UNIVERSITEIT VAN AMSTERDAM

MSc Artificial Intelligence

Master Thesis

Incorporating Semantics in Knowledge Graph Embeddings

by

Stefan F. Schouten

12808113

August 27, 2021

48 ECTS

October 2020 - July 2021

Supervisors:

Thiviyan Thanapalasingam MSc

Daniel Daza MSc

Assessor:

Dr. Paul Groth

INDE lab

INtelligent Data Engineering Lab

Abstract

Knowledge Graph Embeddings (KGEs) aim to represent the entities and relations within a Knowledge Graph using continuous vector representations. This allows them to be utilized more easily for tasks such as Link Prediction, Relation Extraction, and Question Answering. Commonly, these embeddings focus on representing the structure of the Knowledge Graph, but incorporating semantics may increase their utility. We compare several methods that incorporate an abundantly available form of semantics: type-information. We include an existing method, TransT, as well as new methods that either construct entity-embeddings from type-embeddings, or model type-distributions over entity-embeddings. We perform extensive hyper-parameter searches in order to compare the performance of these methods when applied to the Link Prediction task. We observe up to 261% increase of MRR against a TransE baseline for a class of methods that learned to represent each entity by one of its types, using that type's embedding as the entity's embedding. Other methods result in a modest increase in performance relative to their non-semantic baseline. While it remains challenging to incorporate type-information such that it may benefit more than just Link Prediction, the high performance we observe shows there is great potential in its utilization.

Contents

List of Figures

List of Tables

Preface

The first time I became interested in Knowledge Graphs was during my bachelor's thesis project. During the project we developed a prototype for the automated summarization of medical consultation dialogues. The architecture we were tasked with implementing envisioned a knowledge graph (with prespecified ontology) at the center. Facts from the dialogue would first be extracted and stored in the knowledge graph, and then the relevant part of the knowledge graph would be used to generate a summary.

Under some naive assumptions we managed to implement this well enough for a proof of concept. But since then I've been fascinated by the possibility of taking advantage of recent advances in Deep Learning to perform this kind of knowledge extraction from natural language. What motivated me to choose the topic of this thesis was that the inclusion of semantics (both those represented in an ontology, and those from natural languages) in Knowledge Graph Embeddings seems vital for them to be used in this type of knowledge extraction.

I would like to thank my supervisors Thiviyan Thanapalasingam and Daniel Daza for their frequent encouragement and reassurance, as well as their excellent advice throughout this project.

Stefan F. Schouten

Gouda, August 27, 2021

1 Introduction

Knowledge Graphs represent (real-world) information using a graph-based structure. They are used to represent both domain-specific and general information. Nodes in the graph represent entities, whereas the edges between nodes represent a various kinds of relations that exist between two entities.

Knowledge Graphs are already critical to the operation of large tech companies such as Microsoft, Google and Facebook, where they are used to integrate and manage data from diverse sources at a large scale (Noy et al. 2019)[1]. Knowledge graphs are also strongly related to concepts in the Semantic Web, the effort to make information on the Internet machine-readable (Paulheim 2017)[2].

Despite their widespread use, knowledge graphs are generally not complete, entities that should be connected by a given relation are often not. Within the field of Artificial Intelligence, research is conducted into a task known as Knowledge Graph Completion. That task's objective is to identify missing facts which should be added to the knowledge graph. When this task is performed primarily using information found within the knowledge graph itself it is also referred to as Link Prediction (Rossi et al. 2021)[3].

To perform tasks like Link Prediction the Knowledge Graph can be embedded. Traditionally only the Knowledge Graph's syntactical information is incorporated in such an embedding. While this information alone can already model the Knowledge Graph quite well, this thesis will explore ways of also incorporating semantics to improve their performance.

In this chapter we will introduce the relevant notation, explain what task this thesis will focus on, and finally we will formulate the exact research question.

[1] "Industry-Scale Knowledge Graphs: Lessons and Challenges: Five Diverse Technology Companies Show How It's Done" DOI: 10.1145/3329781.3332266

[2] "Knowledge graph refinement: A survey of approaches and evaluation methods" DOI: 10.3233/SW-160218

[3] "Knowledge Graph Embedding for Link Prediction: A Comparative Analysis" DOI: 10.1145/3424672

Knowledge Graph

A Knowledge Graph consists of a set of entities $\mathcal{E} = \{e_i\}$, a set of relations $\mathcal{R} = \{r_j\}$, and a set of facts $\mathcal{D}^+$:

$$\mathcal{D}^+ \subseteq \mathcal{D} = \mathcal{E} \times \mathcal{R} \times \mathcal{E} = \{(e_{head}, r, e_{tail})^{(k)}\} = \{(h, r, t)^{(k)}\}; \quad (1.1)$$

Note that $\mathcal{D}^+$ only contains observations $X_k = True$. What we assume about X for the triples that are not in D^+ determines if we operate under an open- or closed-world assumption. Under an open-world assumption we consider the other X's to be unknown, under a closed-world assumption we consider them to be *False*.

which determines which entities are related by what relations. The set $\mathcal{D}^+$ can also be considered as a set of observations of the Boolean random variable:

$$X^{(k)} \qquad \text{truth value of triple } k. \tag{1.2}$$

We can also consider the components of the triples to be random variables. For an arbitrary triple with index k' we might consider the following conditional random variables:

$$H^{(k')}|x^{(k')}; \qquad R^{(k')}|x^{(k')}; \qquad T^{(k')}|x^{(k')}. \tag{1.3}$$

Here the small $x^{(k)}$ is a value that the random variable $X^{(k)}$ might take (True or False). Throughout this document $\Gamma|a$ is short for $\Gamma|A = a$.

To avoid notational overload we drop the index whenever we refer to an arbitrary triple, giving simply:

$$H|x; \qquad R|x; \qquad T|x. \tag{1.4}$$

Their distribution says something about the likelihood of entities and relations being the Head, Relation, or Tail within a triple with truth value x. Here H, T and R are categorical random variables with $\mathcal{E}$ and $\mathcal{R}$ as their support respectively.

Knowledge Graph Embedding (KGE)

When we learn an embedding we associate symbols with continuous vector representations. We are concerned with learning such representations for the entities and relations in our Knowledge Graph. We wish to do so in order to use machine learning techniques such as Deep Learning. These techniques can then be used for a range of applications such as Link Prediction, Entity Classification, Relation Extraction, and Question Answering (Q. Wang et al. 2017a)[4].

[4] "Knowledge Graph Embedding: A Survey of Approaches and Applications" DOI: 10.1109/TKDE.2017.2754499

When tasked with learning a Knowledge Graph Embedding, we (implicitly) postulate additional unobserved multidimensional random variables[5] for arbitrary triple k':

$$\mathbf{H}^{(k')}|H^{(k')} = h^{(k')}; \qquad \mathbf{R}^{(k')}|R^{(k')} = r^{(k')}; \qquad \mathbf{T}^{(k')}|T^{(k')} = t^{(k')}. \tag{1.5}$$

[5] To avoid notational overload once more, we refer to these as simply $\mathbf{H}, \mathbf{R}, \mathbf{T}$ whenever we mean an arbitrary triple, or as $\mathbf{H}^{(k)}, \mathbf{R}^{(k)}, \mathbf{T}^{(k)}$ when we mean a specific triple's embedding.

In other words, we imagine that the truth value for each possible triple was sampled from a distribution $p(x|\mathbf{h}, \mathbf{r}, \mathbf{t})$; where $\mathbf{h}, \mathbf{r}$, and $\mathbf{t}$ were in turn sampled from unknown posterior $p(\mathbf{h}, \mathbf{r}, \mathbf{t}|h, r, t)$. We are interested in approximating that posterior, thereby also gaining access to samples $(\mathbf{h}, \mathbf{r}, \mathbf{t})$ which we can use as the embeddings we seek. Because we often model the distribution of the embeddings without considering the position of the entities in a triple, we also define:

$$\mathbf{E}_i := \mathbf{H}|H{=}e_i \;=\; \mathbf{T}|T{=}e_i \quad \text{and} \quad \mathbf{R}_j := \mathbf{R}|R{=}r_j. \tag{1.6}$$

That is to say, we model the posterior $p(\mathbf{h}, \mathbf{r}, \mathbf{t}|h, r, t)$ as $q_e(\mathbf{h}|h)q_r(\mathbf{r}|r)q_e(\mathbf{t}|t)$, i.e. forcing ourselves to use the same approximate posterior for $\mathbf{H}$ and $\mathbf{T}$.

When we make modeling decisions we make assumptions about how these random variables are distributed. For this we use the following type of notation:

$$\mathbf{E}_i|\theta_i \sim \text{Distribution}(f(\theta_i)); \tag{1.7}$$

where θ are model parameters. Often Knowledge Graph Embeddings are not modeled as a distribution, but are modeled as point estimates. We denote this as follows:

$$\mathbf{E}_i|\theta_i \doteq f(\theta_i); \tag{1.8}$$

that is to say the embeddings come from a deterministic function.

Semantic Information

Frequently Knowledge Graph Embeddings are learned based only on structural (syntactical) information (Q. Wang et al. 2017a)[6], i.e. which entities are connected by what relations. But the underlying assumption, that the truth value of possible triples can be inferred from syntactical information only, is rather simplifying. The syntactical information might allow us to notice that when an entities connect by relation $r1$ to entity a, they often also connect by relation $r2$ to entity b. But we are never able to utilize other ways entities might be similar. There are a few distinct sources of semantic information we could use to improve upon this.

[6] "Knowledge Graph Embedding: A Survey of Approaches and Applications" DOI: 10.1109/TKDE.2017.2754499

- Often we know Natural Language names of the entities in the Knowledge Graph. These can be seen as a bridge to those languages, this could allow us to utilize models of the semantics of those languages for the Knowledge Graph as well.
- Information of how entities appear (look, sound, behave, etc.) in the real world could help with *inferring* the semantics, since the real world is what we are trying to describe.
- Finally, the entities and relation-types in a Knowledge Graph come from our understanding of the world. One way to improve, is to describe more of that understanding manually by recording semantics in the form of an ontology.

It is the latter that will be the focus of this thesis.

Including Ontological Information. The most prevalent kind of ontological information are groupings of entities into classes. For example, we might have access to a set of entity classes $\mathcal{C} = \{c_s\}$. With corresponding observed random variable:

$$C_{i,s} := C|e_i, c_s \qquad \text{whether entity } i \text{ is of type (or class) } c_s. \tag{1.9}$$

These entity classes are essentially categories or **types** of entities, for example the entity 'Stefan' could be part of the 'Person' type. When including this information in the embedding process we hope that the types an entity falls under are predictive of the truth value of a triple (for example because certain relations generally exist only between entities of certain types).

Other kinds of ontological information include (Hitzler et al. 2012)[7]:

[7] *OWL 2 Web Ontology Language Primer (Second Edition)*

- restrictions, for example on the entity types between which certain relations can exist;
- entity-type and relation hierarchies, for example a 'hasCar' relation could be a specific kind of 'hasProperty' relation;
- characteristics of relations, for example that a relation is symmetrical, i.e. if (h,r,t) is true then (t,r,h) is also true.

Previous work (Hao et al. 2019[8]; Zhou et al. 2020[9]; P. Wang and Zhou 2020[10]) has investigated also embedding the entity-types. One approach is to embed these types in the same space as the entities. In this case we are effectively learning an additional distribution of entity-representations, the difference is that this one is conditioned on the type(s), e.g. $p_i(\mathbf{e}_i|c_{i,1}\dots c_{i,n_c})$. Alternatively, entity-classes can be embedded in a separate space. In both cases related ontological information, like a class hierarchy could also be incorporated.

[8] "Universal Representation Learning of Knowledge Bases by Jointly Embedding Instances and Ontological Concepts" DOI: 10.1145/3292500.3330838

[9] "JECI: A Joint Knowledge Graph Embedding Model for Concepts and Instances"

[10] "JECI++: A Modified Joint Knowledge Graph Embedding Model for Concepts and Instances" DOI: 10.1016/j.bdr.2020.100160

Research Question

Our concern is with 'Incorporating semantics in knowledge graph embeddings'. We choose to focus primarily on the incorporation of type-information to improve Link Prediction performance. Thus we ask the following research question:

*(**RQ1**)* To what extent can type-information improve link-prediction performance? And, *(**RQ2**)* what method of incorporating the type-information is most effective?

To answer this question we will first review existing methods in the literature on this topic in Chapter 2. In Chapter 3 we will describe what methods for incorporating type-information we will investigate. Chapter 4 will describe the experiments, as well as present and analyze the results. Finally, in Chapter 5 we will discuss our findings, and answer the research questions.

We find that type-information is highly informative, and when used on its own, can outperform KGEs that do not use types at all. The difficulty is in incorporating the type-information in a KGE along with the other (entity-level) information, such that it has higher Link Prediction performance, and is still suitable for the same use cases (such as entity classification, question answering, etc.).

2
Literature Review

This chapter will provide an overview of relevant literature. It will start with a few generally applicable Knowledge Graph Embedding (KGE) methods, followed by recent methods that incorporate type-information.

2.1 General KGE Methods

Methods that attempt to incorporate semantics generally do so by modifying existing KGE methods. This section briefly describes some of these existing methods to provide the necessary background for the methods that follow in the rest of the chapter.

A recent survey (Rossi et al. 2021)[1] divides KGE methods used for Link Prediction into three groups: tensor decomposition models, geometric models, and deep learning models. They differ from each other by the way that they calculate scores by which to rank triples on their likelihood of being true.

[1] "Knowledge Graph Embedding for Link Prediction: A Comparative Analysis" DOI: 10.1145/3424672

- Tensor decomposition models view the Knowledge Graph as a set of adjacency matrices collected into a three-dimensional tensor. They model Link Prediction by decomposing the tensor into a combination of low-dimensional vectors, the embeddings.
- Deep learning models use deep learning techniques such as convolutional or recurrent neural networks to perform Link Prediction, learning embeddings in the process.
- Geometric models interpret relations as a geometric transformation between the head and tail. Using embeddings for the head and tail of triples, the scores are computed as:

$$score(\mathbf{h}, \mathbf{r}, \mathbf{t}) = \delta(\tau(\mathbf{h}, \mathbf{r}), \mathbf{t}); \tag{2.1}$$

  where τ is the geometric transformation, and δ is a distance function.

Most existing studies on the inclusion of type-information we describe in this chapter are based on Geometric models.

Translating embeddings for modeling multi-relational data (TransE) (Bordes et al. 2013)

TransE is a geometric model that uses translation, it models the task of learning KGEs as follows:

$$\mathbf{E}_i|\epsilon_i \doteq \epsilon_i \tag{2.2}$$
$$\mathbf{R}_j|\rho_j \doteq \rho_j \tag{2.3}$$
$$score(\mathbf{h},\mathbf{r},\mathbf{t}) = ||\mathbf{h}+\mathbf{r},\mathbf{t}|| \tag{2.4}$$

Where ϵ_i and ρ_j are the model parameters used as the embeddings.

Learning to Represent Knowledge Graphs with Gaussian Embedding (KG2E) (He et al. 2015)

This method is related to the Geometric Models, but models the embeddings as Gaussian distributed.

$$\mathbf{E}_i|\mu_i,\Sigma_i \sim \text{Normal}(\mu_i,\Sigma_i) \tag{2.5}$$
$$\mathbf{R}_j|\mu_j,\Sigma_j \sim \text{Normal}(\mu_j,\Sigma_j) \tag{2.6}$$

To measure a 'distance' between the distributions they either have a asymmetric score based on the KL-Divergence:

$$score(\mathbf{H},\mathbf{R},\mathbf{T}) = D_{KL}(\text{Normal}(\mu_h-\mu_t,\Sigma_h+\Sigma_t),\mathbf{R}). \tag{2.7}$$

Or a symmetric one, the expected likelihood:

$$score(\mathbf{H},\mathbf{R},\mathbf{T}) = \text{Normal}(0;\mu_h-\mu_t-\mu_r,\Sigma_h+\Sigma_t+\Sigma_r). \tag{2.8}$$

2.2 *Incorporating type information*

Semantically Smooth Knowledge Graph Embedding (Guo et al. 2015)

This method makes use of type information to make the embedding space "semantically smooth". Two variants are proposed: Laplacian Eigenmaps and Locally Linear Embedding. The Laplacian Eigenmaps variant is based on the assumption that if two entities are of the same type their embeddings should be close to each other. Locally Linear Embedding assumes that an entity's embedding can be reconstructed from a linear combination of the entity embeddings that belong to the same type.

Type-Constrained Representation Learning in Knowledge Graphs (Krompaß et al. 2015)

This work utilizes type-information to alter various aspects in the optimization processes of the KGE methods that they include in their study. They use type-constraints known to exist for specific relations, restricting between which types that relation may exist. By ruling out any link of that relation ever existing the models they

train can use their parameters to focus on modeling the links that *are* allowed to exist.

Anticipating potentially incomplete or inconsistent type-constraints, they also experiment with approximating the constraints from the data.

Representation Learning of Knowledge Graphs with Hierarchical Types (TKRL) (Xie et al. 2016)

This work introduces a method called Type-embodied Knowledge Representation Learning (TKRL), it learns type-specific representations through projections. For each entity class z it learns a matrix $\mathcal{M}_z$. To obtain the projection matrix $\mathcal{M}_{e_i}$ for an entity e_i it accumulates the matrices for the types that the entity is an instance of.

$$\mathcal{M}_{e_i}|c_{i,1}, c_{i,2}, \ldots, c_{i,S} = \sum_{c_{i,z}} \alpha_z \mathcal{M}_z \tag{2.9}$$

Furthermore the $\mathcal{M}_z$ matrices are themselves recursive aggregations based on all matrices in the hierarchy of the type. These are calculated in one of two ways:

$$\mathcal{M}_z = \prod_{t=1}^{m} \mathcal{M}_{z^{(t)}} \quad \Big/ \quad \mathcal{M}_z = \sum_{t=1}^{m} \beta_t \mathcal{M}_{z^{(t)}} \tag{2.10}$$

It then models the problem as follows:

$$\mathbf{E}_i|\epsilon_i, \mathcal{M}_{e_i} \doteq \mathcal{M}_{e_i}\epsilon_i \tag{2.11}$$

$$\mathbf{R}_j|\rho_j \doteq \rho_j \tag{2.12}$$

$$score(\mathbf{h}, \mathbf{r}, \mathbf{t}) = ||\mathbf{h} + \mathbf{r}, \mathbf{t}|| \tag{2.13}$$

It thus infers $\mathbf{E}_i$ from $\mathcal{C}_{i,\cdot}$ in a deterministic way. It also uses the type information when constructing the negative samples to train on. Specifically, when changing the head or tail entity in a triple, they increase the probability of selecting heads/tails with the same types.

TransT: Type-based multiple embedding representations for knowledge graph completion (Ma et al. 2017)

TransT models the entity embeddings as random variables with a discrete distribution. The relation embeddings are deterministic like in TransE.

We use the Discrete$(\langle x_1, \ldots, x_n \rangle, \mathbf{p})$ distribution to mean a discrete distribution has that assigns probability $\mathbf{p}_i$ to element x_i.

$$\mathbf{E}_i|\epsilon_i^{(1)}, \ldots, \epsilon_i^{(n_i)}, \mathbf{w}_i \sim \text{Discrete}(\langle \epsilon_i^{(1)}, \ldots, \epsilon_i^{(n)} \rangle, \mathbf{w}_i) \tag{2.14}$$

$$\mathbf{R}_j|\rho_j \doteq \rho_j \tag{2.15}$$

$$score(\mathbf{h}, \mathbf{r}, \mathbf{t}) = ||\mathbf{h} + \mathbf{r}, \mathbf{t}|| \tag{2.16}$$

Here $\mathbf{w}_i$ is a model parameter, it is a vector with the probabilities of each possible value $\mathbf{E}_i$ might take. The number of possible values n_i is learned using a Chinese Restaurant Process.

Because we now have a non-deterministic set of embeddings, we have the following marginal likelihood:

$$p(x^{(k)}|h^{(k)}, r^{(k)}, t^{(k)}) = \mathop{\mathbb{E}}_{\mathbf{h}\sim\mathbf{H}^{(k)}} \left[\mathop{\mathbb{E}}_{\mathbf{t}\sim\mathbf{T}^{(k)}} \left[p(x^{(k)}|\mathbf{h}, \mathbf{r}^{(k)}, \mathbf{t}) \right] \right] \tag{2.17}$$

Instead of maximizing the likelihood, Ma et al. maximizes one of three posteriors. Specifically they use a Margin Ranking Loss between a posterior and the corrupted posterior:

We use x^+ as short for $X = True$.

$$\mathcal{L} = \begin{cases} -\ln p(h|r,t,x^+) + \ln p(h'|r,t,x^+) & h' \neq h \\ -\ln p(r|h,t,x^+) + \ln p(r'|h,t,x^+) & r' \neq r \\ -\ln p(t|h,r,x^+) + \ln p(t'|h,r,x^+) & t' \neq t \end{cases} \tag{2.18}$$

These posteriors are proportional to the product of the likelihood and a prior which we will define below:

$$p(h|r,t,x) \propto p(x|h,r,t)p(h|r,t) \tag{2.19}$$
$$p(r|h,t,x) \propto p(x|h,r,t)p(r|h,t) \tag{2.20}$$
$$p(t|h,r,x) \propto p(x|h,r,t)p(t|h,r) \tag{2.21}$$

TransT uses type information to estimate the prior. Every entity belongs to set of types, from this two sets of *common* types are constructed for every relation. These sets are the types that are used *common*ly in the head/tail of a triple with this relation. They are constructed using a special kind of intersection of sets, which contains all elements that occur with at least a minimum frequency ρ in the sets on which the intersection is performed.

With these type sets constructed for each entity and relation, the prior is now estimated using the following similarity function s inspired by the Jaccard Index:

$$s(\mathcal{T}_a, \mathcal{T}_b) = \frac{|\mathcal{T}_a \cap \mathcal{T}_b|}{|\mathcal{T}_a|}, \tag{2.22}$$

where $\mathcal{T}_a, \mathcal{T}_b$ are type sets. The priors are then given by:

$$p(h|r,t) \propto s(r_{head}, h)^{\lambda_{head}} \quad s(t,h)^{\lambda_{relation}}; \tag{2.23}$$
$$p(r|h,t) \propto s(r_{head}, h)^{\lambda_{head}} \quad s(r_{tail}, t)^{\lambda_{tail}}; \tag{2.24}$$
$$p(t|h,r) \propto s(r_{tail}, t)^{\lambda_{tail}} \quad s(h,t)^{\lambda_{relation}}; \tag{2.25}$$

where $\lambda_{head}, \lambda_{tail}, \lambda_{relation} \in \{0, 1\}$ are hyperparameters[2].

[2] Note that expressions like $s(r_{head}, h)$ are shorthands for the similarity between the **type sets of** r_{head} and h.

Besides using type information to estimate the prior, TransT also uses type information to determine the number of embeddings each entity should have. To accomplish this it associates each embedding of an entity with a type set, which together, are referred to as that entity's *Semantics*[3]. Before training, each entity e_i has a single embedding, and the type set associated with it that entity's own type set ($Semantics_{e_i} = \{\mathcal{T}_i\}$). During training there is a probability that that the number of embeddings for an entity (being the head or tail of triple in the training data) is increased. This probability is based on the similarity of the different semantics of the head or tail

[3] Each type set in an entity's *Semantics* is meant to represent a different interpretation of that entity. And the embedding associated with each type set is meant to allow the model to learn a representation that encapsulates this different interpretation.

to the types of the entities that are *common*ly used with the triple's relation. For the tail the probability would be:

$$p_{new,tail}(h,r,t) = \left(1 - \max_{\sigma_i \in Semantics_t} s(\sigma_i, r_{tail})\right) \frac{\beta e^{-||r||_1}}{\beta e^{-||r||_1} + p(x|h,r,t)}. \tag{2.26}$$

When a positive result is sampled from this probability, the type set r_{tail} is used as the type set associated with the new embedding (the new semantic interpretation).

Differentiating Concepts and Instances for Knowledge Graph Embedding (Lv et al. 2018)

This work introduces TransC, which is similar to TransE and other translational methods, but additionally models concepts (types or entity-classes) as circles in the embedding space. They optimize their model such that: entities lie within the circle of the class that they are an instance of; and, when a class has a super-class, its circle is contained within the super-class's circle.

Knowledge graph embedding with concepts (Guan et al. 2019)

This work introduces the Knowledge Graph Embeddings with Concepts (KEC) model, which consists of two components. The first component is the concept graph embedding, it is a skip-gram based model, which learns representations of entities and its concepts by trying to predict the 'context' in which they occur. This context is defined as all the entities that share a concept with the entity. The study predicts the probability of an entity occurring in the target entity's context using both the dot product between the target entity and potential context entities, as well as the dot product between the entity's concepts and potential context entities.

The second component is the actual knowledge graph embedding, which is based on TransE and utilizes the embeddings from the concept graph embedding to construct a hyperplane representing the concept relevance between entities. To accomplish this the translational difference $\mathbf{h} + \mathbf{r} - \mathbf{t}$ is projected onto the hyperplane to represent how likely this link is given the concept (semantic) relevance. The two components are trained consecutively.

Universal Representation Learning of Knowledge Bases by Jointly Embedding Instances and Ontological Concepts (JOIE) (Hao et al. 2019)

This method learns embeddings for concepts (types) as well as entities. Specifically they identify two views: the ontology-view which models concepts and the meta-relations that exists between them; and the instance-view which models entities and their relations. They model the relationship between these two views in one of two ways: by cross-view grouping (CG); or by cross-view

transformation (CT). In the first case concepts and entities are embedded in the same space, and a loss-term is introduced to force entities to be close to their corresponding concepts. In the second case (CT) concepts and entities are embedded in separate spaces that are nonetheless aligned by a non-linear transformation with a corresponding loss term. An additional variant is introduced where the ontology-view is made "hierarchy-aware" by also modeling the 'subclass_of' meta-relation.

A [Modified] Joint Knowledge Graph Embedding Model for Concepts and Instances (JECI[++]) (Zhou et al. 2020; P. Wang and Zhou 2020)

This too is a framework for embedding both concepts and entities. In this work the hierarchy of concepts is used to construct 'simplified concepts' which are the concepts at the leaves of the concept tree. If a dataset links an entity to both a concept and its parent-concept, only the link to the concept that is lowest in the tree, i.e. the simplified concept, is kept.

When embedding an entity, this method computes a context vector from which the embedding is predicted. The context vector is a weighted sum of the embeddings of that entity's neighbors in the Knowledge Graph. Circular convolution between the context vector and a concept embedding is used to predict the location of the concept's sub-concept that this entity is a part of. This is performed recursively until there are no more sub-concepts, then a final convolution predicts the entity's embedding rather than a further sub-concept.

Overview

Each method that incorporates type-information does so with a combination of three techniques: using type-information to construct the entity-embeddings; using type-information to directly influence the score assigned to each triple; or using type-information to modify the optimization objective.

What stands out is that many of these methods are already quite complex and are not always compared to simple baselines, making it unclear what the frame of reference should be when judging how well these methods are performing. This seems particularly true for those methods that use type-information to construct the entity-embeddings.

3
Methodology

This chapter describes and motivates the various ways in which we incorporate type information. Some methods are original designs, for others we take inspiration from previous work. The methods are described using both the notation introduced in the previous chapters, and using diagrams depicting the structure of the model (for example, a typical geometric model is depicted in Figure 3.1).

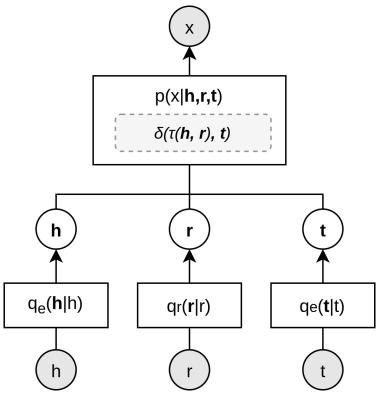


Figure 3.1: Example of model diagram. The circles represent random variables, gray circles are observed, white circles unobserved. The boxes and arrows represent modeling decisions, for example here we model the probability of x as being dependent on the embeddings $\mathbf{h}, \mathbf{r}, \mathbf{t}$.

3.1 Objectives

The main objective is to answer the research question as given in Chapter 1:

> ***(RQ1)*** To what extent can type-information improve link-prediction performance? And, ***(RQ2)*** what method of incorporating the type-information is most effective?

However, while answering our research question it might be useful to keep in mind some additional objectives. To maximize the utility of our investigations, we will keep in mind three secondary objectives, which are described below. Note that our methodology is not directly aimed at addressing these, but all things being equal, will try to nonetheless.

Link Prediction as a proxy for general purpose embedding

Knowledge Graph Embeddings may be used for a variety of downstream tasks such as: relation extraction, question answering, or recommender systems (Q. Wang et al. 2017b)[1]. It will be useful to keep in mind how the different ways we incorporate type-information will affect these kinds of use cases.

[1] "Knowledge Graph Embedding: A Survey of Approaches and Applications" DOI: 10.1109/TKDE.2017.2754499

Anonymous entity embedding

Using type information can allow us to embed entities that were not seen during training, as long as we know its type. To obtain such embeddings we need to model our embeddings as being conditioned on the types of the entity, i.e. $\mathbf{E}_i | C_{i,s} = c_{i,s}$. Of particular interest is any model that accomplishes this without giving up the

possibility of learning a specific embedding for the entities we do see during training.

3.2 *Using types to estimate prior probability of links*

The first method to be included in our experiments is the type-based prior introduced by Ma et al. in their paper on TransT (described in Section 2.2). It gives us an estimation of $p(h|r,t)$, $p(r|h,t)$, and $p(t|h,r)$. It can easily be used in combination with any model to re-weight the scores assigned to each triple. When used with a model that gives the likelihood of a triple, we can use the priors to get approximated posteriors $p(h|r,t,x)$, $p(r|h,t,x)$, and $p(t|h,r,x)$.

By directly influencing the prediction for links, this approach is aimed at solving the primary objective. It does not address either of the additional objectives.

We do not include all configurations of TransT that Ma et al. propose. They distinguish between the configurations 'type information', 'multiple vectors' and 'multiple+type'. In the first configuration the type-based prior is used to reweigh the scores as described above. The second configurations refers to the representation of entities by multiple vectors. The number of vectors to be learned for each entity is based on the available type information (see Section 2.2). The last configuration combines the first two. We include only the 'type information' configuration, because any performance gain of the other configurations cannot easily be attributed to use of type-information, since it could also be attributed to the increase in the number of parameters.

Type-linkprior only

This method evaluates the type-based prior on its own. We use only the score given by the prior to rank which triples are most likely to exist. This gives us an indication of how informative the prior is, as well as a way to compare the datasets on the relative richness of their type information. This configuration was not included in the experiments of Ma et al.

Type-linkprior+ScoringModel

This method combines an existing ScoringModel, i.e. a model that already scores each triple, with the type-based prior. The final scores are those assigned by the ScoringModel multiplied by the estimated prior probability. When the ScoringModel is TransE we obtain the configuration from which TransT derives its name. This method is equivalent to Ma et al.'s 'type information' configuration.

3.3 Using types to construct the entity-embeddings

Another way to use type information is to let each entity's set of types influence what the embedding of those entities will be. We experiment with three different kinds of embedders each with their own way of letting the types influence the entity-embeddings. This type of model is displayed in Figure 3.2.

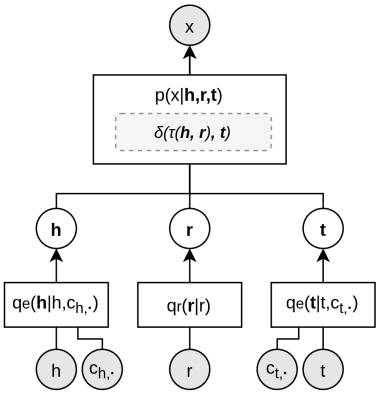


Figure 3.2: Diagram of model that utilizes types to construct entity-embeddings.

Type-mean embedder

This embedder is perhaps the simplest way of creating an entity's embedding from its types. Its main purpose is to serve as a baseline. It learns embeddings for the types, and models an entity's embedding as the average of that entity's type embeddings, i.e.:

$$\mathbf{C}_s|\theta_s \doteq \theta_s \tag{3.1}$$

$$\mathbf{E}_i|c_{i,1}, c_{i,2}, \ldots, c_{i,S} = \frac{\sum_{c_{i,z}} \mathbf{c}_z}{\sum_{c_{i,z}} 1} \tag{3.2}$$

where θ_s are the model parameters that provide the point estimate for the s-th type's embedding.

Variant: adding entity offset embeddings. This variant learns an additional vector for each entity that we add to the mean of the type-vectors.

$$\mathbf{E}_i|\epsilon_i, c_{i,1}, c_{i,2}, \ldots, c_{i,S} = \frac{\sum_{c_{i,z}} \mathbf{c}_z}{\sum_{c_{i,z}} 1} + \epsilon_i \tag{3.3}$$

This addition could allow for more accurate embeddings for the entities, since two entities with the same types can now be specialized.

This method could also be able to satisfy one of our secondary objectives. Specifically if the entity-offset is randomly dropped-out for some batches during training, it could allow us to embed both seen and unseen entities. Unseen entities are embedded by forgoing the entity-specific offset. Furthermore the inclusion of the type-information in the entity-embeddings themselves may benefit downstream tasks.

Type-attentive embedder

This embedder improves upon the type-mean embedder by learning to weight the type-embeddings differently for each entity. We do so with an attention mechanism:

$$q_i = (\epsilon_i)^\top \tag{3.4}$$

$$k_i = v_i = (\theta_s|\mathcal{C}_{i,s})^\top \tag{3.5}$$

$$\mathbf{E}_i \doteq \text{attn}(q_i, k_i, v_i) \tag{3.6}$$

where ϵ_i and θ_s are entity- and type-specific model parameters respectively.

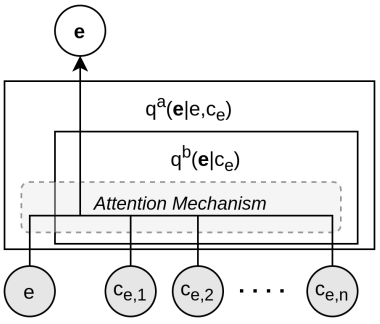


Figure 3.3: Diagram of the type-attentive embedder, depicting the two variants q^a which includes the entity e in the keys and values, and q^b which does not.

Variant: adding the entity-vector to the keys and values. In this variant we allow the entity-vector to be included in the weighted average.

$$k_i = v_i = (\epsilon_i; \theta_s | c_{i,s})^\top \tag{3.7}$$

This variant allows further specialization for each entity.

Mutual information loss. To encourage the inclusion of type information in the final entity-embeddings, we introduce an additional optimization objective. Taking inspiration from Hjelm et al. 2018[2], we employ a mutual information based secondary objective.

[2] "Learning Deep Representations by Mutual Information Estimation and Maximization" URL: https://openreview.net/forum?id=Bklr3j0cKX

3.4 Using types to provide additional supervision

Type-embedprior embedder

The main intuition motivating this model is to think of types as distributions over entities. The most straightforward way to translate that intuition into a model, would be to sample from type-distributions to obtain our entity embeddings. However, each entity may have multiple types, and it is unclear how we could tractably sample from a set of type-distributions.

To avoid this difficulty, we therefore do not sample the entity embeddings from the type distributions, but instead learn independent entity embeddings that are nonetheless likely under their types' distributions.

$$\mathbf{E}_i | \epsilon_i \doteq \epsilon_i \tag{3.8}$$

$$\mathbf{C}_s | \mu_s, \sigma_s \sim N(\mu_s, \sigma_s) \tag{3.9}$$

To accomplish this, we introduce a second optimization objective. Besides the objective to maximize the likelihood of the data, we now also wish to maximize the likelihood of the entities under the distributions of their types, and minimize the likelihood of entities under the distributions of other types[3]:

$$\arg\max_{\epsilon,\mu,\sigma} \quad \prod_{C_{i,s}} f_{\mathbf{C}_s}(\mathbf{e}) - \prod_{\neg C_{i,s}} f_{\mathbf{C}_s}(\mathbf{e}) \tag{3.10}$$

where $f_{\mathbf{C}_s}$ is the probability density function of $\mathbf{C}_s$.

[3] See 4.1 for more information on how we implement this.

Relation to other work. There exists an interesting parallel between the design of this embedder and the work of Hao et al. (described in Section 2.2). We can think of our model as a probabilistic reinterpretation of the same ideas. They too use a modified optimization objective, however their additional loss is not the likelihood of an entity under their types' distribution. Instead it is merely the distance between an entity embedding and its types' deterministic embeddings.

Hao et al. also distinguish between two variants, one where the distance is calculated directly to the type vectors, and another where a transformation is applied to the type vectors first. The variant with a transformation can also be reinterpreted probabilistically, if the transformation is implemented as a normalizing flow.

Overview

Each of the methods we described use a strategy that we previously identified in the literature. We can see an overview of the methods, the kinds of input information they have access to, and the strategies that they employ in Table 3.1.

method	*inputs*		*modification*
	ϵ_i	$c_{i,\mathbf{s}}$	
non-semantic	✓		-
type-linkprior-only		✓	triple score
type-linkprior	✓	✓	triple score
type-mean		✓	entity-embeddings
type-mean+	✓	✓	entity-embeddings
type-attentive	✓	✓	entity-embeddings
type-attentive-alt	✓	✓	entity-embeddings
type-embedprior	✓	✓	optimization objective

Table 3.1: Overview of each method, including the kinds of input (entity, type) each method has, as well as the strategy used to incorporate type-information.

4
Experiments

This chapter will describe what experiments have been performed to answer the research question formulated in Chapter 1, and give the results of those experiments. The description of the experiments includes implementation details of the methods described in the previous chapter, as well as the strategy employed for the hyper-parameter search. The results are include both performance metrics resulting from the hyper-parameter searches, and analyses of the methods aimed at confirming the methods are working as intended.

4.1 Implementations

We make the implementations of the methods described in Chapter 3 and the experiments described here freely available in the newly created SemKGE (Semantic KGE) repository,[1] which was developed as a plugin to the LibKGE library[2] (Broscheit et al. 2020)[3].

[1] github.com/sfschouten/semantic-kge

[2] github.com/uma-pi1/kge

[3] "LibKGE - A knowledge graph embedding library for reproducible research" DOI: 10.18653/v1/2020.emnlp-demos.22

Type-linkprior

This method is implemented such that the variables $\lambda_{head}, \lambda_{relation}, \lambda_{tail}$ from Formulas 2.23, 2.24, 2.25 are learnable during training. As such, we introduce a hyper-parameter to control whether they are learned or not. If this hyper-parameter is set to true, the hyper-parameters that are normally used to set these values, now merely set their initial value.

Type-attentive

The optimal learning rate for training a method like TransE can be quite high (Ruffinelli et al. find an optimal learning rate of 0.25327). Since the attention mechanism in the type-attentive embedder is the only part of the model that contains fully connected layers (which generally require lower learning rates), we tune the learning rate separately for that component.

Type-embedprior

This method's additional optimization objective maximizes the difference in probability density assigned to an entity-embedding

by that entity's types' distributions, and the density assigned by the other types' distributions. We implement this by drawing one negative type sample for each type of each entity.

Additional losses with constrained optimization.

Optimizing with multiple objectives is often done by creating a weighted sum of loss terms. But this strategy may be flawed, recently it was shown that this approach may fail even on small toy problems (Degrave and Korshunova 2021)[4]. A potential solution is to construct a constrained optimization problem, where we identify one of our loss terms as the primary objective and incorporate the others as constraints. Using the Modified Differential Method of Multipliers (Platt and Barr 1987)[5] allows us to utilize gradient descent with such a constrained optimization problem. This method has been employed successfully with Variational Autoencoders (Rezende and Viola 2018)[6], but should lead to more easily tunable hyperparameters regardless of the application.

As such the type-attentive's mutual information loss, and the type-embedprior's type-distribution density loss are both incorporated as constraints.

[4] *How We Can Make Machine Learning Algorithms Tunable* URL: https://www.engraved.blog/how-we-can-make-machine-learning-algorithms-tunable/

[5] "Constrained Differential Optimization" URL: http://papers.nips.cc/paper/4-constrained-differential-optimization

[6] *Taming VAEs* URL: http://arxiv.org/abs/1810.00597

4.2 *Datasets*

The datasets we use for our experiments are FB15K-237 (Toutanova et al. 2015)[7], WN18RR (Dettmers et al. 2018)[8], CoDEx-S and CoDEx-M (Safavi and Koutra 2020)[9]. The first two are well known benchmarks for Link Prediction (Ruffinelli et al. 2019)[10]. They are the successors of the FB15K and WN18 datasets, which had been found to have test leakage (Rossi et al. 2021)[11]. The latter are two variants of a recently introduced set of benchmarks called CoDEx. Safavi and Koutra show that CoDEx covers more diverse and interpretable content than FB15K-237, and that CoDEx is a more difficult . Statistics for each of these datasets can be seen in Table 4.1.

[7] "Representing Text for Joint Embedding of Text and Knowledge Bases" DOI: 10.18653/v1/D15-1174

[8] "Convolutional 2D Knowledge Graph Embeddings" URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17366

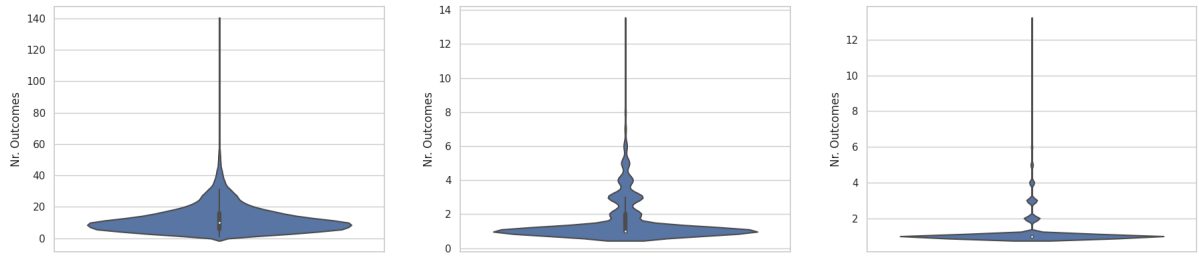
[9] "CoDEx: A Comprehensive Knowledge Graph Completion Benchmark" DOI: 10.18653/v1/2020.emnlp-main.669

[10] "You CAN Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings" URL: https://openreview.net/forum?id=BkxSmlBFvr

[11] "Knowledge Graph Embedding for Link Prediction: A Comparative Analysis" DOI: 10.1145/3424672

	entities	relations	triples			avg. types per entity	types in use	total types
			train	valid	test			
FB15K-237	14,541	237	272,115	17,535	20,466	12.422	3,865	4,054
WN18RR	40,943	11	86,835	3,034	3,134	1.000	4	4
CoDEx-S	2,034	42	32,888	1,827	1,828	1.619	502	3,443
CoDEx-M	17,050	51	185,584	10,310	10,311	1.253	1,505	3,443

Table 4.1: Dataset statistics.



(a) FB15K-237 (b) CoDEx-S (c) CoDEx-M

Figure 4.1: Distribution of number of types per entity across test portion of the datasets. The WN18RR dataset is omitted because it has exactly one type for each entity.

4.3 *Technical Details*

All experiments were run on the SURFsara Lisa GPU cluster[12]. The nodes we used had Nvidia GeForce 1080Ti GPUs with 11GB of VRAM.

[12] `userinfo.surfsara.nl/systems/lisa/description`

4.4 *Evaluating & Optimizing Link Prediction*

The Link Prediction task is evaluated as a ranking problem. Given a triple we know to be correct (h, r, t) (those triples in the validation or test set), we use our model to predict what the head or tail should be, i.e. $(?, r, t)$ and $(h, r, ?)$. We rank all other entities on their likelihood of being the missing head or tail. The ranking that the model predicts is evaluated by what ranks are assigned to the original correct head and tail.

Evaluation metrics used to quantify the quality of the rankings include: Mean Rank (MR), Mean Reciprocal Rank (MRR), and Hits@N. The Mean Rank is the mean of the ranks assigned to the correct triples. This metric gives a number between one and the number of entities, the lower the better. It is easily interpreted, but its sensitivity to outliers is causing it to be abandoned in favor of MRR (Rossi et al. 2021)[13]. The Mean Reciprocal Rank is the mean of multiplicative inverse of the ranks. It gives a number between 0 and 1, the higher the better. It is much less sensitive to outliers. Hits@N gives the percentage of correct triples with an assigned rank below N, the higher the better.

These metrics can be calculated in two different ways, they can be computed *raw* or *filtered*. This choice determines how we treat other predictions of heads/tails that also correct triples, that are not the triple for which we are currently doing head/tail prediction. If we report the *raw* metrics, we leave these other correct triples alone, if we report the *filtered* metrics we filter them out. The *filtered* metrics are generally preferred because we do not care about the model ranking other correct triples above the currently evaluated triple.

[13] "Knowledge Graph Embedding for Link Prediction: A Comparative Analysis" DOI: `10.1145/3424672`

Loss functions used to optimize for these ranking metrics need special attention, because the metrics themselves are not continuous. This is because these metrics require sorting the scores assigned to triples in order to compute their rank. So to optimize with gradient descent, we use separate loss functions as a proxy for these metrics.

One possibility is to model the truth of a triple as being sampled from a Bernoulli distribution.

$$X|\mathbf{h},\mathbf{r},\mathbf{t} \sim \text{Bernoulli}(\sigma(score(\mathbf{h},\mathbf{r},\mathbf{t}))) \tag{4.1}$$

Where σ is a function that maps the scores to $(0,1)$. The difference between the model distribution and the data distribution is then quantified as the Binary Cross Entropy (BCE) loss.

Alternatively we can think of the true triples as being sampled from a set of triples $\mathcal{S}_{(h,r,t)}$.

$$p(\boldsymbol{\mathcal{S}}_{(h,r,t)}) = \sigma\left(score(\mathbf{h}',\mathbf{r}',\mathbf{t}') \mid (\mathbf{h}',\mathbf{r}',\mathbf{t}') \in \boldsymbol{\mathcal{S}}_{(h,r,t)}\right)^{\top} \tag{4.2}$$

$$K^{+}|\boldsymbol{\mathcal{S}}_{(h,r,t)} \sim \text{Categorical}(|\boldsymbol{\mathcal{S}}_{(h,r,t)}|, p(\boldsymbol{\mathcal{S}}_{(h,r,t)})) \tag{4.3}$$

$$X^{(k')} = [k' = k^{+}] \tag{4.4}$$

Where σ is the softmax function, $S_{(h,r,t)}$ is the set of triples and $\mathbf{S}_{(h,r,t)}$ are their embeddings. In practice this set is based on a triple $(h,r,t) \in \mathcal{D}^{+}$, it includes that triple itself and a set of negative triples constructed from the original. When modeled this way the difference between the model and data distribution is quantified as the Categorical Cross Entropy loss.

Other loss functions like Margin Ranking and Squared error are sometimes also used. However, in contrast to the losses mentioned above, these others are optimal for none of the methods included in the hyper-parameter search performed by Ruffinelli et al.

Training approaches differ from each other by how batches of triples are constructed. Negative sampling is an approach that first samples triples from the training data, and then constructs set of negatives for each triple by corrupting one of the triple's components. Ruffinelli et al. implement two other approaches, 1vsAll, and KvsAll. The 1vsAll approach is similar to negative sampling but adds every possible way of corrupting a triple as a negative, even if the resulting triples occur in the training data. The KvsAll approach only corrupts either the head or tail, and contrary to 1vsAll, labels all that occur in the training data as positive, and only those that do not as negative.

4.5 Hyper-parameter Searches

Our main experiment is an extensive hyper-parameter search. For each of the methods described in Chapter 3, we use TransE as the underlying KGE model. The strategy employed for the hyper-parameter searches is based on the strategy used in Ruffinelli et al.

[14] "You CAN Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings" URL: https://openreview.net/forum?id=BkxSmlBFvr

2019[14]. Each search is based on 30 rounds of parameters drawn from a Sobol sequence (Sobol' 1967)[15]. Ruffinelli et al. follow this with 15 rounds of Bayesian optimization, but since they report that this has minimal effect we do not include these rounds.

[15] "On the distribution of points in a cube and the approximate evaluation of integrals"

Training

We use a Categorical Cross Entropy loss, minimizing the KL Divergence between the model and data distributions, and use negative sampling as our training strategy. These two decisions were shown by Ruffinelli et al. to be optimal for the TransE KGE model.

We use a 'shared' implementation of negative-sampling, meaning that the same negative samples are used for each triple in a batch. Note that it is ensured that a triple does not get itself as a negative.

An extensive overview of which hyper-parameters were tuned can be found in Appendix A

Results

The results of this hyper-parameter search can be seen in Tables 4.2 (MRR), 4.3 (MR), and 4.4 (Hits@10).

No search was done for the type-attentive embedder on the WN18RR dataset. Using the type-attentive embedder on that dataset is not productive given that the dataset has exactly one type per entity. We do include the type-attentive-alt method since its attention mechanism can still learn to weigh the type representation against the entity representation.

4.6 Analysis

Among the results are observations both expected and unexpected. As expected, we see a clear correlation between the performance of the type-linkprior-only model and the type-linkprior model. Overall we see a modest improvement over the non-semantic baseline.

However, among the type-mean and type-attentive models we see unexpected high performance for the simplest type-mean model. We also see the variants which allow entity-specific specialization underperform their counterparts. Both of these observations will require further analysis to understand.

The type-embedprior method induces a modest improvement in MRR for all but the CoDEx-M dataset.

Type-linkprior[-only]: hyper-parameters

In Table 4.5 we can see the hyper-parameters for the type-linkprior-only method and the values that obtained the highest MRR for each dataset. Because this method fails to obtain a positive MRR or Hits@10 for the WN18RR dataset, the hyper-parameter values found for that dataset do not reflect anything meaningful. For the

name	WN18RR	FB15K-237	CoDEx-S	CoDEx-M
non-semantic	0.226	0.308	0.354	0.306
type-linkprior-only	0.000	0.140	0.094	0.051
type-linkprior	0.191	0.324	0.354	0.284
type-mean	0.723	0.272	0.446	0.357
type-mean+	0.174	0.319	0.350	0.274
type-attentive	-	**0.615**	**0.612**	**0.477**
type-attentive-alt	**0.816**	0.298	0.341	0.456
type-embedprior	0.243	0.316	0.378	0.298

Table 4.2: Mean Reciprocal Rank obtained by each method for best set of hyperparameters.

name	WN18RR	FB15K-237	CoDEx-S	CoDEx-M
non-semantic	1774.06	176.93	**43.78**	**289.89**
type-linkprior-only	19581.08	686.44	290.47	4655.25
type-linkprior	4617.39	**158.69**	49.45	434.35
type-mean	9714.45	233.67	88.20	1959.57
type-mean+	5912.00	171.31	54.84	582.91
type-attentive	-	5374.72	78.17	984.77
type-attentive-alt	**1557.96**	188.45	54.46	967.30
type-embedprior	3185.04	171.81	46.87	571.06

Table 4.3: Mean Rank obtained by each method for set of hyperparameters with highest MRR.

name	WN18RR	FB15K-237	CoDEx-S	CoDEx-M
non-semantic	51.53	49.90	64.37	46.65
type-linkprior-only	00.00	21.91	18.56	7.85
type-linkprior	43.75	51.14	58.73	44.68
type-mean	72.33	40.89	64.50	47.21
type-mean+	42.55	49.02	57.85	42.22
type-attentive	-	**61.57**	**83.03**	**54.09**
type-attentive-alt	**82.23**	47.05	58.35	51.85
type-embedprior	63.25	49.57	62.62	45.05

Table 4.4: Hits@10 (%) obtained by each method for best set of hyperparameters.

other datasets we can see that the values found for the hyperparameters are similar. Furthermore, we can see that optimizing the λ parameters using gradient descent was not useful.

	WN18RR	FB15K-237	CoDEx-S	CoDEx-M
ρ	0.455	0.179	0.079	0.172
λ_{head}	0.548	0.877	0.919	0.862
$\lambda_{relation}$	0.780	0.026	0.122	0.095
λ_{tail}	0.746	0.805	0.714	0.717
learn_lambda	true	false	false	false

Table 4.5: The hyperparameters specific to the type-linkprior-only that were used to obtain the highest MRR.

Type-mean: performance

The type-mean method outperforms the non-semantic baseline for three out of four datasets. This is surprising as this is the simplest method we test. By only learning separate vectors for each type, rather than each entity, it uses less parameters than the non-semantic baseline. The performance is most striking for the WN18RR dataset, where it gets an MRR score of 0.7233 . Since this dataset has only 4 types, and 11 relations, the number of vectors it uses to obtain this score is only 15 (!). The entity's representation is calculated by averaging the representations of that entities types. Because WN18RR has only one type for each entity, we are effectively substituting each entity with its type.

To understand why this results in high predictive performance, we can take a deeper look at the statistics of the WN18RR dataset. In particular we look at each relation's type-statistics, i.e. how many links exist between each type for that relation. These statistics are displayed in Table 4.6. In this table we can see that almost all of

	NN	*VB*	*JJ*	*RB*
NN	27826 (79.97%)	38 (00.11%)	11 (00.03%)	4 (00.01%)
VB	78 (00.22%)	6791 (19.52%)	9 (00.03%)	11 (00.03%)
JJ	11 (00.03%)	7 (00.02%)	0 (00.00%)	0 (00.00%)
RB	7 (00.02%)	3 (00.01%)	0 (00.00%)	0 (00.00%)

Table 4.6: Type statistics for most frequently occurring relation in WN18RR.

this relation's links are between a noun (NN) and another noun, or between a verb (VB) and another verb. This means that using only this information we can rank all potential links between two nouns first, and rank links between two verbs after that. The other relations in the dataset have similar statistics, with one or two combinations of types being far more frequent than the rest.

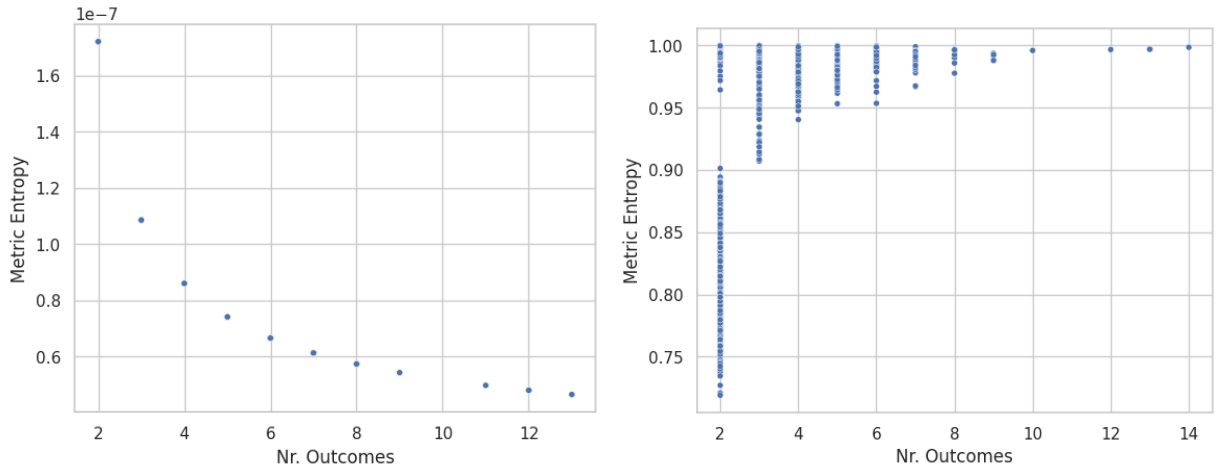
Evidently for WN18RR statistics such as those found in Table 4.6 are informative enough to obtain very high performance on the Link Prediction task. However this does come at a cost, we now only have an embedding for each type, not each entity. If Link Prediction is the only concern then this might be a good tradeoff,

but if the goal was to have a general-purpose knowledge graph embedding, it likely is not.

This method seems to also do very well for the CoDEx datasets, although not as well as for WN18RR. Type-mean does not perform well for FB15K-237. This is unsurprising when we look at the number of types per entity for each dataset. An average 1.62 and 1.25 types per entity for the CoDEx-S and CoDEx-M datasets puts them only just above the 1.0 of WN18RR. But FB15K-237 has an average of 12.42 types per entity (see Table 4.1). With CoDEx most entities still have one type, like on WN18RR, but with FB15K-237 this is not the case. And without (mostly) a single type per entity the type-statistics cannot be represented like is possible for WN18RR.

Type-mean+'s performance is much more similar to the non-semantic baseline. We see a modest improvement in MRR for FB15K-237 and about equal MRR for the CoDEx-S dataset. For the WN18RR and CoDEx-M datasets this method show a decrease in MRR. It seems that this method's additional entity-specific representation creates an inductive bias that keeps it from learning the same behavior we see for the type-mean method.

Type-attentive: attention weight distributions

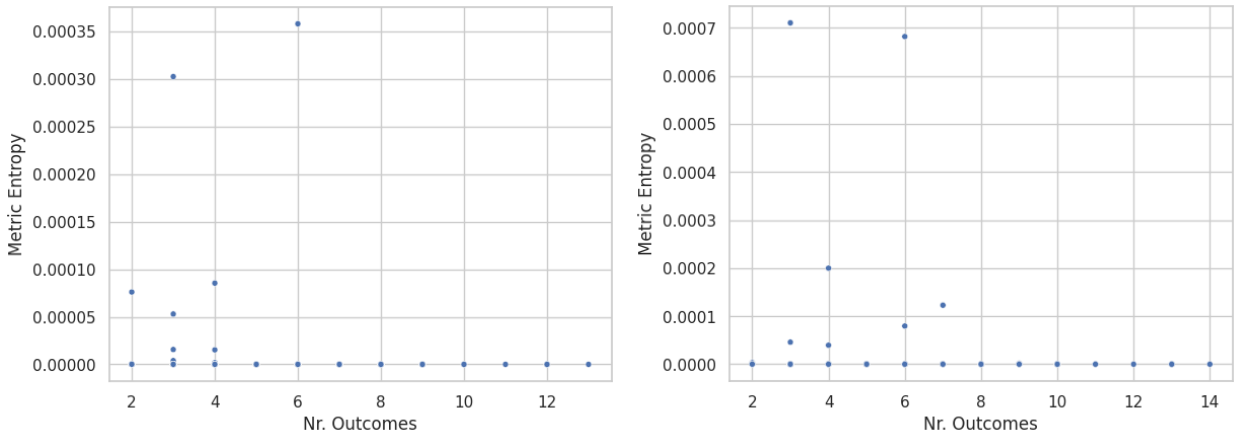


(a) Type-attentive

(b) Type-attentive-alt

Figure 4.2: Scatter plot of metric entropy over the number of outcomes for the CoDEx-S dataset.

The intuition behind the type-attentive method was to have a weighted average between type representations rather than a simple mean. Like the type-mean method we see unexpectedly high MRR for the type-attentive method. We see even higher performance for this method, now for FB15K-237 too. In all likelihood because of similar reasons as the type-mean's high performance. This method can do even better at representing the type-statistics by learning to pick the one type that is most representative of each entity using the attention mechanism. To confirm this is the case we look at the attention weights as a probability distribution, and measure its Shannon Entropy to determine if most attention is paid



(a) Type-attentive

(b) Type-attentive-alt

Figure 4.3: Scatter plot of metric entropy over the number of outcomes for the CoDEx-M dataset.

to one or a few types or if the attention is fairly spread out.

In Figure 4.2 we can see a scatter plot of the Metric Entropy[16] of the attention weights for the type-attentive and type-attentive-alt methods when applied to the CoDEx-S dataset. We can see that the type-attentive method is assigning very low entropy weights to the type-vectors, regardless of the number of types of the entity[17]. This confirms that the the type-attentive method is choosing one type for each entity, using the type-statistics only to achieve high predictive performance. The entropy for the type-attentive-alt variant looks rather different, its attention distributions are relatively high entropy. This time the method seems to operate more like intended, taking in information from a variety of its types. Unfortunately, with an MRR that is slightly below the non-semantic baseline, this does not result in an increase in performance.

In Figure 4.3 we can see the same type of plot but for the CoDEx-M dataset. Here we see that entropy is low for both of the method's variants, which explains why performance for both is similar. It seems that in rare cases even the type-attentive-alt method can uncover this type-statistics only mode where it attends only to a single representative type.

[16] Metric Entropy is a normalized variant of the regular Shannon Entropy. It is obtained by dividing the Shannon Entropy by the information length (in bits, i.e. $log_2(|X|)$ where $|X|$ is the number of possible outcomes of the random variable).

[17] Note that for the type-attentive method the number of outcomes is often one (because that entity has only one type). These cases are not displayed in the plot, as Metric Entropy is undefined in those cases.

	FB15K-237	CoDEx-S	CoDEx-M
lr (default)	$3.6 \cdot 10^{-2}$	$1.7 \cdot 10^{-4}$	$3.6 \cdot 10^{-2}$
lr (attn)	$3.9 \cdot 10^{-1}$	$6.6 \cdot 10^{-1}$	$3.9 \cdot 10^{-1}$

Table 4.7: Type-attentive learning rates for runs with best MRR.

Type-attentive: hyper-parameters

In Table 4.7 we can see that the attention mechanism's learning rate is tuned to a higher value than we see for the rest of the model. The original intent of the separately tuned learning rate was to allow the attention mechanism to use a lower learning rate, not higher. It is possible that these learning rates play a role in reaching the low

entropy modes. The mutual information hyper-parameter was set below zero (disabled) for each dataset.

Type-embedprior: learned distributions

The intuition behind this method is that entities of the same type are similar, and should therefore be close to each other in the embedding space. Specifically, we construct distributions for each type and want the embedding of entities of those types to be likely under those distributions, and only those distributions. To measure how well the learned distributions match this intuition, we now look at the degree of overlap between distributions. We hope that the distributions of types that do not have any entities in common also do not overlap in the embedding space. We quantify the extent to which types have entities in common with the Jaccard Index, and the degree of overlap with the Bhattacharyya Coefficient.

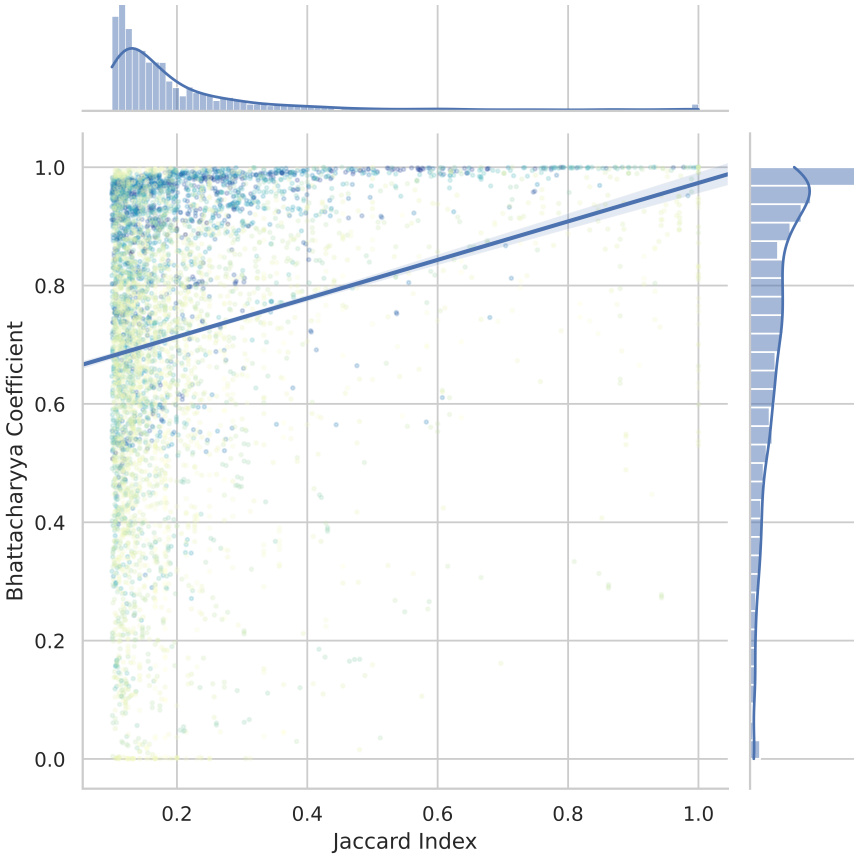


Figure 4.4: Combination of scatter plot, univariate KDE curves, and a linear regression fit, depicting the relation between the Jaccard Index of type pairs (as measured when considering types as sets of entities), and the Bhattacharyya Coefficient between the learned type distributions of said type pairs.

A scatter plot of type-pairs with these two quantities on the x- and y-axis respectively can be seen in Figure 4.4. This figure also shows a linear regression fit, and displays the cardinality of the union of the two type sets using the color of the points, where bright yellow are the lowest and dark blue the highest cardinality[18]. We see a clear correlation between the two quantities, but also see that it is mostly one-way. In an ideal scenario the points in the plot would form a diagonal line, showing perfect correlation. Instead,

[18] To reduce the total number of points and improve the utility of the plot, type-pairs with a Jaccard Index below 0.1 were left out. As were type-pairs with fewer than 10 entities between them, and types paired with themselves.

we observe that most points occur above this hypothetical line. This indicates that types with a lot of entities in common also overlap in the embedding space, but types that do not have many entities in common, may still overlap considerably in the embedding space. We also observe that the type-pairs with the most entities between them exhibit more overlap in the embedding space, which is to be expected.

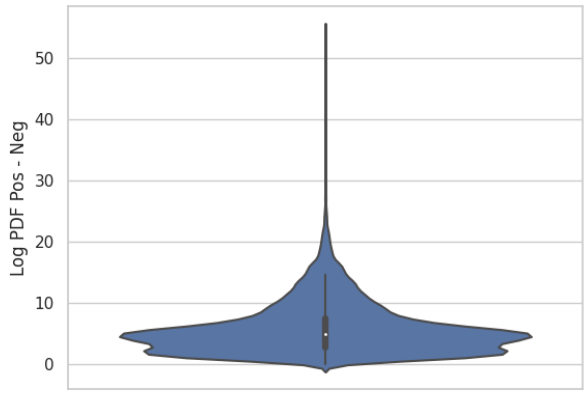
Type-embedprior: hyper-parameters

The type-embedprior embedder seeks to make the entity-embeddings likely under their type-distributions. To do this it uses a minimum constraint on log-density assigned by the correct types minus the log-density assigned by other types. In other words, an entity's true types must assign at least some number of times more density to the entity's embedding, than other types do.

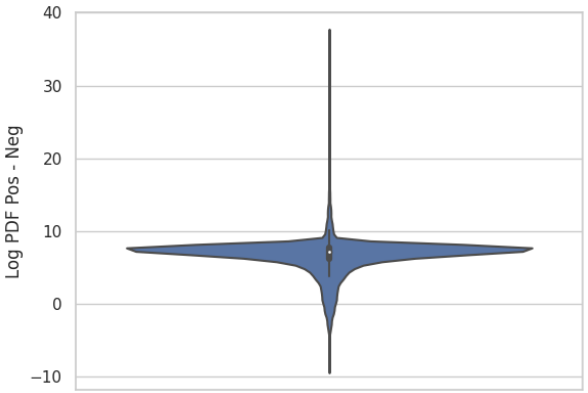
	WN18RR	FB15K-237	CoDEx-S	CoDEx-M
min log-density Δ	6.982	7.470	7.470	6.588

Table 4.8: Minimum difference in log-density required of entity-embedding under the distributions of that entities types and set of sampled negative types (for the best performing model from the hyper-parameter search).

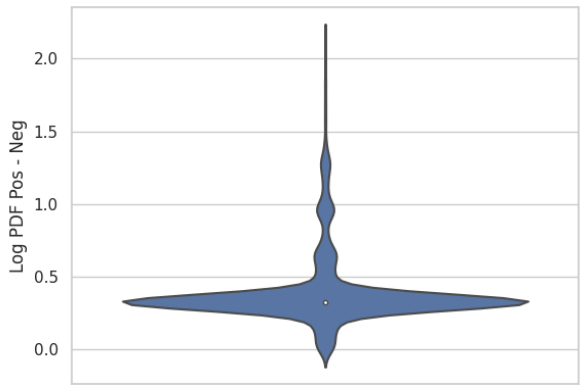
In Table 4.8 we can see the value of the hyper-parameter that governs the minimum difference in log-density required, for the best run from the hyper-parameter search. We can see that for each dataset a similar value is selected. However, as can be seen in Figure 4.5, the actual difference in log-density that is obtained by the best performing checkpoint for each dataset does differ. For FB15K-237 and WN18RR we can see that the constraint is satisfied for a lot of entities but not all. For the CoDEx datasets we can see that the constraint is violated for all entities.



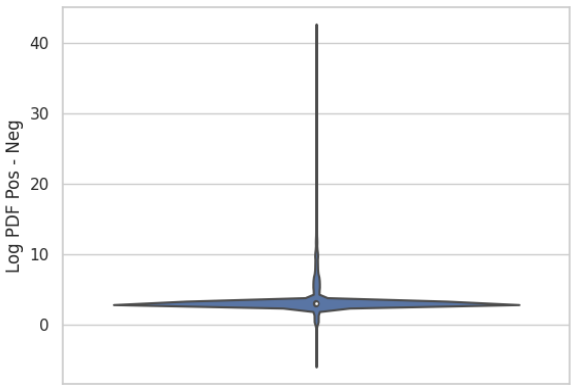
(a) FB15K-237



(b) WN18RR



(c) CoDEx-S



(d) CoDEx-M

Figure 4.5: Difference in log-density between positive and negative types.

5
Conclusion & Discussion

This chapter will outline the conclusions that may be drawn from the results of the experiments. They will be followed by some additional remarks, either detailing what else we may conclude from the results, or discussing how the results relate to previous investigations.

Answering the Research Question

> *(**RQ1**)* To what extent can type-information improve link-prediction performance? And, *(**RQ2**)* what method of incorporating the type-information is most effective?

By embedding a type for each entity instead of the entity itself – which was an unintended effect of both the type-mean and type-attentive methods for some datasets – we can get Link Prediction performance that is well beyond what seems possible otherwise. The improvement in MRR against the non-semantic baseline ranges from 261% (for WN18RR) to 56% (for CoDEx-M). From this we can conclude that type-information can greatly improve Link Prediction performance.

The second question asks what method is most effective. When judged by our primary objective, which was Link Prediction performance, it is clear that for most datasets the type-attentive method is best. By representing entities by their most representative types, the method can exploit the type-information to obtain great Link Prediction performance. The results on WN18RR, where the type-attentive-alt variant results in the highest MRR, shows that including the entity in the keys and values of the attention mechanism may yield even greater performance. Although the results of this method on the other datasets shows that this variant is less likely to find the low-entropy mode that makes the high MRR possible in the first place. If a way can be found to reliably find the low-entropy mode (see the Future Work section below) this variant may be superior.

Further Remarks

Additional Objectives

In Section 3.1 we identified two additional objectives, the first of which was the ability to utilize Knowledge Graph Embeddings for downstream tasks. For that objective the type-attentive method seems less suited. For a lot of these tasks it is important that we actually learn a separate embedding for each entity, but as discussed previously, when the type-attentive method performs well it effectively only learns type-embeddings. Thus, despite a much more modest gain in MRR, the type-embedprior seems the better option when suitability for downstream tasks is important.

The other objective we identified was the ability to embed entities using their type(s) that were not seen during training. The type-mean can certainly do this, but that is all it can do, in essence it treats all entities as unseen entities, using only their type(s) for the embedding. Type-attentive still uses an entity representation to calculate the attention weights, so embedding unseen entities would be non-trivial. It might be possible to drop out the entity-specific representation during training so the model may learn to deal with this case effectively. Using the type-embedprior to embed unseen entities is also non-trivial, but for this method there is no mechanism at all to decide how to combine the distributions of the entity's types. Only if the unseen entity has just one type (or we know which is most representative) can we use the type-embedprior by simply sampling from that type's distribution.

Comparing experimental results with Ma et al. 2017[1]

[1] "Transt: Type-based multiple embedding representations for knowledge graph completion"

In Ma et al. 2017 TransT's performance for Link Prediction is tested on the outdated FB15K and WN18 datasets (see Section 3.2). Furthermore, it is not clear that the baselines against which TransT was compared were run using the same experimental setup. From their code-base it seems that the baselines' performance came from previously reported numbers.

Both of these shortcomings are addressed in our experiments. We report the performance of the TransT configuration which Ma et al. refer to as 'type information' (see Section 2.2). We show that this method only outperforms the baseline on FB15K-237. Ma et al. report an improvement in Mean Rank and Hits@10 on the WN18 dataset, we do not see the same for the WN18RR dataset.

A prior on links: Type-linkprior(-only) vs. Type-attentive

When the type-attentive model learns a low-entropy distribution over types – effectively choosing one type per entity as the most-representative – it no longer truly learns to embed entities, instead it only learns to embed types. The type-linkprior and type-linkprior-only methods also do not try to incorporate type-

information in entity-embeddings, but instead (re-)weigh the likelihood of triples being true. However, of these two methods the type-attentive embedder is clearly the superior in terms of Link Prediction performance (see Table 4.2). If we wanted to model a type-based prior probability over links, then a separate KGE that uses the type-attentive embedder would be a far better choice than using the type-linkprior.

Future Work

Taming the type-attentive[-alt] embedder

In its current form, the type-attentive embedder is somewhat unpredictable, it exhibits substantially different behavior depending on whether it reached a low- or high-entropy mode (see Section 4.6). Say we want to use the type-attentive embedder in its low-entropy mode, then currently we have no way of making sure that this is what it learns. Only when a particularly high learning rate is chosen for the attention mechanism does the model occasionally reach a well performing low-entropy state.

The most obvious way to address this problem is to constrain the entropy of the distributions produced by the attention mechanism directly. By doing this we can: produce high-entropy attention distributions to test if the originally intended behavior of the type-attentive embedder is useful; and produce low-entropy attention distributions to reliably reproduce the observed 'most-representative type per entity' behavior.

Combining type-attentive and type-embedprior

Given the striking Link Prediction results of the type-mean and type-attentive models, it would make sense to explore ways of using their strengths while addressing their shortcomings. The main shortcoming of these methods is that they do not allow for learning of entity-specific embeddings. Instead they reduce the problem to learning only type-embeddings, and in the case of the type-attentive model, learning a most-representative type per entity.

To improve upon this we might take inspiration from the type-embedprior. It tries to learn both type-embeddings and entity-embeddings, the main intuition being that encouraging entities of the same type to be close to each other in the embedding space may be beneficial. However, in practice this resulted in only a modest improvement (see Section 4.5). It is possible that this is because there was not enough 'signal' available for the type-distributions. This would explain why the added benefit we observe is so much less than that observed by Hao et al. 2019[2], who supervise the type-embeddings with links from an ontology (meta-relations).

Instead of using links from an ontology – which are available only for select datasets – we could use the regular links from the knowledge graph, using the type-attentive embedder to obtain a

[2] "Universal Representation Learning of Knowledge Bases by Jointly Embedding Instances and Ontological Concepts" DOI: 10.1145/3292500.3330838

most-representative type per entity with which to translate entity-to-entity links into type-to-type links. These links could be used to supervise either the distances between the type-distributions' means μ, or to supervise the type-distributions using distance metrics like those used by He et al. 2015[3].

Another version of this would forego the probabilistic aspect of the type-embeddings entirely. Thereby obtaining a method even closer to that of Hao et al. (the only difference being the use of links from the knowledge graph, rather than the ontology).

[3] "Learning to Represent Knowledge Graphs with Gaussian Embedding" DOI: 10.1145/2806416.2806502

Extending to other KGE methods

Our experiments have been exclusively in combination with TransE. It will be useful to perform experiments in combination with other KGE methods. Investigating the generalization of our methodology to non-geometric KGE methods is of particular interest.

Bibliography

Bordes, Antoine et al. (2013). "Translating Embeddings for Modeling Multi-Relational Data". In: *Advances in Neural Information Processing Systems* 26. URL: https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html (visited on 03/12/2021).

Broscheit, Samuel et al. (Oct. 2020). "LibKGE - A knowledge graph embedding library for reproducible research". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, pp. 165–174. DOI: 10.18653/v1/2020.emnlp-demos.22. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.22 (visited on 05/25/2021).

Degrave, Jonas and Ira Korshunova (Jan. 30, 2021). *How We Can Make Machine Learning Algorithms Tunable*. Engraved. URL: https://www.engraved.blog/how-we-can-make-machine-learning-algorithms-tunable/ (visited on 06/28/2021).

Dettmers, Tim et al. (Apr. 25, 2018). "Convolutional 2D Knowledge Graph Embeddings". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. Thirty-Second AAAI Conference on Artificial Intelligence. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17366 (visited on 05/27/2021).

Guan, Niannian, Dandan Song, and Lejian Liao (Jan. 15, 2019). "Knowledge graph embedding with concepts". In: *Knowledge-Based Systems* 164, pp. 38–44. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2018.10.008. URL: https://www.sciencedirect.com/science/article/pii/S0950705118304945 (visited on 03/12/2021).

Guo, Shu et al. (July 2015). "Semantically Smooth Knowledge Graph Embedding". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. ACL-IJCNLP 2015. Beijing, China: Association for Computational Linguistics, pp. 84–94. DOI: 10.3115/v1/P15-1009. URL: https://www.aclweb.org/anthology/P15-1009 (visited on 03/12/2021).

Hao, Junheng et al. (2019). "Universal Representation Learning of Knowledge Bases by Jointly Embedding Instances and Ontological Concepts". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19.

Anchorage, AK, USA: Association for Computing Machinery, pp. 1709–1719. ISBN: 9781450362016. DOI: 10.1145/3292500.3330838. URL: https://doi.org/10.1145/3292500.3330838.

He, Shizhu et al. (Oct. 17, 2015). "Learning to Represent Knowledge Graphs with Gaussian Embedding". In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM '15. New York, NY, USA: Association for Computing Machinery, pp. 623–632. ISBN: 978-1-4503-3794-6. DOI: 10.1145/2806416.2806502.

Hitzler, Pascal et al. (Dec. 2012). *OWL 2 Web Ontology Language Primer (Second Edition)*. W3C.

Hjelm, R. Devon et al. (Sept. 27, 2018). "Learning Deep Representations by Mutual Information Estimation and Maximization". In: International Conference on Learning Representations. URL: https://openreview.net/forum?id=Bklr3j0cKX (visited on 07/02/2021).

Krompaß, Denis, Stephan Baier, and Volker Tresp (2015). "Type-Constrained Representation Learning in Knowledge Graphs". In: *The Semantic Web - ISWC 2015*. Ed. by Marcelo Arenas et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp. 640–655. ISBN: 978-3-319-25007-6. DOI: 10.1007/978-3-319-25007-6_37.

Lv, Xin et al. (Oct. 2018). "Differentiating Concepts and Instances for Knowledge Graph Embedding". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. EMNLP 2018. Brussels, Belgium: Association for Computational Linguistics, pp. 1971–1979. DOI: 10.18653/v1/D18-1222. URL: https://www.aclweb.org/anthology/D18-1222 (visited on 03/12/2021).

Ma, Shiheng et al. (2017). "Transt: Type-based multiple embedding representations for knowledge graph completion". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pp. 717–733.

Noy, Natasha et al. (Apr. 1, 2019). "Industry-Scale Knowledge Graphs: Lessons and Challenges: Five Diverse Technology Companies Show How It's Done". In: *Queue* 17.2, Pages 20:48–Pages 20:75. ISSN: 1542-7730. DOI: 10.1145/3329781.3332266.

Paulheim, Heiko (2017). "Knowledge graph refinement: A survey of approaches and evaluation methods". In: *Semantic Web* 8. 3, pp. 489–508. ISSN: 2210-4968. DOI: 10.3233/SW-160218. URL: https://doi.org/10.3233/SW-160218.

Platt, John C. and Alan H. Barr (Jan. 1, 1987). "Constrained Differential Optimization". In: *Proceedings of the 1987 International Conference on Neural Information Processing Systems*. NIPS'87. Cambridge, MA, USA: MIT Press, pp. 612–621. URL: http://papers.nips.cc/paper/4-constrained-differential-optimization (visited on 06/01/2021).

Rezende, Danilo Jimenez and Fabio Viola (Oct. 1, 2018). *Taming VAEs*. arXiv: 1810.00597 [cs, stat]. URL: http://arxiv.org/abs/1810.00597 (visited on 06/28/2021).

Rossi, Andrea et al. (Jan. 4, 2021). "Knowledge Graph Embedding for Link Prediction: A Comparative Analysis". In: *ACM Transactions on Knowledge Discovery from Data* 15.2, 14:1–14:49. ISSN: 1556-4681. DOI: 10.1145/3424672.

Ruffinelli, Daniel, Samuel Broscheit, and Rainer Gemulla (Sept. 25, 2019). "You CAN Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings". In: International Conference on Learning Representations. URL: https://openreview.net/forum?id=BkxSmlBFvr (visited on 05/25/2021).

Safavi, Tara and Danai Koutra (Nov. 2020). "CoDEx: A Comprehensive Knowledge Graph Completion Benchmark". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. EMNLP 2020. Online: Association for Computational Linguistics, pp. 8328–8350. DOI: 10.18653/v1/2020.emnlp-main.669. URL: https://www.aclweb.org/anthology/2020.emnlp-main.669 (visited on 03/12/2021).

Sobol', Il'ya Meerovich (1967). "On the distribution of points in a cube and the approximate evaluation of integrals". In: *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki* 7.4, pp. 784–802.

Toutanova, Kristina et al. (Sept. 2015). "Representing Text for Joint Embedding of Text and Knowledge Bases". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. EMNLP 2015. Lisbon, Portugal: Association for Computational Linguistics, pp. 1499–1509. DOI: 10.18653/v1/D15-1174. URL: https://www.aclweb.org/anthology/D15-1174 (visited on 05/27/2021).

Wang, Peng and Jing Zhou (2020). "JECI++: A Modified Joint Knowledge Graph Embedding Model for Concepts and Instances". In: *Big Data Research*, p. 100160. ISSN: 2214-5796. DOI: 10.1016/j.bdr.2020.100160. URL: https://www.sciencedirect.com/science/article/pii/S2214579620300289.

Wang, Q. et al. (2017a). "Knowledge Graph Embedding: A Survey of Approaches and Applications". In: *IEEE Transactions on Knowledge and Data Engineering* 29.12, pp. 2724–2743. DOI: 10.1109/TKDE.2017.2754499.

– (Dec. 2017b). "Knowledge Graph Embedding: A Survey of Approaches and Applications". In: *IEEE Transactions on Knowledge and Data Engineering* 29.12, pp. 2724–2743. ISSN: 1558-2191. DOI: 10.1109/TKDE.2017.2754499.

Xie, Ruobing, Zhiyuan Liu, and Maosong Sun (2016). "Representation Learning of Knowledge Graphs with Hierarchical Types." In: *IJCAI*, pp. 2965–2971.

Zhou, Jing et al. (2020). "JECI: A Joint Knowledge Graph Embedding Model for Concepts and Instances". In: *Semantic Technology*. Ed. by Xin Wang et al. Cham: Springer International Publishing, pp. 82–98. ISBN: 978-3-030-41407-8.

Appendices

A Details hyper-parameter searches

Parameters - Training

Like Ruffinelli et al. we also use a learning rate schedule based on PyTorch's ReduceLROnPlateau class, with a reduction factor of 0.95, and a lower threshold of 0.0001. Table A.1 list the training hyper-parameters that were tuned, and the values that were considered.

parameter	*values*
batch size	128, 256, 512, 1024
optimizer	Adam, Adagrad
learning rate (lr)	[0.0001, 1.]
lr schedule patience	[0, 10]

Table A.1: The hyper-parameters relating to the training process.

Parameters - Embeddings

If a run uses 'xavier_normal' or 'xavier_uniform' the gain is always 1.0, if 'normal' is used the mean is always 0.0 . Table A.2 list the embedding hyper-parameters that were tuned, and the values that were considered.

parameter	*values*
dimensions	128, 256, 512
initialization	xavier_normal, xavier_uniform, normal, uniform
initialization.normal.std	[0.00001, 1.0]
initialization.uniform.a	[-1.0, -0.00001]
regularization	none, l1, l2, l3
regularization.weighted	True, False
regularization.weight †	[1.0e-20, 1.0e-01]
dropout †	True, False
dropout.prob †	[0.0, 0.5]

Table A.2: The hyper-parameters relating to the embeddings. Parameters marked with a † are parameters that are set separately for each kind of embedding (entity, relation, type).

Parameters - TransE and negative-sampling

We use a 'shared' implementation of negative-sampling, meaning that the same negative samples are used for each triple in a batch. Note that it is ensured that a triple does not get itself as a negative. Table A.3 list the TransE and negative-sampling hyper-parameters that were tuned, and the values that were considered.

Parameters - Type-linkprior

Table A.4 list the Type-linkprior hyper-parameters that were tuned, and the values that were considered.

parameter	*values*
l-norm	1.0, 2.0
normalize.p †	None, 2.0
num-negative-samples.s	[1, 1000]
num-negative-samples.o	[1, 1000]

Table A.3: The hyper-parameters relating to TransE and negative sampling. Parameters marked with a † are parameters that are set separately for each kind of embedding (entity, relation, type).

parameter	*values*
λ_{head}	[0.0, 1.0]
$\lambda_{relation}$	[0.0, 1.0]
λ_{tail}	[0.0, 1.0]
ρ	[0.0, 1.0]
learn-λ	True, False

Table A.4: The hyper-parameters relating to the Type-linkprior method.

Parameters - Type-mean

The only hyper-parameter specific to the type-mean methods is the 'use_entity_embedder' that determines if we are using the entity-specific offsets as described in 3.3.

Parameters - Type-attentive

Table A.5 list the Type-attentive hyper-parameters that were tuned, and the values that were considered.

parameter	*values*
type_attn_nr_heads	1, 2, 4, 8
type_attn_lr	[0.0001, 1.0]
mutual_information_min	[-20.0, 20.0][4]
mutual_information_min_scale	[0.01, 100.0]
mutual_information_min_damping	[0.01, 100.0]

Table A.5: The hyper-parameters relating to the Type-attentive method.

[4] Negative values disable the mutual information constraint.

Parameters - Type-embedprior

Table A.6 list the Type-attentive hyper-parameters that were tuned, and the values that were considered.

parameter	*values*
type_prior_init_std	[0.0, 1.0]
type_nll_min	[-20.0, 0.0]

Table A.6: The hyper-parameters relating to the Type-embedprior method.

B Additional performance metrics of hyper-parameter search

name	WN18RR	FB15K-237	CoDEx-S	CoDEx-M
non-semantic	5.24	21.37	21.13	22.14
type-linkprior-only	00.00	9.58	3.09	3.29
type-linkprior	1.17	23.06	23.81	19.72
type-mean	72.33	23.31	35.74	30.11
type-mean+	5.19	23.31	23.56	19.68
type-attentive	-	**61.44**	**45.87**	**44.79**
type-attentive-alt	**81.05**	21.13	22.03	41.35
type-embedprior	6.38	22.57	25.42	21.68

Table B.1: Hits@1 (%) obtained by each method for best set of hyperparameters.

name	WN18RR	FB15K-237	CoDEx-S	CoDEx-M
non-semantic	36.01	34.23	42.88	34.27
type-linkprior-only	00.00	14.98	11.77	4.78
type-linkprior	35.78	35.73	40.07	32.37
type-mean	72.33	29.29	47.13	37.21
type-mean+	23.88	34.90	39.60	30.67
type-attentive	-	**61.44**	**75.18**	46.86
type-attentive-alt	**82.22**	32.79	38.81	**46.99**
type-embedprior	40.05	34.92	43.43	33.43

Table B.2: Hits@3 (%) obtained by each method for best set of hyperparameters.