straightforward comparison with our results.

Other similar studies focus on the impact of specific hyperparameters. [Akrami et al., 2018] compares the performance of a range of KGE models with respect to the existence of symmetrical relations within the dataset. The study concludes that symmetry in the

dataset determines marked improvements in performance. In another experimental study, [Mohamed et al., 2019] studies the impact of the loss function hyperparamenter on the performance of a wide range of KGE models and highlights the fact that it plays a crucial role, conclusion which is supported in this work as well.

6 Conclusions

In this thesis we researched the impact of using domain specifics datasets on the training and hyperparamenters of KGE models. We performed an experimental study that performed hyperparameter search, trained and evaluated five of the most popular decomposable KGE models on two domain specific datasets (AIFB and MUTAG). We evaluated the models using the entity ranking protocol and assessed which of the hyperparameters impacted performance the most. The results are compared with previous studies on generic benchmark datasets that used equivalent methodologies and observed that in our experiment the models performed generally better. The KGE models trained on AIFB dataset outperformed significantly the ones trained on FB15k-237 or WNRR. This result can be explained through the existence of relation symmetry and fact redundancy in some of the AIFB relations, as well as the incorporation of ontological relations in the training. Models trained on MUTAG however have a smaller performance gap compared to the ones trained on FB15k-237 or WNRR. For both AIFB and MUTAG, the model architecture achieving the best performance was DistMult, followed closely by ComplEx. Cross entropy was still found to be the best loss function as in [Ruffinelli et al., 2020], but the best training strategy was largely 1VsAll in our study. While some hyperparameters make a clear performance difference, some others depend heavily on the dataset used, specifically in terms of relation density and domain specificity. Our study showed that domain specific datasets contribute to better KGE performance mostly due to the ontological structure and their intrinsic redundancy in expressing facts through the triples. We conclude that the performance of a KGE model is determined not only by the model architecture and the hyperparameter values, but also it is very heavily influenced by the dataset used. Possible performance gains can be achieved with an effective method to incorporate special semantics of ontological restrictions into the training of the models on domain specific datasets, which remains an areas of further research.

References

[Akrami et al., 2018] Akrami, F., Guo, L., Hu, W., and Li, C. (2018). Re-evaluating embedding-based knowledge graph completion methods. pages 1779–1782.

[Bal et al., 2016] Bal, H., Epema, D., de Laat, C., van Nieuwpoort, R., Romein, J., Seinstra, F., Snoek, C., and Wijshoff, H. (2016). A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer*, 49(5):54–63.

[Balazevic et al., 2019] Balazevic, I., Allen, C., and Hospedales, T. (2019). TuckER: Tensor factorization for knowledge graph completion. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5185–5194, Hong Kong, China. Association for Computational Linguistics.

[Bechhofer et al., 2004] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneijder, P., and Stein, L. A. (2004). OWL Web Ontology Language Reference. Recommendation, World Wide Web Consortium (W3C). See `http://www.w3.org/TR/owl-ref/`.

[Bloehdorn and Sure, 2007] Bloehdorn, S. and Sure, Y. (2007). Kernel methods for mining instance data in ontologies. In Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., and Cudré-Mauroux, P., editors, *The Semantic Web*, pages 58–71. Springer Berlin Heidelberg.

[Bordes et al., 2013] Bordes, A., Usunier, N., Garcia-Durán, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 2787–2795, Red Hook, NY, USA. Curran Associates Inc.

[Bühmann et al., 2018] Bühmann, L., Lehmann, J., Westphal, P., and Bin, S. (2018). Dl-learner structured machine learning on semantic web data. In *Companion Proceedings of the The Web Conference 2018*, page 467–471. International WWW Conferences Steering Committee.

[Dettmers et al., 2018] Dettmers, T., Minervini, P., Stenetorp, P., and Riedel, S. (2018). Convolutional 2d knowledge graph embeddings. In *AAAI*.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12(null):2121–2159.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization.

[Klyne and Carroll, 2004] Klyne, G. and Carroll, J. J. (2004). Resource description framework (rdf): Concepts and abstract syntax. W3C Recommendation.

[Lacroix et al., 2018] Lacroix, T., Usunier, N., and Obozinski, G. (2018). Canonical tensor decomposition for knowledge base completion. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2863–2872, Stockholmsmässan, Stockholm Sweden. PMLR.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[Mohamed et al., 2019] Mohamed, S. K., Novácek, V., Vandenbussche, P.-Y., and Muñoz, E. (2019). Loss functions in knowledge graph embedding models. In *DL4KG@ESWC*.

[Nickel et al., 2011] Nickel, M., Tresp, V., and Kriegel, H.-P. (2011). A three-way model for collective learning on multi-relational data. pages 809–816.

[Ruffinelli et al., 2020] Ruffinelli, D., Broscheit, S., and Gemulla, R. (2020). You CAN teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*.

[Shang et al., 2019] Shang, C., Tang, Y., Huang, J., Bi, J., He, X., and Zhou, B. (2019). End-to-end structure-aware convolutional networks for knowledge base completion. *ArXiv*, abs/1811.04441.

[Sobol, 1967] Sobol, I. M. (1967). Distribution of points in a cube and approximate evaluation of integrals. *Zh. Vych. Mat. Mat. Fiz.*, 7:784–802.

[Socher et al., 2013] Socher, R., Chen, D., Manning, C. D., and Ng, A. (2013). Reasoning with neural tensor networks for knowledge base completion. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 926–934. Curran Associates, Inc.

[Toutanova and Chen, 2015] Toutanova, K. and Chen, D. (2015). Observed versus latent features for knowledge base and text inference.

[Trouillon et al., 2016] Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., and Bouchard, G. (2016). Complex embeddings for simple link prediction. *ArXiv*, abs/1606.06357.

[Yang et al., 2015] Yang, B., tau Yih, W., He, X., Gao, J., and Deng, L. (2015). Embedding entities and relations for learning and inference in knowledge bases. *CoRR*, abs/1412.6575.

7 Appendix

		RESCAL	TransE	DistMult	ComplEx	ConvE
AIFB	Learning rate	0.09557	0.09509	0.41499	0.00050	0.07954
	Scheduler patience	6	6	7	9	4
	Entity reg. weight	1.90^{-08}	2.13^{-07}	2.53^{-08}	2.40^{-19}	3.28^{-12}
	Relation reg. weight	4.04^{-03}	3.44^{-10}	9.62^{-15}	6.43^{-09}	3.29^{-13}
	Entity emb. dropout	0.42	0.10	0.43	0.00	0.32
	Relation emb. dropout	0.16	0.04	0.05	0.48	0.11
	Projection dropout (ConvE)	–	–	–	–	0.44
	Feature map dropout (ConvE)	–	–	–	–	0.22
MUTAG	Learning rate	0.00301	0.00838	1.00000	0.02447	0.01219
	Scheduler patience	4	1	4	6	10
	Entity reg. weight	4.55^{-14}	3.27^{-13}	2.44^{-09}	5.65^{-09}	1.33^{-15}
	Relation reg. weight	1.84^{-18}	6.53^{-17}	1.11^{-11}	2.04^{-06}	5.66^{-15}
	Entity emb. dropout	0.00	0.00	0.50	0.44	0.41
	Relation emb. dropout	0.00	0.00	0.50	0.07	0.00
	Projection dropout (ConvE)	–	–	–	–	0.16
	Feature map dropout (ConvE)	–	–	–	–	0.37

Table 5: : Hyperparameters of best performing models found with Bayesian optimization. All other hyperparameters are the same as in Table 4

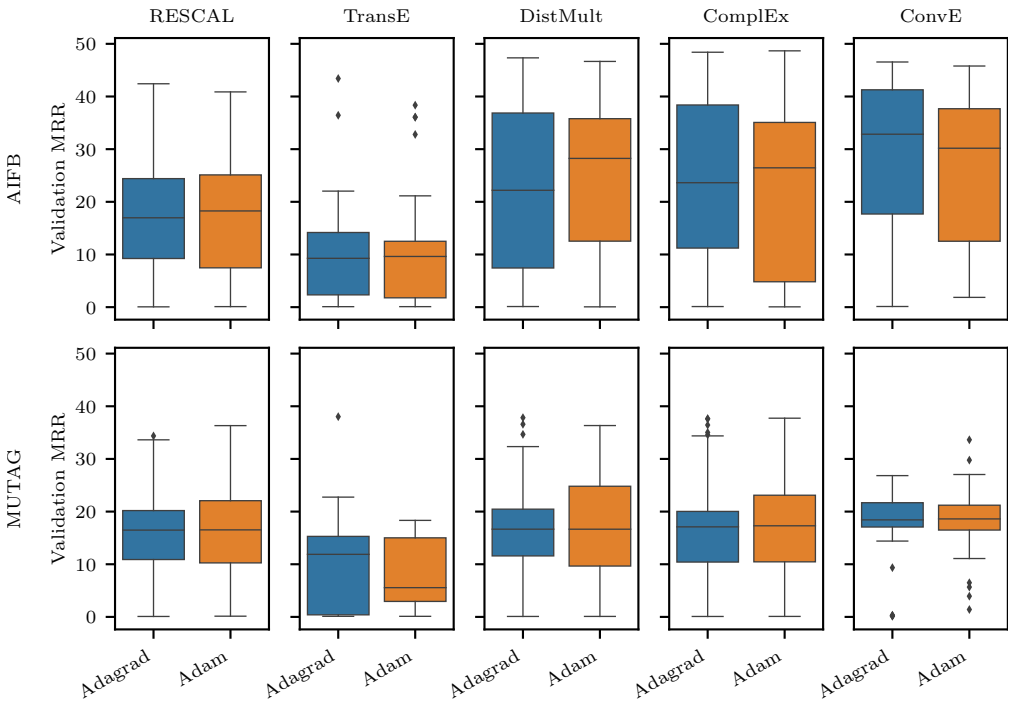

Figure 4: Distribution of filtered MRR (%) on validation data over each of the models for different optimizers. Top row: AIFB; bottom row: MUTAG

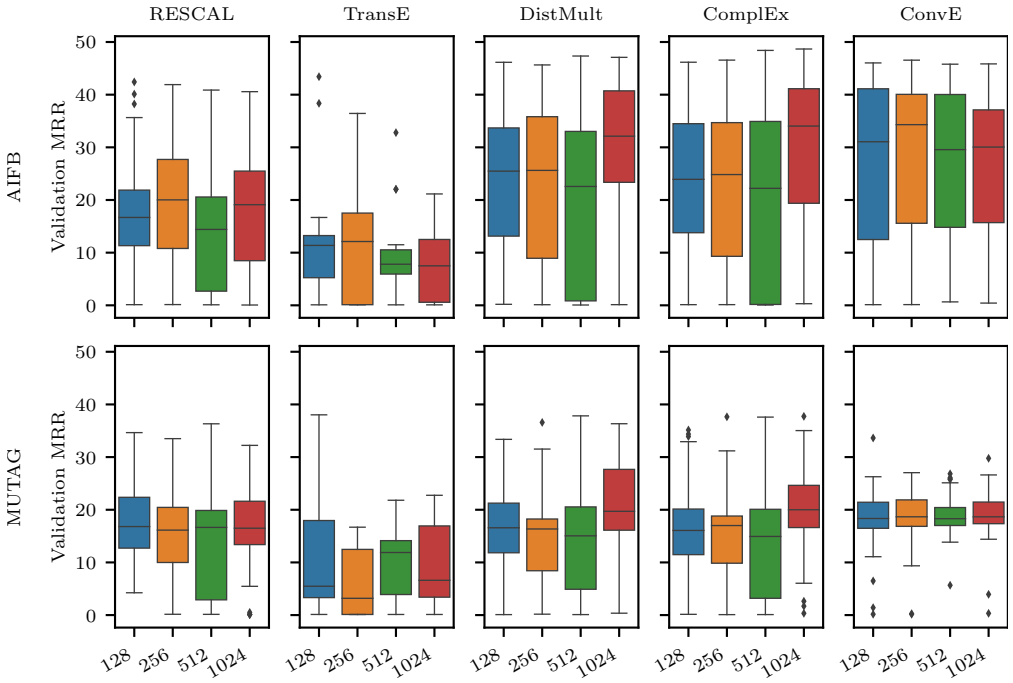

Figure 5: Distribution of filtered MRR (%) on validation data over each of the models for different batch sizes. Top row: AIFB; bottom row: MUTAG

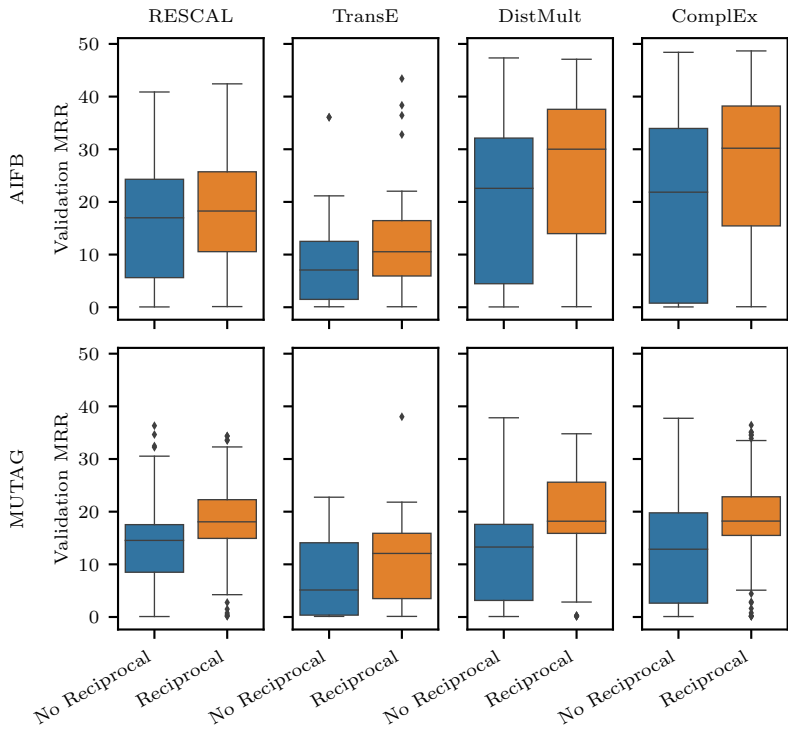

Figure 6: Distribution of filtered MRR (%) on validation data over each of the models for either reciprocal relations being used. Top row: AIFB; bottom row: MUTAG

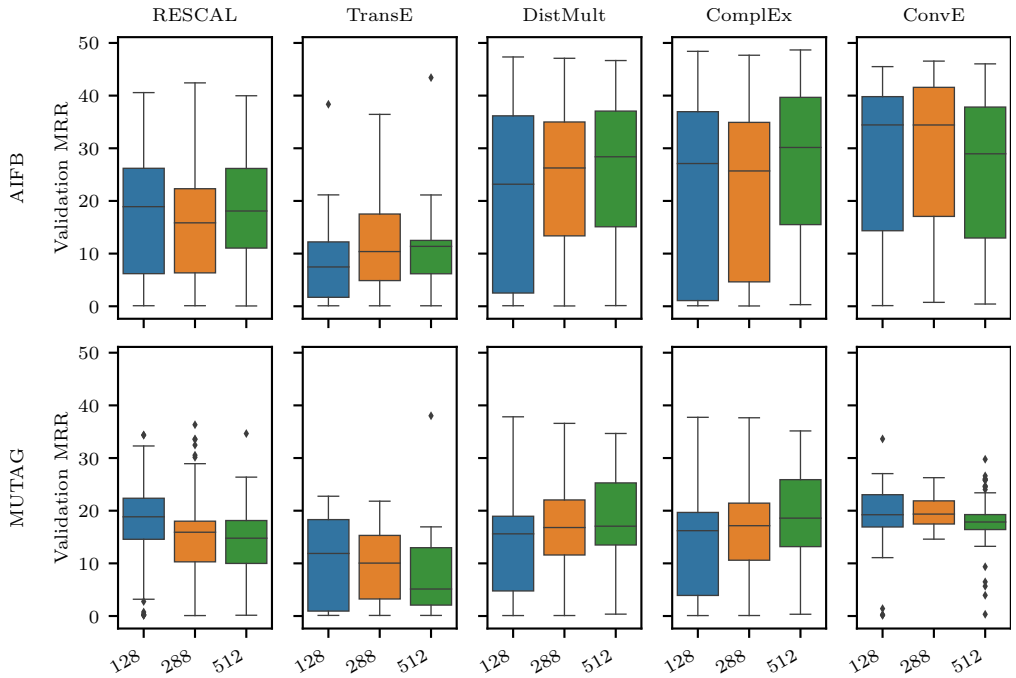

Figure 7: Distribution of filtered MRR (%) on validation data over each of the models for different embedding sizes. Top row: AIFB; bottom row: MUTAG

		RESCAL		TransE		DistMult		ComplEx		ConvE	
AIFB	Mean MRR		*41.6*		*45.6*		*48.6*		*48.6*		*47.5*
	Embedding size	256	*(-1.0)*	512	*(-7.2)*	128	*(-1.5)*	512	*(-0.2)*	288	*(-1.5)*
	Training type	1vsAll	*(-1.6)*	NegSamp	–	1vsAll	*(-1.5)*	1vsAll	*(-0.9)*	1vsAll	*(-1.5)*
	Reciprocal	Yes	*(-0.7)*	Yes	*(-9.5)*	No	*(-1.5)*	Yes	*(-0.2)*	Yes	–
	No. subject samples (NegSamp)	–		2		–		–		–	
	No. object samples (NegSamp)	–		56		–		–		–	
	Label Smoothing (KvsAll)	–		–		–		–		–	
	Loss	CE	*(-9.0)*	CE	*(-9.2)*	CE	*(-4.7)*	CE	*(-4.4)*	CE	*(-2.1)*
	Margin (MR)	–		–		–		–		–	
	L_p-norm (TransE)	–		L2		–		–		–	
	Optimizer	Adagrad	*(-0.7)*	Adagrad	*(-7.2)*	Adagrad	*(-1.9)*	Adam	*(-0.2)*	Adagrad	*(-1.7)*
	Batch size	128	*(0.3)*	128	*(-9.2)*	512	*(-1.5)*	1024	*(-0.2)*	256	*(-1.5)*
	Learning rate	0.12983		0.04122		0.50338		0.00034		0.03293	
	Scheduler patience	5	*(0.3)*	6	*(-7.2)*	7	*(-1.5)*	7	*(-0.9)*	1	*(-1.5)*
	L_p regularization	L3	*(0.3)*	L2	*(-9.2)*	L2	*(-1.5)*	L2	*(-0.9)*	L1	*(-1.7)*
	Entity emb. weight	1.02^{-08}		1.32^{-07}		1.48^{-18}		9.58^{-13}		6.10^{-16}	
	Relation emb. weight	1.08^{-13}		3.72^{-18}		1.44^{-18}		2.29^{-02}		1.03^{-16}	
	Frequency weighting	No	*(-0.7)*	No	*(-23.6)*	No	*(-1.5)*	Yes	*(-0.2)*	Yes	*(-1.7)*
	Embedding normalization (TransE)										
	Entity	–		No		–		–		–	
	Relation	–		No		–		–		–	
	Dropout										
	Entity embedding	0.00		0.00		0.05		0.08		0.01	
	Relation embedding	0.00		0.00		0.44		0.06		0.00	
	Projection (ConvE)	–		–		–		–		0.42	
	Feature map (ConvE)	–		–		–		–		0.15	
	Embedding initialization	Normal	*(0.3)*	XvNorm	*(-7.2)*	Unif.	*(-1.9)*	XvNorm	*(-0.2)*	XvNorm	*(-2.0)*
	Std. deviation (Normal)	0.15763		–		–		–		–	
	Interval (Unif)	–		–		[-0.31, 0.31]		–		–	

Table 6: Hyperparameters of best performing models found with quasi-random hyperparameter optimization on *AIFB*. Settings that apply only to certain configurations are indicated in parenthesis. For each hyperparameter, we also give the reduction in filtered MRR for the best configuration that does not use this value of the hyperparameter (in parenthesis).

		RESCAL		TransE		DistMult		ComplEx		ConvE	
MUTAG	Mean MRR		*36.7*		*37.9*		*47.5*		*38.0*		*30.8*
	Embedding size	256	*(-2.1)*	512	*(-15.2)*	128	*(-10.9)*	128	*(-0.4)*	128	*(-1.0)*
	Training type	1vsAll	*(-2.3)*	NegSamp	–	1vsAll	*(-12.7)*	1vsAll	*(-3.0)*	1vsAll	*(-4.2)*
	Reciprocal	No	*(-2.3)*	Yes	*(-15.2)*	No	*(-12.7)*	No	*(-1.6)*	Yes	–
	No. subject samples (NegSamp)	–		2		–		–		–	
	No. object samples (NegSamp)	–		56		–		–		–	
	Label Smoothing (KvsAll)	–		–		–		–		–	
	Loss	CE	*(-6.7)*	CE	*(-22.9)*	CE	*(-18.7)*	CE	*(-8.4)*	BCE	*(-1.0)*
	Margin (MR)	–		–		–		–		–	
	L_p-norm (TransE)	–		L2		–		–		–	
	Optimizer	Adam	*(-2.3)*	Adagrad	*(-19.6)*	Adagrad	*(-11.2)*	Adam	*(-0.4)*	Adam	*(-4.0)*
	Batch size	512	*(-2.1)*	128	*(-15.2)*	512	*(-10.9)*	1024	*(-0.4)*	128	*(-1.0)*
	Learning rate	0.00246		0.04122		0.50338		0.03859		0.03884	
	Scheduler patience	9	*(-2.1)*	6	*(-15.2)*	7	*(-10.9)*	5	*(-0.4)*	0	*(-1.0)*
	L_p regularization	L3	*(-2.1)*	L2	*(-15.2)*	L2	*(-12.8)*	L2	*(-2.9)*	L2	*(-1.0)*
	Entity emb. weight	1.09^{-05}		1.32^{-07}		1.48^{-18}		1.69^{-15}		5.15^{-04}	
	Relation emb. weight	6.55^{-06}		3.72^{-18}		1.44^{-18}		3.40^{-08}		8.90^{-20}	
	Frequency weighting	Yes	*(-2.3)*	No	*(-16.1)*	No	*(-11.2)*	Yes	*(-0.4)*	No	*(-4.0)*
	Embedding normalization (TransE)										
	Entity	–		No		–		–		–	
	Relation	–		No		–		–		–	
	Dropout										
	Entity embedding	0.00		0.00		0.05		0.00		0.39	
	Relation embedding	0.00		0.00		0.44		0.34		0.05	
	Projection (ConvE)	–		–		–		–		0.13	
	Feature map (ConvE)	–		–		–		–		0.29	
	Embedding initialization	XvNorm	*(-2.3)*	XvNorm	*(-15.2)*	Unif.	*(-10.9)*	Unif.	*(-0.4)*	XvUnif	*(-1.0)*
	Std. deviation (Normal)	–		–		–		–		–	
	Interval (Unif)	–		–		[-0.31, 0.31]		[-0.11, 0.11]		–	

Table 7: Hyperparameters of best performing models found with quasi-random hyperparameter optimization on *MUTAG*. The settings that apply only to certain configurations are indicated in parenthesis. For each hyperparameter, we also give the reduction in filtered MRR for the best configuration that does not use this value of the hyperparameter (in parenthesis).

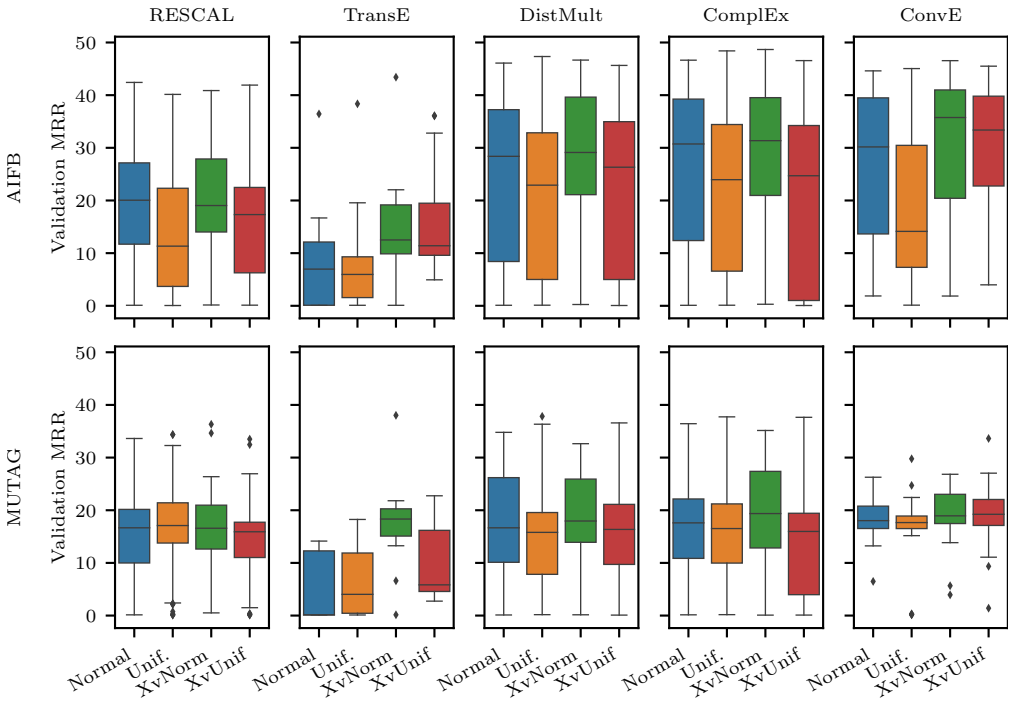

Figure 8: Distribution of filtered MRR (%) on validation data over each of the models for different initializers. Top row: AIFB; bottom row: MUTAG

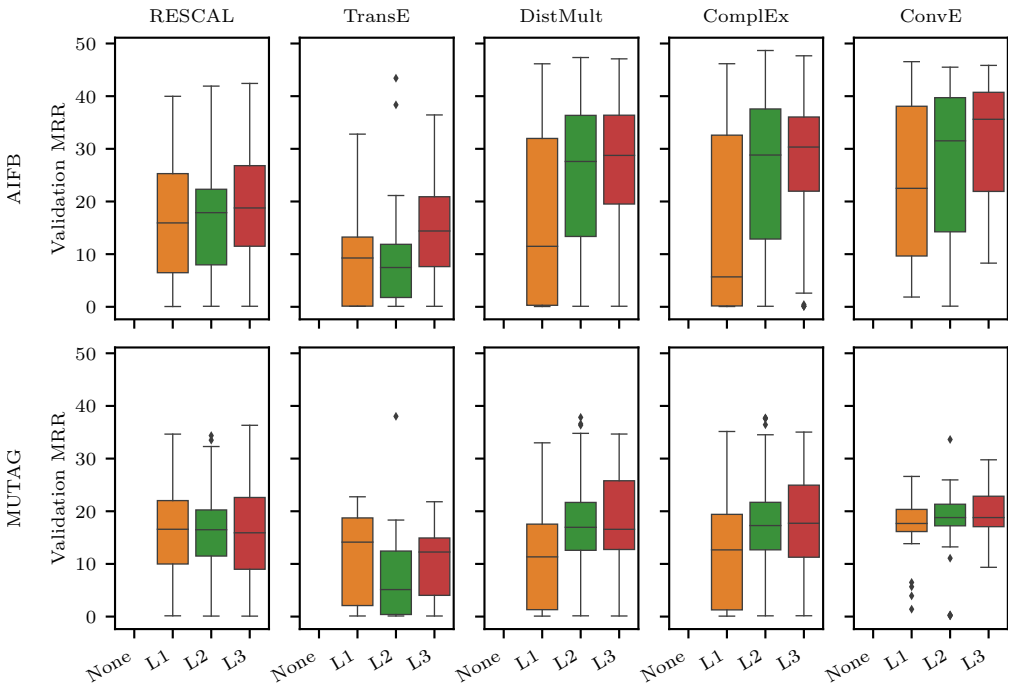

Figure 9: Distribution of filtered MRR (%) on validation data over each of the models for different penalty for the loss function. Top row: AIFB; bottom row: MUTAG

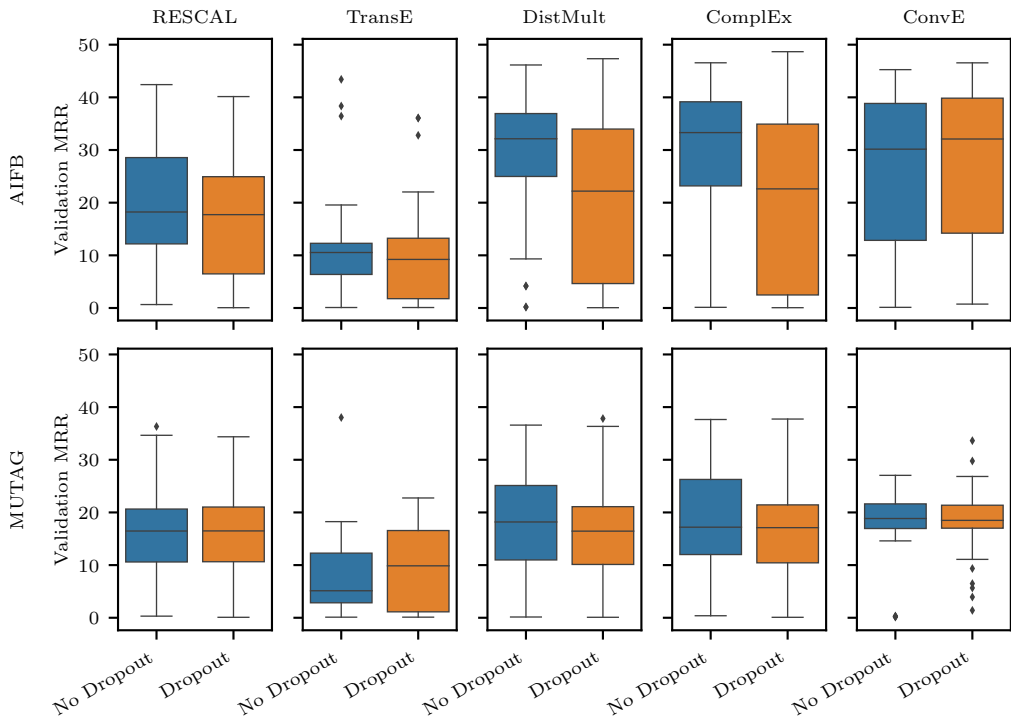

Figure 10: Distribution of filtered MRR (%) on validation data over each of the models with dropout/no dropout. Top row: AIFB; bottom row: MUTAG