

Deep learning for noise-tolerant RDFS reasoning¹

Bassem Makni^a, James Hendler^b

^a *AI Reasoning, IBM Research, 1101 Kitchawan Rd, Yorktown Heights, NY 10598, USA*

E-mail: bassem.makni@ibm.com

^b *TWC, Rensselaer Polytechnic Institute, 110 8th St, Troy, NY 12180, USA*

E-mail: hendler@cs.rpi.edu

Editors: First Editor, University or Company name, Country; Second Editor, First University or Company name, Country and Second University or Company name, Country

Solicited reviews: First Solicited reviewer, University or Company name, Country; anonymous reviewer

Open review: First Open Reviewer, University or Company name, Country

Abstract. Since the 2001 envisioning of the Semantic Web (SW) [1], the main research focus in SW reasoning has been on the soundness and completeness of reasoners. While these reasoners assume the veracity of input data, the reality is that the Web of data is inherently noisy. Although there has been recent work on noise-tolerant reasoning, it has focused on type inference rather than full RDFS reasoning. Even though RDFS closure generation can be seen as a Knowledge Graph (KG) completion problem, the problem setting is different—making KG embedding techniques that were designed for link prediction not suitable for RDFS reasoning. This paper documents a novel approach that extends noise-tolerance in the SW to full RDFS reasoning. Our embedding technique—that is tailored for RDFS reasoning—consists of layering RDF graphs and encoding them in the form of 3D adjacency matrices where each layer layout forms a *graph word*. Each input graph and its entailments are then represented as sequences of graph words, and RDFS inference can be formulated as translation of these graph words sequences, achieved through neural machine translation. Our evaluation on LUBM1 synthetic dataset shows 97% validation accuracy and 87.76% on a subset of DBpedia while demonstrating a noise-tolerance unavailable with rule-based reasoners.

Keywords: Deep learning, Semantic Web, RDFS reasoning, Noise-tolerance, Neural machine translation, Graph words

1. Introduction

The Web is inherently noisy and as such its extension is noisy as well. This noise is as a result of inevitable human error when creating the content, designing the tools that facilitate the data exchange, conceptualizing the ontologies that allow machines to understand the data content, mapping concepts from different ontologies, etc. For instance, noise can be a consequence of building Linked Open Data (LOD) from semi-structured or non-structured data. When LOD is built from non-structured data such as text using

Named Entity Linking (NEL) or relation extraction tools—whose accuracy is not perfect—they generate erroneous triples. Thus, the integrity of the inference becomes questionable.

It is foolish to expect that the Web or the SW will ever be free of noise. Many research efforts concentrate on noise detection and data cleansing in the Web of data. Knowing that there will always be other instances or types of noise that will be overlooked, other research efforts focus on noise-tolerance instead. Most of the current work in the latter category targets adding some noise-tolerant reasoning capabilities without aiming for full semantic reasoning.

Humans are able to learn from very few examples while providing explanations for their decision making

¹The code, models and datasets for this paper are available at: <https://github.com/Bassem-Makni/NMT4RDFS>

process. In contrast, deep learning techniques- even though robust to noise and very effective in generalizing across a number of fields including machine vision, natural language understanding, speech recognition etc. - require large amounts of data and are unable to provide explanations for their decisions. Attaining human-level robust reasoning requires combining sound symbolic reasoning with robust connectionist learning as outlined in [2]. “We argue that to face this challenge one first needs a framework in which inductive learning and logical reasoning can be both expressed and their different natures reconciled.” [2, 1] However, connectionist learning uses low-level representations- such as embeddings- rather than “symbolic representations used in knowledge representation” [3, 18]. This challenge constitutes what is referred to as the *Neural-Symbolic gap*. The aim of this research is to provide a stepping stone towards bridging the *Neural-Symbolic gap* specifically in the SW field and RDF Schema (RDFS) reasoning in particular.

This paper documents a novel approach that takes previous research efforts on noise-tolerance to the next level of full RDFS reasoning. The proposed approach utilizes the recent advances in deep learning- that showed robustness to noise in other machine learning applications such as computer vision and natural language understanding- for semantic reasoning. The first step towards bridging the Neural-Symbolic gap for RDFS reasoning is to represent Resource Description Framework (RDF) graphs in a format that can be fed to neural networks. The most intuitive representation to use is graph representation. However, RDF graphs differ from simple graphs as defined in the graph theory in a number of ways. We examine in the literature different graph models for RDF from which we conclude that the proposed models were neither designed for RDFS reasoning requirements nor are they suitable for neural network input. The proposed graph model for RDF consists of layering RDF graphs and encoding them in the form of 3D adjacency matrices. Each layer layout in the 3D adjacency matrices forms what we termed as a *graph word*. Every input graph and its corresponding inference are then represented as sequences of graph words. The RDFS inference becomes equivalent to the translation of graph words that is achieved through neural network translation.

The evaluation confirms that deep learning can in fact be used to learn RDFS rules from both synthetic as well as real-world SW data while showing noise-tolerance capabilities as opposed to rule-based reasoners.

1.1. Contributions and outline

The main contributions in this paper are:

- **Noise Intolerance Conditions.** In order to illustrate the intolerance of rule-based reasoners to noise in SW data, a taxonomy for noise types in SW data according to the impact of the noise on the inference is drawn. Additionally, the necessary conditions for a noise type to be propagable (i.e affect the inference) by any RDFS rule is discerned.
- **Layered Graph Model for RDF.** Even though the literature encompasses quite a few propositions for graph models for RDF, none of them were designed for RDFS reasoning specifically. We propose a layered graph model for RDF that fulfills this requirement.
- **Graph Words.** Using the layered graph model, we propose a novel way of representing RDF graphs as a sequence of graph words. The main observation that led to this design is that layers of RDF graphs in a restricted domain are slightly variable.
- **Graph-to-Graph Learning.** By representing RDF graphs as a sequence of graph words, we were able to use neural network translation techniques for translation of graph words. This constitutes a novel approach for graph-to-graph learning.
- **Full RDFS reasoning with noise tolerance.** Our evaluation shows not only comparable results with rule-based reasoners on intact data but also exceptional noise-tolerance compared to them: 99% for the deep reasoner vs 0% (by design) for Jena in the *UGS*₁₀₀ dataset.

In Section 2, we use three aspects to position our research with respect to the related work. Section 3 draws a taxonomy for noise types in SW data and illustrates the process of ground truthing and noise induction for LUBM and a subset of DBpedia— that are used as examples to describe the design of the overall approach. We examine different graph models for RDF and motivate the design of the layered graph model for RDF in Section 4. Then the creation of the RDF tensors and the RDF graph words as well as the description of the graph words translation are presented respectively in Section 5 and Section 6. The results of the experiments are described in the Section 7. In Section 8, we review the related literature in terms of noise-tolerance in the SW, deep learning and the SW and graph embedding techniques— specifically KG

embedding. Finally the learned lessons, main contributions and future work are illustrated in Section 9.

2. Background and Problem Statement

In this section we use three aspects to position our research with respect to related work:

- Noise handling strategies: active vs adaptive
- Knowledge graph completion categories: schema-guided vs data-driven
- Graph embedding output: Node/Edge embedding vs whole-graph embedding.

2.1. Noise handling strategies

We classify the strategies of handling noise in SW data into two categories:

- **Active noise handling** consists of detecting noise and cleansing the data before performing any tasks that might be affected by the presence of noise
- **Adaptive noise handling** the previous category provides solutions that are tailored to certain types of noise as described in Section 8.1.1. Given the unrealistic expectation of cleansing every type of noise in SW data, adaptive noise handling approaches focus rather on building techniques that are noise-tolerant. The research described in this paper falls into this category as we are building a noise-tolerant RDFS reasoner.

2.2. Knowledge graph completion categories

RDFS closure can be seen as a Knowledge Graph Completion (KGC) problem— multi-relational link prediction problem in particular— where each RDFS rule (see Appendix A) generates different types of links:

- (a) links between TBox concepts (*RDFS10*, *RDFS11*)
- (b) links between TBox properties (*RDFS5*, *RDFS6*)
- (c) links between ABox entities and TBox concepts (*RDFS2*, *RDFS3*, *RDFS9*)
- (d) links between ABox entities (*RDFS7*)

We refer to the RDFS closure computation as *schema-guided KGC* because the links are generated according to the ontology (TBox), unlike *data-driven KGC* where the links are predicted based on the analysis of the existing links in the KG. Data-driven KGC

models “heavily rely on the connectivity of the existing KG and are best able to predict relationships between existing, well-connected entities” [4, 1]. The predicted links from data-driven KGC might be seen as links between ABox entities and thus similar to the (d) case in the schema-guided KGC. However, there is a crucial difference: the generated relation (or link label) by the *RDFS7* rule is a super-property that might not be seen in the initial KG as it is defined only in the TBox as a node not even as a link type, whereas all the relations generated by data-driven KGC are necessarily seen before in the initial KG.

2.3. Graph embedding output

Graph embedding approaches can be classified using several criteria. One particular criterion of interest in our survey of the state of the art (Section 8) is the “problem setting” [5]. The problem setting uses the type of graph input as well as the embedding output to classify the embedding approach. For the input part of the problem setting, the graph can be either:

- **Homogeneous.** Where all the nodes are of the same type and all the edges are of the same type as well.
- **Heterogeneous** where there are multiple types of nodes and/or multiple types of edges. This is the case for RDF graphs.

The majority of graph embedding approaches (detailed in Section 8) yield node representation in a low dimensional space. This is why graph embedding and node embedding are often used interchangeably. However, there are other types of graph embedding outputs such as:

- **Edge embedding.** The output in this case is a low dimensional representation of the edges. This is particularly useful in the case of knowledge graphs [6] where the type of edges between nodes is crucial to determine their similarities.
- **Whole graph embedding** the output is a vector representation of the whole graph— not only node or edge vectors. The embedding vectors of similar graphs should be neighbors in the embedding space. The embedding of RDF graphs— in order to learn their inference— falls under this category.

2.4. Problem statement

For learning RDFS reasoning, the whole-graph embedding is required because the input of the learning algorithm is the input graph and the output is the inference graph. However, existing embedding techniques for KGs were not designed for RDFS reasoning and they raise two main challenges if they were to be used for this task.

1. The first challenge is the need to check the validity of every possible triple using the scoring function $f_r(h, t)$ (described in Section 8) in order to generate the full materialization.
2. The second challenge is the embedding of the relations that are seen only in the inference as the embeddings should be learned only from the input graph and be used to generate the inference graphs. For instance, when the property *masters-DegreeFrom* in LUBM appears in the input graph, its super-property *degreeFrom* appears in the inference graph by applying the *RDFS7* rule [7]. If *degreeFrom* was not seen in the input graph then its embedding was not learned.

The baseline experiments detailed in Section 7.3 illustrate these challenges empirically.

3. Ground Truthing and Noise Induction

For this research, the input is from one of two types of datasets: a synthetic dataset from LUBM and a real-world dataset from DBpedia [8]. The inferential target for these datasets is set using a rule-based SW reasoner (Jena [9]). Essentially, the goal for the deep reasoner is to learn the mapping between input RDF graphs and their entailed graphs in the presence of noise. Thus, noise was induced in the synthetic dataset to test the noise-tolerance of the deep reasoner.

3.1. Taxonomy of Semantic Web noise types

The literature contains a few taxonomies [10, 11] for the types of noise that can impact RDF graphs; however they are not drawn with respect to the impact of the noise on the inference. The taxonomy illustrated in Fig. 1 serves this purpose. It should be noted that the propagation of noise is dependent on the inference rule.

TBox Noise is the type of noise that resides within the ontology, such as in the class hierarchy or domain

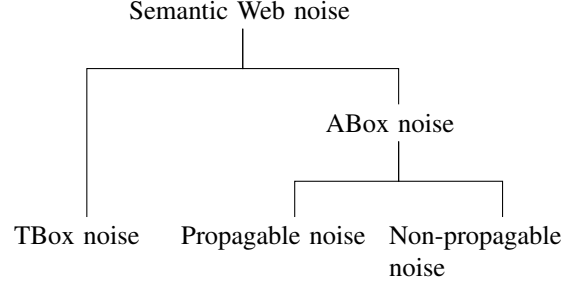


Fig. 1. Semantic Web noise taxonomy

and range properties. This type of noise impacts inference over the whole dataset. For example, in the DBpedia ontology, the property *dbo:field* has domain *dbo:Artist* which implies that every scientist in the DBpedia dataset who has a *dbo:field* property (such as *dbr:Artificial_intelligence* or *dbr:Semantic_Web*) will be labelled a *dbo:Artist* after inference. Reasoning with tolerance to TBox noise is outside the scope of this research for the following reason: the use of rule-based reasoners for ground truthing with noise in the TBox biases the whole ground truth, which makes noisy inferences omnipresent and not just anomalies that can be detected and fixed.

The following assumption is made in order to scope this research within a manageable framework:

Assumption 1 (Noise locality). *The noise is latent only in the ABox, but the TBox is devoid of noise.*

Definition 1 (Triple corruption). The process of morphing an existing triple in an RDF graph by changing one of the triples' resources. This can result in either propagable or non-propagable noise.

Definition 2 (Non-propagable noise). Any corrupted triple in the input graph that does not have any impact on the inference.

This can occur at least in these cases:

1. The original triple does not generate any inference nor does the corrupted triple.
2. The original triple does not generate any triple but the corrupted triple generates an inference that is generated also by another triple in the input graph. (For example if the corrupted triple is equal to another triple in the graph)
3. The original triple and the corrupted triple generate the exact same inference.

Table 1
RDFS9 rule from [7]

RDFS rule	If	Then
RDFS9	$x \text{ rdfs:subClassOf } y .$ $z \text{ rdf:type } x .$	$z \text{ rdf:type } y .$

4. The corrupted triple generates a set of triples that is a proper subset of the set of triples generated by the original triple. However the difference between the two sets is also generated by other input triples.

Definition 3 (Propagable noise). Any corrupted triple in the input graph that changes the inference.

In order to discern the necessary conditions for RDFS rules to propagate noise, first the input patterns of the premises of the RDFS rules ([7]) are classified as TBox pattern or ABox patterns (Appendix B). The rules that have only TBox type patterns, such as *RDFS5* (which defines the properties hierarchy) and *RDFS11* (which defines the class hierarchy), are excluded because any corruption of triples matching these patterns will induce TBox noise. For the remaining rules that have both TBox and ABox patterns (i.e. *RDFS2*, *RDFS3*, *RDFS7* and *RDFS9*), only the ABox triple can be corrupted. In Table 2, the necessary conditions for RDFS9 rule (Table 1) to generate a noisy inference from a corrupted triple are identified. In plain English, the *RDFS9* rule will generate a noisy inference if and only if the corrupted type x' has a super-class y' defined in the ontology and the original type x either does not have a super-class or y' is not a super-class of x . The necessary conditions for the remaining rules are listed in Appendix C.

Table 2
Propagable noise by rule-based RDFS reasoners for RDFS9 rule

RDFS rule	Triple corruption	Conditions	Noisy inference
RDFS9	$z \text{ rdf:type } x .$ \rightarrow $z \text{ rdf:type } x' .$	$(x' \text{ rdfs:subClassOf } y' .)$ $\wedge ((\neg \exists y, x \text{ rdfs:subClassOf } y .) \vee$ $(\forall y, x \text{ rdfs:subClassOf } y .$ $\implies \neg(y = y'))))$	$z \text{ rdf:type } y' .$
Example	$\text{stu1 rdf:type ub:Student} .$ \rightarrow $\text{stu1 rdf:type ub:University} .$	$\text{ub:University rdfs:subClassOf}$ $\text{ub:Organization} .$	stu1 rdf:type $\text{ub:Organization} .$

3.1.1. Mapping to the DBpedia noise taxonomy drawn in "User-driven Quality Evaluation of DBpedia" [11]

In [11], the authors examine different types of noise in the DBpedia instances. Thus all the categories of noise presented are of type ABox noise. Every category of noise can either be propagable or non-propagable depending on the property in the noisy triple. For example, in *Object value is incorrectly extracted* [11], the noise can be propagable if the corresponding property has any super-properties defined in the ontology and non-propagable otherwise—such as in the following example provided in the paper:

```
DBpedia:Oregon_Route_238 dbpprop:map
"238.0"^^http://DBpedia.org/datatype/second.
```

where the property *dbpprop:map* does not have any super-properties in the DBpedia ontology.

3.2. Ground-Truthing in LUBM1

Lehigh University Benchmark (LUBM) [12] is a benchmark for SW repositories. The LUBM ontology conceptualizes 42 classes from the academic domain and 28 properties describing these classes' relationships. LUBM1, an RDF graph of one hundred thousand triples, was generated according to this ontology, and contains 17,189 subject-resources within 15 classes (for instance 5,999 resources of type *ub:Publication* and 15 resources of type *ub:Department*). Let R be the set of these subject-resources. For each resource r in R , a graph g is built by running the SPARQL DESCRIBE query. Appendix D contains the graph description of the resource *GraduateStudent9*. Let G be the set of graphs g obtained after this step. For each graph g in G , the RDFS inferential closure is generated according to the LUBM ontology using Jena. Let I be the set of inference

graphs. Appendix E contains the inference graph of the input graph in Listing 1. Finally, G and I are split into 60% training (G_{train}, I_{train}), 20% validation (G_{val}, I_{val}) and 20% (G_{test}, I_{test}) testing sets using a stratified splitting technique where the resource class is used as the label for the stratification. The goal of the stratification is to have the required percentage of each resource type in the training and test sets. Otherwise there is a risk of having all the small classes in the training set, which will mistakenly inflate the accuracy. For instance, the stratified split leads to 9 graphs describing resources of type *ub:Department* in the training set and 3 graphs describing resources of the same type in validation and test sets respectively. The input of the supervised learning algorithm is the set of graphs G_{train} , the target is their corresponding inference graphs I_{train} and the goal is to learn the inference generation.

3.2.1. Noise Induction in LUBM1

In [13], a methodology for noise induction in LUBM was proposed in which three datasets were constructed by corrupting type assertions according to a given noise level.

RATA Instances of type *TeachingAssistant* were corrupted to be of type *ResearchAssistant*. This type of noise is non-propagable because both concepts, *TeachingAssistant* and *ResearchAssistant*, are sub-classes of the concept *Person*.

UGS Instances of type *GraduateStudent* were corrupted to be of type *University*. This type of noise is propagable by the RDFS rule *RDFS9* because these concepts are not siblings. A rule reasoner will generate a noisy inference by deducing that the student instance is of type *Organization* which is the super-class of *University*.

GCC Instances of type *Course* were corrupted to be of type *GraduateCourse*. This type of noise is also non-propagable.

As [13] focus only on noisy type assertions, two additional datasets were created with noisy property assertions for the purpose of this research.

TEPA The property *publicationAuthor* is corrupted to be *teachingAssistantOf*. This noise is propagable by the RDFS rules *RDFS2* and *RDFS3* as the two properties have different domains and ranges.

WOAD The property *advisor* is corrupted to be *worksFor*. This noise is non-propagable as the property *worksFor* does not have any domain or range specification in the LUBM ontology, but by re-

Table 3

Number of resources per class in the scientists dataset

Class	Number of resources
dbo:Scientist	25,760
dbo:Place	22,035
dbo:EducationalInstitution	6,048
dbo:Award	1,166

moving the property *advisor* the type inference that was made about the student and advisor is lost.

3.3. Ground Truthing the Scientist Dataset from DBpedia

From DBpedia [14], a dataset of scientists' descriptions was built; 25,760 URIs for scientists' descriptions were retrieved. In order to diversify the types of classes in the scientists dataset, a few other classes that are related to the Scientist concept in DBpedia were also collected, namely: *EducationalInstitution*, *Place* and *Award*. Table 3 lists the number of resources per class in the scientist dataset. The total number of triples obtained in the scientists dataset is $\simeq 5.5$ million. No artificial noise was induced in this dataset as it already has pre-existing noise. An example of noisy type assertion is the resource *dbr:United_States* being of type *dbo:Person*. There are 1,761 resources in DBpedia that are of types *dbo:Person* and *dbo:Place* simultaneously, which obviously indicates that one of them is a noisy triple.

4. Layered Graph Model for RDF

Despite its effectiveness as a standardized “framework for representing information in the Web”[15] and as an essential building block for the SW, the graph representation for the RDF model remains an open question in the SW research community. Even though the RDF conceptual model is designed as a graph, it differs from the graph theory definition of graphs in a number of ways. RDF graphs are heterogeneous multigraphs. Moreover, an edge in the ABox can be a node in the TBox (describing the properties hierarchy for example). Current research efforts to represent the RDF model as graphs—based on a: bipartite graph model [16], hypergraph model [17–19] or metagraph models [20]—target different goals ranging from storing and querying RDF graphs to reducing space and

time complexity to solving the reification and provenance problem. Unfortunately, these goals do not coincide with RDFS reasoning. Moreover they use complex graph models which are not suitable for neural network input.

This paper describes a layered graph model that uses simple directed graphs to achieve the goal of representing RDF graphs and their inference graphs according to the RDFS rules. It is important to note that the mapping between RDF to the proposed layered model is irreversible—meaning that the reconstruction of the original RDF graph is not guaranteed. Thus, the layered graph model is not suitable for storing and querying RDF data.

4.1. Notations and Definitions

In Appendix B, the premises of RDFS rules were classified into ABox patterns and TBox patterns.

Definition 4. TBox rule is a rule where its premises are all of type TBox pattern.

The Tbox rules in RDFS are:

1. *RDFS5*: the *subPropertyOf* transitivity rule
2. *RDFS6*: the *subPropertyOf* reflexivity rule
3. *RDFS11*: the *subClassOf* transitivity rule
4. *RDFS10*: the *subClassOf* reflexivity rule

As these rules' patterns are present in the ontological level and there is only one ontology per training set, there are not enough samples to learn these rules. Thus, it is assumed that there is a materialized version of the ontology where the TBox rules are already applied. This materialized version is inferred only once and is part of the training input. Let:

O : be the materialized ontology.

P : be the set of properties in O .

$P^+ = P \cup \{rdf:type\}$

np : be the size of the set P^+

$(p_1, p_2, \dots, p_{np})$: be a tuple of the elements of P^+ (It is crucial to maintain the same order of elements in this tuple throughout the training process)

$SubjObj(T)$: be the set of subject and object resources of the RDF graph T (formally defined in Appendix F).

Definition 5. A Layered directed graph is a graph that has multiple sets of directed edges where each layer has its own set of edges.

An n -layered directed graph is a layered directed graph of n layers. More formally, an n -layered directed graph

is defined as:

$G(V, (E_1, E_2, \dots, E_n))$ where the edges part is a tuple containing n sets of directed edges.

Definition 6. Layered directed graph for RDF :

An RDF graph T is represented by a layered directed graph:

$G(SubjObj(T), (E_1, E_2, \dots, E_{np}))$ where:

$$(e_i, e_j) \in E_l \iff \begin{cases} (e_i, p_l, e_j) \in T \\ e_i \in SubjObj(T) \\ e_j \in SubjObj(T) \end{cases}$$

It is important to note that the transformation of an RDF graph into its layered directed graph representation is not bijective as two non-isomorphic RDF graphs can have the same layered directed graph representation.

Proof by construction. Let T be an RDF graph and L_T be its layered directed graph representation according to the ontology O and its tuple of properties $(p_1, p_2, \dots, p_{np})$. If $(s, p, o) \notin T$ and $p \notin (p_1, p_2, \dots, p_{np})$ then the RDF graph $T' = T \cup (s, p, o)$ is not isomorphic to T but has the same representation L_T . \square

However this transformation guarantees that if two RDF graphs have the same layered directed graph representation then their RDFS inference graphs according to the ontology O are isomorphic.

Appendix G lists two examples of layered RDF graphs: one for the graph description of the resource *GraduateStudent9* and one for its corresponding inference graph.

5. RDF Tensors, and the Graph Words Embedding

The way the generic methodology of supervised machine learning is applied in this work is depicted in Figure 2, where the pair (input, target) is the input graph and its corresponding inference. In a nutshell, tensors representing the input graph g and its corresponding inference i are created. The tensors of these graphs are then used in the training phase. The algorithm outputs the tensor of the graph and its encoding dictionary that will be used in the decoding phase to regenerate the original graph. In addition to preparing the RDF graph for input into a neural network, the main goal of this phase is to capture the pattern sim-

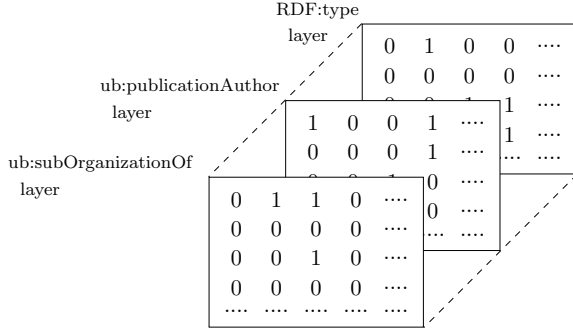


Fig. 3. 3D Adjacency matrix

ilarities between graphs in such a way that “similar” graphs will have similar tensors. An example of “similar” graphs is: two graphs containing RDF descriptions of two resources of the same type (such as two *Publications*’ descriptions in the LUBM1 dataset).

5.1. Tensor Creation

The goal of this phase is to use the layered graph model for RDF in creating RDF tensors. Each RDF graph will be represented as a 3D adjacency matrix, where each layer is the adjacency matrix relative to one property (Fig. 3). An ID must be assigned to each resource in the RDF graph to allow it to be represented as a 3D adjacency matrix. The process of assigning these

IDs for the input graphs and their corresponding inference graphs must satisfy the following requirement:

It is mandatory that the encoding dictionary for a given graph g contains all the possible resources that might be used in its corresponding inference graph i .

The proof for this requirement is detailed in Appendix H.

5.1.1. Simplified version

In the simplified version of the algorithm, two dictionaries were created: one for the subject and object IDs— which is split into a global and a local dictionary— and one for the property IDs. The global resources dictionary contains the subject and object resources that are used throughout the G set (which are basically the RDFS classes in the ontology). The *local_resources_dictionary* is created incrementally during the encoding routine for each graph g in G . It holds the IDs of the resources that are not present in the *global_resources_dictionary*. The *local_resources_dictionary* is populated with an offset equal to the length of *global_resources_dictionary*— that is, 57 in the case of LUBM1. The largest ID in the *local_resources_dictionary* for every graph in G is less than 80. This value is used to initialize the size of the 3D adjacency matrix. These dictionaries are then used

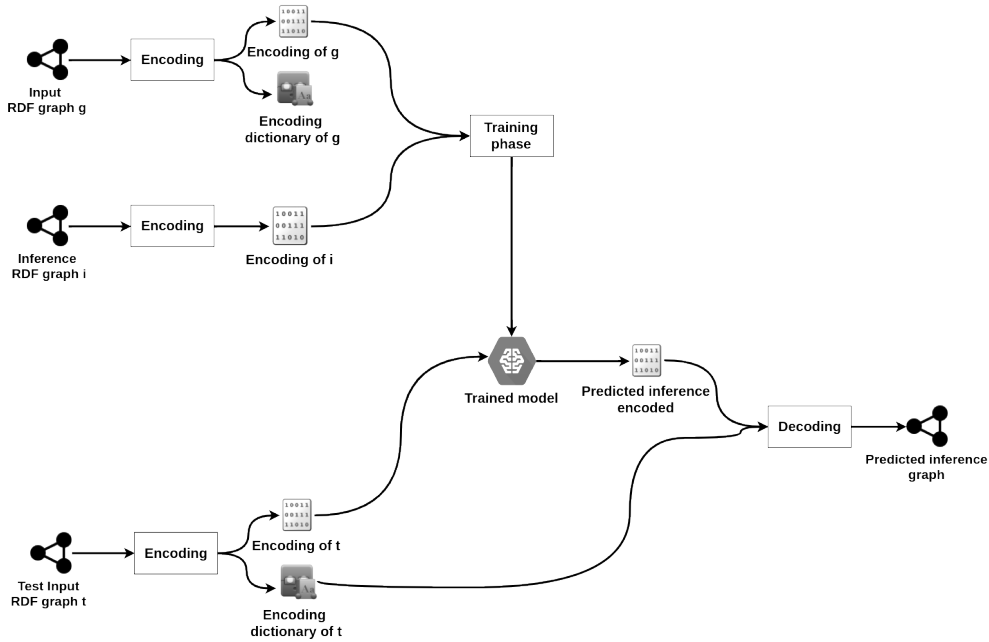


Fig. 2. Encoding/decoding in training and inference phases

in the encoding routine to transform a layered graph representation into an RDF tensor and vice-versa in the decoding routine. The details of the encoding/decoding algorithms are in Appendix I.

The previously stated goal— capturing the pattern similarities between graphs describing resources of the same type— can be achieved by this simplified encoding technique when the cardinality of each property is variable within a small range. For instance, in LUBM1, students take more or less the same number of courses, and a publication has between one to seven authors. To get the full list of these statistics, the following SPARQL query is run:

```
select ?type ?property
  (group_concat (?count) as ?
   possible_values)
where {
  select distinct ?type ?property
    (count(?object) as ?count)
  where {
    ?subject ?property ?object .
    ?subject a ?type .
  }
  group by ?type ?subject ?property
}
group by ?type ?subject ?property
order by ?type ?property
```

The inner query counts the number of objects per property per class and the outer query concatenates the possible values. Appendix M contains a sample of these statistics in LUBM1.

Alas, this is not the case in real-world knowledge graphs such as DBpedia, where even graphs describing resources of the same type differ widely. For example the DBpedia graph describing Professor James Hendler [21] has 40 objects for the property RDFt including *owl:Thing*, *foaf:Person*, *dbo:Person*, *dul:Agent*, *dbo:Agent*, *dbo:Scientist*, *schema:Person*, *yago:Scholar110557854*, etc. Out of these 40 objects, 12 are in the *global_resources_dictionary* because they are concepts in the DBpedia ontology and the other 28 objects will populate the *local_resources_dictionary*. In contrast, the DBpedia graph describing Professor Yoshua Bengio [22] has only 12 links for the property RDFt and all of the objects are in *global_resources_dictionary*. This implies that the RDFt layers in the 3D adjacency matrices for Professor Hendler and Yoshua Bengio graphs will be very different. In fact all the subsequent layers will be very different. For instance, when encoding the layer of the property *dbo:almaMater* for Professor

Hendler’s graph, the resources *dbr:Brown_University*, *dbr:Southern_Methodist_University* and *dbr:Yale_University* will have IDs 29, 30 and 31 respectively as there is already 28 resources in the *local_resources_dictionary*. When encoding the same layer for Professor Bengio’s graph, the resource *dbr:McGill_University* will have ID 1 as the corresponding *local_resources_dictionary* is still empty. Consequently, this has a domino effect on the rest of the layers. To overcome this limitation, a more advanced tensor creation method was necessary to capture the patterns of real-world knowledge graphs.

5.1.2. Advanced version

The main idea of the advanced encoding/decoding technique is to create a *local_resources_dictionary* per layer instead of a *local_resources_dictionary* for the whole graph being encoded. While this may seem sufficient to overcome the limitation of the simple encoding technique, a few challenges in the encoding of the inference graphs as well as in the decoding phase for both the input and the inference graphs are encountered. The details of these challenges and the proposed solutions for the advanced tensor creation technique are detailed in Appendix J. When using the advanced encoding technique, the number of properties is actually the number of “active” properties— where active properties are the set of properties in the T-Box that are used in the A-Box. This reduces the size of the 3D adjacency matrices dramatically especially in the case of the Scientists dataset where only a small subset of the DBpedia properties are used.

5.2. Graph Words

At this stage, every RDF graph is represented as a 3D adjacency matrix of size: (*number_of_properties*, *max_number_of_resources*, *max_number_of_resources*) where each layer represents an adjacency matrix according to one property.

In theory the maximum number of possible layer layouts in a dataset of size *dataset_size* is:

$$\min(2^{\max_nb_resources^2}, dataset_size * nb_properties)$$

When encoding an RDF graph from the LUBM1 dataset— which contains 17,189 RDF graphs— a 3D adjacency matrix of size (18 × 800 × 800) is obtained. Where 18 is the size of active properties set in LUBM1 and 800 is the maximum number of resources per graph. The maximum number of layer layouts is

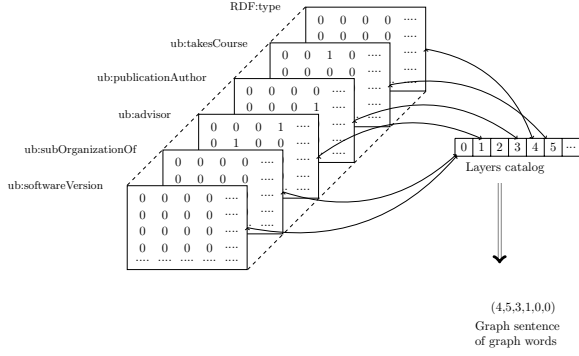


Fig. 4. From a 3D adjacency matrix to a sentence of graph words

equal to:

$\text{minimum}(2^{800^2} \simeq 10^{192,659}, 18 * 17,189) = 309,402$ possible layouts. However, the actual number of layouts when encoding LUBM1 is much smaller than this theoretical bound (131 and 490 for the sets G and I respectively). This observation is a good indication that the encoding algorithm has achieved one of its major goals of having similar encodings for “similar” graphs. Let $Catalog_G$ and $Catalog_I$ be the layers’ catalogs for the sets G and I respectively where each layout is assigned an ID. The 3D adjacency matrix can now be represented as a sequence of layouts’ IDs as shown in Fig. 4. The layouts in the catalogs are termed “graph words”, as the sequence (or phrase) of graph words represents a 3D adjacency matrix and thus an RDF graph. Representing an RDF graph as a sequence of graph words has two main advantages:

1. Reducing the size of the encoded dataset: only the ID of the layer’s layout along with a catalog of layouts is saved.
2. Exploitation of the research results in neural machine translation.

6. Graph Words Translation for RDFS Reasoning

At this stage, there is a parallel corpus of graph words for the input and inference graphs. This representation has the following drawbacks: difficulty handling “unknown” graph words and insensitivity to graph word similarities. *Unknown* graph words can be encountered when a graph word is seen only in the test set but not during the training phase; when inducing noise, most of the graph words will be unknown. A common technique in Natural Language Processing (NLP) is to assign the same ID for *unknown* words, which is not a significant deterrent to success in most

learning tasks involving natural language. However, in our case if the same ID is assigned to every unknown graph word, then the learning process will be compromised and will not generate the exact inference. (Briefly, the proof by construction—that the use of the same ID for every unknown graph word is deterrent to learning the graph words translation—consists of building two graphs having the same input representations but having different targets.)

By encoding an RDF graph as a sequence of layers where each layer contains an adjacency matrix of a directed graph according to one property, homogeneous graph embedding algorithms can be used on each layer. The High-Order Proximity preserved Embedding (HOPE) algorithm [23] was used as it had the best reconstruction accuracy when tested on the catalog of graph words. The graph words embedding also solves the problem of capturing the similarities between graph words.

By representing the RDF graph input as well as its corresponding inference graph by two sequences of graph words, the RDFS inference becomes equivalent to the translation of graph words. Thus, Neural Machine Translation (NMT) models can be applied to learn the RDFS inference generation. NMT models typical architecture consisted of [24] Recurrent Neural Network (RNN) Encoder-Decoder where the encoder RNN transforms the sequence of words from the input sentence into a fixed-length hidden representation and the decoder RNN generates the target sentence from the hidden representation. More recent architectures that used convolutional networks for NMT such as [25] outperformed RNN based architectures in terms of accuracy and training speed.

For designing the graph words translation model, we used keras [26] with TensorFlow [27] backend. It is basically a sequence-to-sequence model [28] with a Bidirectional Recurrent Neural Network (BRNN) [29] encoder. The overall architecture of the model is depicted in Fig. 5. The input layer consists of a tensor of shape (18×3200) where 18 is the size of the graph words’ sequence—which is the size of the active properties set in LUBM1. Each graph word represents a layout for an adjacency matrix of size (800×800) . When embedding the adjacency matrices using HOPE embedding technique, we chose an embedding dimension of 4. Hence the second dimension in the tensor $3,200 = 800 * 4$. The second layer *graph_input_dense* is a densely-connected layer which transforms each graph word embedding into a vector of size 256. The *gru_sequence_encoder* is a Gated Re-

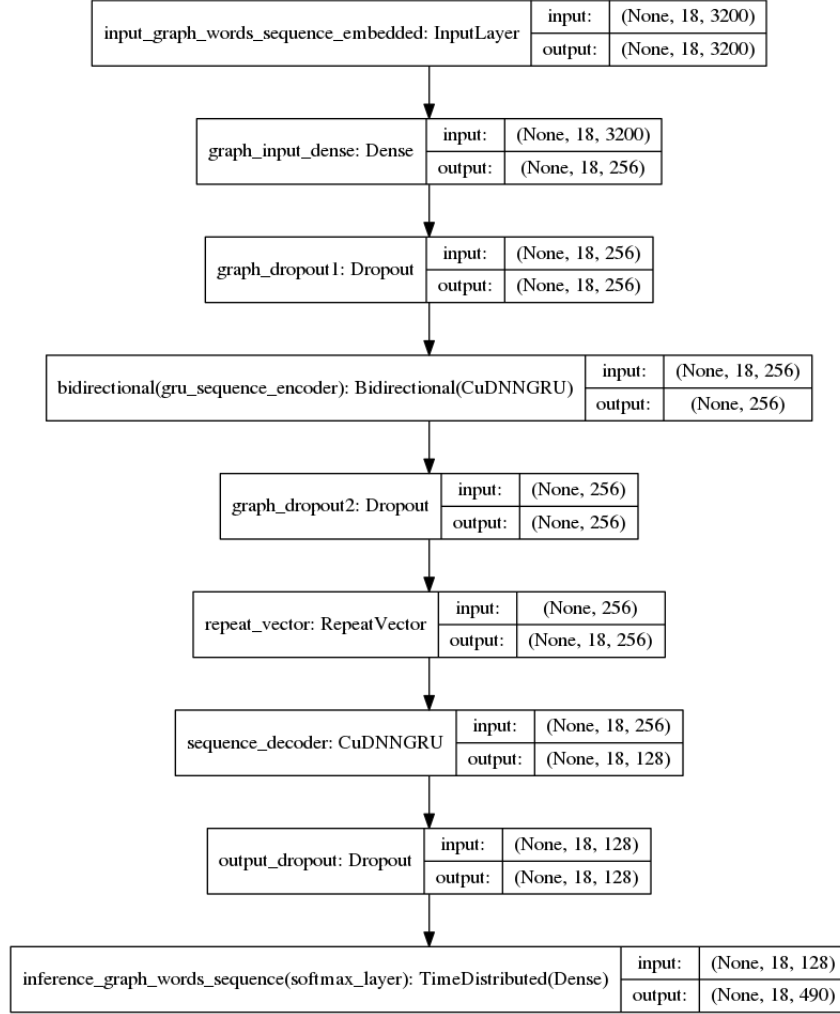


Fig. 5. Graph words translation model for LUBM1

current Unit (GRU) [30] that transforms the sequence into the hidden representation of size 128. The *bidirectional* layer feeds the sequence in positive and negative time direction to the GRU, hence the size 256. The *sequence_decoder* layer decodes the hidden representation into a sequence of size 18. The softmax layer is a densely-connected layer with softmax activation. The output of this layer is of size 490 which is the size of the inference graph words layers' catalogue. The *TimeDistributed* layer applies the *softmax_layer* on the 18 sequence elements of the previous layer to output a sequence of 18 graph words. The dropout layers have a dropout factor of 0.2 and are introduced in the model architecture to prevent overfitting and improve generalization.

For the training phase, we used Adam [31] optimizer—with a learning rate of 0.001 a first moment decay rate of 0.9 and a second moment decay rate of 0.999—and a categorical cross-entropy for the loss.

7. Evaluation

7.1. Hardware Setup

The training was done on a server, which has four Tesla K40m NVIDIA Graphics Processing Unit (GPU)s. Each GPU has 2880 Compute Unified Device Architecture (CUDA) cores and 12GB of memory. The models were trained using all the GPUs in parallel.

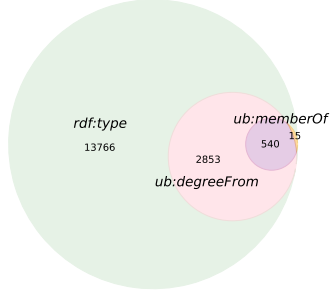


Fig. 6. The distribution of properties in LUBM1 inference

7.2. Data Analysis

In this section, we perform a statistical analysis on the training data for both LUBM1 and the scientists' dataset. This analysis is based on the relations' distribution across the training input and inference graphs. The motive behind this analysis is to get an insight to the performance of type prediction approaches on these datasets in comparison with our proposed approach.

7.2.1. LUBM1 Data Analysis

When we consider all the triples in the inference of LUBM1 across all the graphs, there are 130,377 triples with the property *rdf:type* which constitute 94.17% of the total number of generated triples. The remaining two properties in the inference *ub:degreeFrom* and *ub:memberOf* materialize in 6,988 triples (5.05%) and 1,080 triples (0.78%) respectively. However, when we consider the distribution of these two properties among the 17,189 graphs in the set G , it becomes apparent that they are spread across a larger portion of the graphs (19.84% of the inference graphs contain the property *ub:degreeFrom* and/or *ub:memberOf* while the remaining 80.1% graphs contain only triples with the property *rdf:type*). The Venn diagram in Fig. 6 illustrates the distribution of properties in LUBM1 inference.

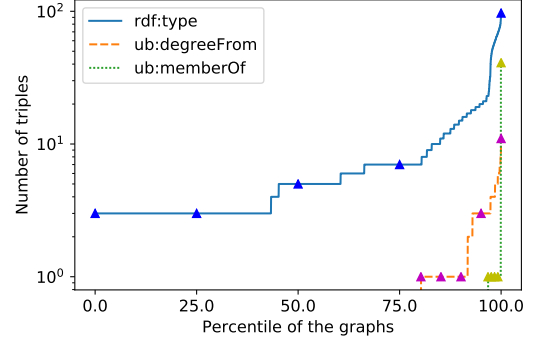


Fig. 7. Frequency of the properties in LUBM1 inference

Such distribution is due to the frequency of the properties in the inferred graphs. As shown in Fig. 7, every graph in LUBM1 inference contain at least 3 triples with the property *rdf:type* and more than 50% of the graphs contain between 5 and 97 triples with this property. On the other hand, 50% of the graphs containing the property *ub:degreeFrom* have only one triple with this property and 75% of the graphs containing the property *ub:memberOf* have only one triple with this property. Fig. 8 illustrates the frequency of each property in the input and inference graphs of LUBM1 as well as the entailment rules that generated the triples containing these properties.

Given these statistics about the frequency of properties in LUBM1 inference, it can be concluded that the upper bound of accuracy for type prediction systems is around 80%. Meaning that the perfect type predictor system can get a per-graph accuracy of 80% at most. This is due to the fact that approximately 20% of the inference graphs contain at least one property that is not *rdf:type*— which is outside the scope of type prediction systems. This percentage is even dramatically higher in the Scientists dataset as presented in the following section.

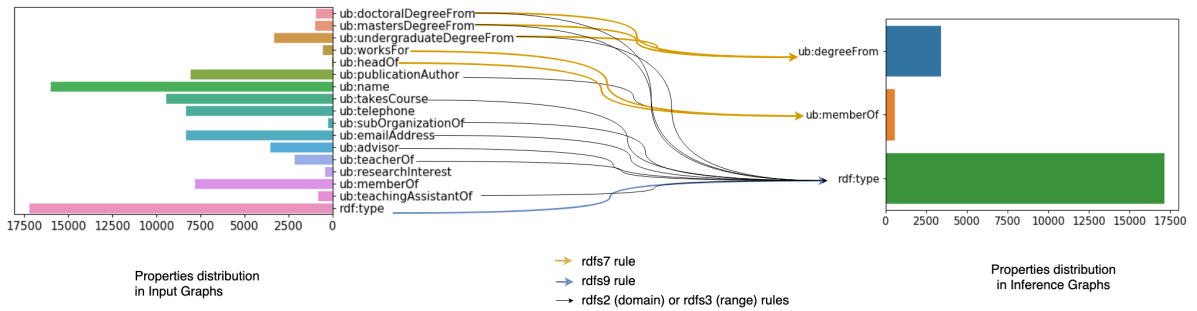


Fig. 8. Distribution of properties in input and inference graphs in LUBM1

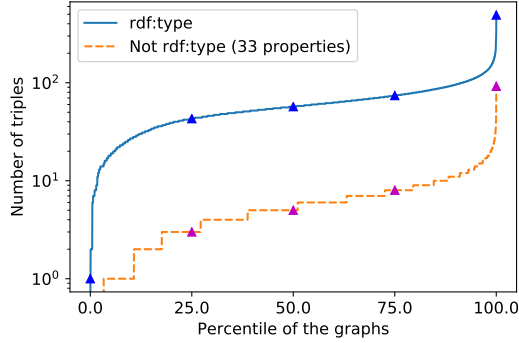


Fig. 9. Frequency of the properties in the Scientists dataset inference

7.2.2. Scientists Dataset Analysis

The Scientist's dataset contains more than fifty thousands graphs. When considering all triples in the inference of these graphs, there are 3,238,260 triples with the property *rdf:type* which constitutes 83.26%. The remaining 651,028 triples (16.74%) contain one of the 33 properties of the inference. Similarly to LUBM1, the frequency of the triples with *rdf:type* across the inference graphs is much higher than the other properties. For instance, more than 75% of the inference graphs contain between 43 and 491 triples with the property *rdf:type* while 75% of the inference graphs contain in total less than 16 triples with the remaining properties. However, unlike LUBM1, the dispersion of the remaining properties across the inference graphs is much higher—96.74% of the inference graph contain at least one triple with a property that is not *rdf:type*. This analysis sets the upper bound limit for type prediction systems at 3.26% per-graph accuracy for the Scientists dataset.

7.3. Baseline Experiments

As discussed previously in 2.4, existing KG embedding techniques that were designed for data-driven Knowledge Graph Completion (KGC) are not suitable for RDFS reasoning for the following reasons:

1. For learning RDFS reasoning, the whole-graph embedding is required rather than node/edge embeddings.
2. The closure computation requires checking the validity of every possible triple using the scoring function.
3. More importantly, any relation that is seen only in the inference (for instance generated by the *RDFS7* rule [7]) cannot be learned from the input graph.

On the other hand, these techniques are more suitable to learn from the whole KG at once and there is no need to partition the KG into subgraphs containing resource descriptions as in our approach.

In order to set the baseline and provide empirical evidence of the previous claims about using KG embedding techniques for RDFS reasoning, we run the following experiments: the embedding of LUBM1(100,867 triples containing 26,454 resources and 18 properties) is computed using 3 embedding techniques—namely TransH [32], ComplEx [33] and HolE [34] by utilizing the OpenKE toolkit [35]. OpenKE also provides a binary classifier to check the validity of triples. In order to generate the full materialization of LUBM1 using these techniques, the set of all possible triples—containing $26,454^2 * 18 \approx 12$ billion triples—need to be generated. Then the classifier can be used to check the validity of each possible triple. To make the experiment more manageable, we instead generate 100,000 random negative triples that are neither part of the input graph nor the inference

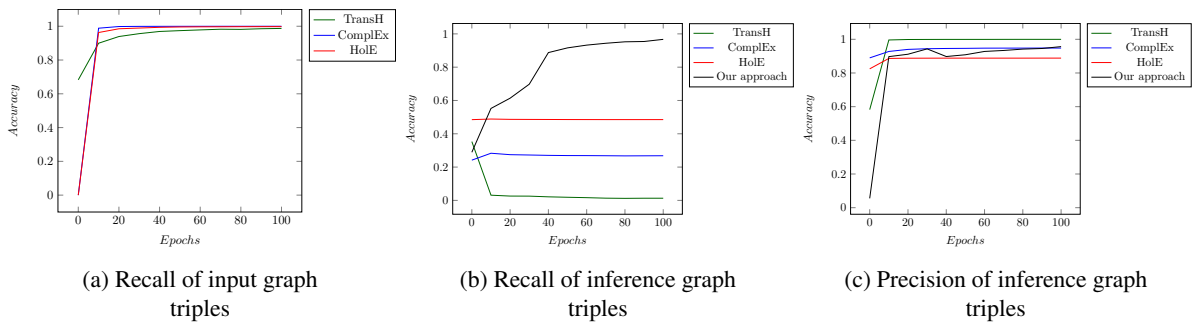


Fig. 10. RDFS inference with KG embedding techniques?

Table 4

Per-property precision and recall on LUBM1 test set

Property	True Positives	False Positives	False Negatives	Precision	Recall
rdf:type	13,642	602	725	95.77%	94.95%
ub:degreeFrom	1,392	0	3	100%	99.78%
ub:memberOf	110	32	104	77.46%	51.4%
Overall	15,144	634	832	95.98%	94.79%

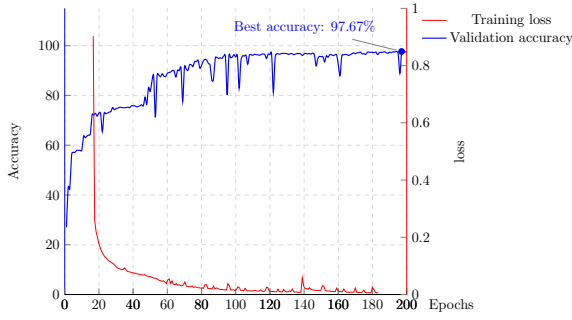


Fig. 11. Training results on intact LUBM1 data

graph and the classifier is used to check the validity of these random triples as well as the LUBM1 inference (31,612 triples). The results of these experiments are presented in Fig. 10 (the higher the better). Given that our approach is generative—that generates the inference graph from the input graph—rather than via binary classification of valid triples, it assumes validity of the input graph and thus omitted from Fig. 10a. The three embedding techniques were able to validate the triples in the input training graph (Fig. 10a) and to invalidate the negative triples (Fig. 10c), but were not able to validate most of the triples in the inference graph (Fig. 10b). HolE performs better than existing KG embedding techniques at 48.51% but did not improve with more training epochs (vs 97.7% for our approach on the test set).

7.4. Evaluation on Intact LUBM1

Fig. 11 shows the training process on the LUBM1 dataset. After approximately 12 minutes of training, 98.8% training accuracy was achieved. When testing the trained model on the intact LUBM1 test set, an overall per-graph accuracy of 97.7% was obtained.

Table 4 presents the overall per-triple precision and recall as well as the breakdown of these metrics for each property in the inference. The precision and recall are much higher for the properties *rdf:type* and *ub:degreeFrom* compared with *ub:memberOf*.

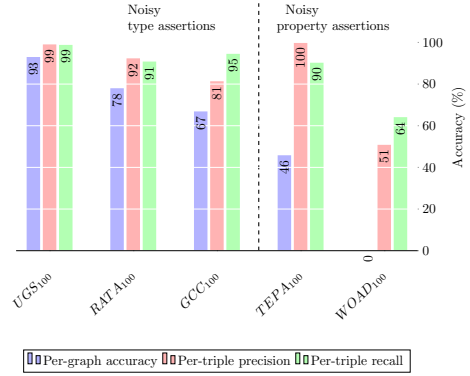


Fig. 12. Macro and micro evaluation on noisy LUBM1 datasets

This can be explained by the fact that the property *ub:memberOf* is much less frequent in the LUBM1 training set of—which contains only one university.

7.5. Evaluation on Noisy LUBM1 Data

In this experiment, the trained model was tested on the noisy datasets created as described in Section 3. Two metrics were designed:

- **Macroscopic metric: Per-graph accuracy** : Inferences in this metric (Depicted in Fig. 13) are scored correct when *dr* and *i* are isomorphic—in other words, when the deep reasoner inference from the corrupted graph is isomorphic to the Jena inference from the intact graph.
- **Microscopic metric: Per-triple precision/recall**. The previous metric overlooks the fact that some triples, generated by the deep reasoner and not by Jena, were in fact valid. In this metric, three materialization graphs are generated:

1. Jena materialization from the intact graphs (*J*)
2. The deep reasoner materialization from the corrupted graphs (*DR*)
3. An OWL-RL [36] materialization of LUBM1 to check the validity of the false positives from the deep reasoner.

Let V be the set of valid false positive triples. The quasi-confusion metric is computed as shown in Fig. 14. The macro and micro evaluation on the 5 noisy datasets (Fig. 12) shows exceptional noise-tolerance compared to rule-based reasoners: 99% for the deep reasoner vs 0% (by design) for Jena in the UGS_{100} dataset.

It is counter-intuitive that our approach performs better on propagable noise such as UGS compared with non-propagable noise such as GCC . This can be explained by the fact that the propagable noise case is very unlikely and there are no "similar" graphs in the training set. For instance, in UGS a $ub:GraduateStudent$ is corrupted to be of type $University$, which makes the university take a course in the input graph. Being more "similar" to students' graphs, it is inferred to be of type $ub:Person$ rather than $ub:Organization$. This is not the case for non-propagable noise such as GCC where a $ub:Course$ is corrupted to be of type $ub:GraduateCourse$, which is likely—i.e. "similar" graphs can exist in the training set.

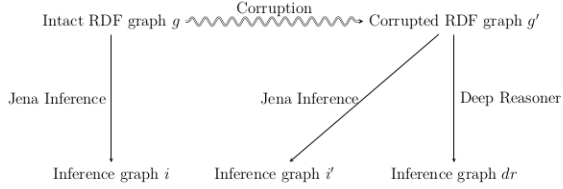


Fig. 13. Macroscopic metric: Per-graph accuracy

		Deep reasoner inference		
		p	n	total
Jena inference	p'	True positive: $(DR \cap J) \cup V$	False negative: $J \setminus DR$	P'
	n'	False positive: $(DR \setminus J) \setminus V$	True negative: -	N'
total		P	N	

Fig. 14. Refined confusion matrix

7.6. Evaluation on the Scientists Dataset

The model used for the scientists dataset is like the LUBM1 model, except for the hyper-parameters. Training to a validation accuracy of 87.76% takes over 16 hours (Fig. 15). The person-place examples were used for noise-tolerance evaluation; out of the 1,761 noisy examples of person-place in DBpedia, the 'scientists' dataset contains 94. Unlike the LUBM1 case—where training was done on intact data and testing on controlled noisy data—'scientists' training data was noisy. When an input graph has a resource of type $dbo:Person$, Jena infers that it's also of type $dbo:Agent$ since $dbo:Person$ is a subclass of $dbo:Agent$. For the person-place graphs, this constitutes noise propagation because $dbo:Agent$ and $dbo:Place$ are disjoint classes. To evaluate noise-tolerance in the deep reasoner, a test is run to check whether it inferred that a person-place is of type $dbo:Agent$. Of the 94 examples, 6 inferences only contain this noisy inference. However, some of the remaining 88 inferences either had false positives or missed valid triples inferred by Jena. 38 inference graphs were *perfect*, containing exactly the inference from Jena minus the noisy triple; examples include: *dbr:Socialist_Republic_of_Croatia*, *dbr:Teylers_Museum* and *dbr:Meta_River*. These make up 40% of the noisy examples. For the remaining person-place inferences, a few contain "false positive" triples not generated by Jena. For example, the deep reasoner inference from the *dbr:Big_Ben* graph, missed these two triples compared to Jena:

```

dbr:Big_Ben a dbo:Agent .
dbr:Big_Ben dul:isDescribedBy
dbr:Gothic_Revival .
  
```

(the first *should* be missed), and generated the following *extra* triple:

```

dbr:Big_Ben a dbo:HistoricPlace .
  
```

It should be noted that this information is not explicitly (i.e. embedded in the DBpedia graph of the resource *dbr:Big_Ben*) nor implicitly (i.e. can be inferred). It is therefore counted as false positive even though it "makes sense". The deep reasoner inferred this information by capturing the generalization that resources with similar links to *dbr:Big_Ben* are usually of type *dbo:HistoricPlace*.

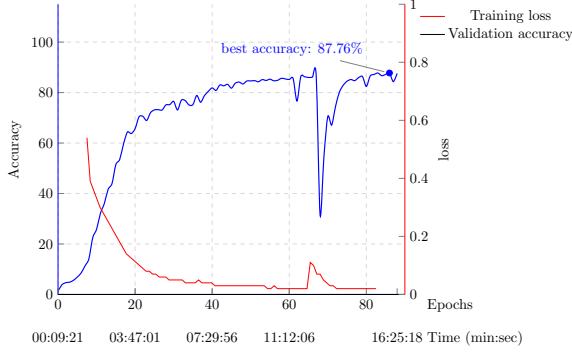


Fig. 15. Training results on DBpedia Scientists dataset

8. Related Work

In this section we review the state of the art in terms of:

- Handling noise in SW data
- Graph embedding (KG embedding in particular)
- Approximate semantic reasoning
- Deep learning for semantic reasoning

8.1. Handling Noise in Semantic Web Data

The strategies for handling noise in SW data can either be active or adaptive:

8.1.1. Active Noise Handling

Most of the work in this category focuses on detecting and fixing noisy data in the LOD. LOD can be created using structured, semi-structured or non structured data. DBpedia [8], for example, is created from semi-structured Wikipedia articles. Non structured texts can also feed NEL tools to create LOD. These two methodologies are more likely to generate noisy triples due to the non perfect accuracy of NEL tools.

In [37], the authors describe two algorithms that they designed to improve the quality of LOD. The *SDType* algorithm falls into the category of adaptive noise handling and will be described in the corresponding section. *SDValidate* identifies wrong triples when there is a large deviation between the resource types. The main idea of this algorithm is to assign a *relative predicate frequency*—describing the frequency of predicate/object combinations—for every statement. Probability distributions are then used to decide if a statement with low *relative predicate frequency* should be considered erroneous. Both algorithms are validated

on DBpedia and Never-Ending Language Learning (NEL) [38] knowledge bases.

In [13], the authors focus on detecting noisy type assertions. They built a few synthetic noisy datasets based on LUBM. Then a multi-class classifier is trained to learn disjoint classes.

In [39, 40], the focus is on incorrect numerical data in LOD datasets. [39] uses a two phase detection approach. In the first phase, outliers of numerical values are detected for every property and in the second phase, the *owl:sameAs* property is used to confirm or reject the outliers. [40] uses a few unsupervised learning techniques including Kernel Density Estimation (KDE) [41] combined with semantic grouping to identify the outliers.

8.1.2. Adaptive Noise Handling

Given the unrealistic expectation of cleansing every type of noise in SW data, adaptive noise handling approaches focus rather on building techniques that are noise-tolerant. In the *SDType* algorithm [37, 42], the *rdf:type* inference uses information from the ABox rather than ontological descriptions from the TBox. For instance, instead of using the *rdfs:domain* and *rdfs:range* of the properties to infer the resources' types, which will propagate noise, a weighted voting heuristic is used instead to determine the types of the resources. The weights are generated from the statistical distribution between predicates and types. For example, given that the property *dbo:location* is mostly connected to objects of type *dbo:Place*, then this property will have high weight to infer the type *dbo:Place*.

To the best of our knowledge, most of the previous work in the literature about reasoning with noisy SW data focuses on type inference. This research is the first to aim at full RDFS reasoning with noise-tolerance capability.

8.2. Graph Embedding

This review is partially based on three recent surveys of graph embedding techniques and their applications [5, 43, 44]. We update the latter survey by including the work on RDF graph embedding. The authors of [43] also provide an open source Python library (Graph Embedding Methods) for graph embedding comparison that we used to compare the discussed embedding techniques on RDF graphs.

It is needless to stress the omnipresence of graph based representations for research problems and real world applications ranging from social network anal-

ysis to recommendation systems to protein interaction networks to knowledge graphs and SW graphs in particular. This can be considered as the main motive for graph analytics research. Graph analytics tasks include centrality analysis, nodes classification [45], link prediction [46] etc. The latter is the closest to our research because the inference RDF graph can be seen as the link prediction applied to the input graph.

8.2.1. Why Embedding Graphs?

In performing the previous tasks of graph analytics, two of the main challenges — especially when processing large scale graphs— are size and time complexity. One technique that tackles these challenges is graph embedding. In a nutshell, the embedding consists of finding a mapping from the original space to a continuous vector space of lower dimension while preserving certain required properties. In graph embedding, the desired properties to be preserved can be node proximity, node similarities or dissimilarities, structural proximity etc.

8.2.2. How to Embed Graphs?

In order to briefly describe the embedding process, a few preliminary notions from [43] should be introduced. Let:

S be the adjacency matrix of the graph $G(V, E)$ where:

$$s_{i,j} = \begin{cases} 0 & \text{if the nodes } v_i \text{ and } v_j \text{ are not connected} \\ w_{i,j} & \text{the weight of the edge } e_{i,j} \end{cases}$$

The first-order proximity between two nodes is defined as the weight of their edge.

The second-order proximity between two nodes is defined by the similarity between their respective immediate neighbors. More formally, let s_i and s_j be the i -th and j -th row vectors of the adjacency matrix respectively. These row vectors represent the first-order proximity between a given node and all the other nodes of the graph. The distance between s_i and s_j represents the second-order proximity between the nodes v_i and v_j .

Similarly, higher order proximity can be defined using the second-order proximity. Using these preliminary notions, [43] defines graph embedding as:

Given a graph $G = (V, E)$, a graph embedding is a mapping $f : v_i \rightarrow y_i \in \mathbb{R}^d \forall i \in [n]$ such that $d \ll |V|$ and the function f preserves some proximity measure defined on graph G . [43, 2]

$[n]$ denotes the set of indices $\{1, 2, \dots, n\}$.

8.2.3. Graph embedding methods

Based on the techniques used to compute such embeddings, a taxonomy for graph embedding approaches can be drawn:

Matrix Factorization Methods Matrix factorization consists of decomposing a matrix into two or more matrices where their product regenerates the original matrix. Graph embedding techniques using matrix factorization start by generating a matrix representation of the graph and then compute the factorization to obtain the embedding. In its simplest form, the matrix representation of the graph can just be the nodes' adjacency matrix S . Other matrix representations of the graph include the Laplacian matrix [47] and the Katz similarity matrix [48], which measure the nodes' centrality. A few examples of graph embedding approaches using matrix factorization are: Locally Linear Embedding [49], Graph factorization [50] and HOPE [51]. The authors of the HOPE algorithm aimed to preserve the asymmetric transitivity property, which is an important property in directed graphs. The feature of preserving the asymmetric transitivity is desirable in RDF graphs embedding as the *rdfs:subPropertyOf* and *rdfs:subClassOf* are asymmetric transitive properties. In order to speedup the matrix factorization of sparse matrices, the authors of HOPE use singular-value decomposition.

Random Walks Methods [52] defines random walks on graphs by:

Given a graph and a starting point, we select a neighbor of it at random, and move to this neighbor; then we select a neighbor of this point at random, and move to it etc. The (random) sequence of points selected this way is a random walk on the graph. [52, 1]

When the size of the graph is too large to traverse in a reasonable time and space complexity, random walks can be used to approximate the computation of certain properties of the graph. In node2vec [53], the authors compute biased-random walks to obtain a balanced traversal between depth first and breadth first traversal. Then they apply a similar technique to word2vec [54] by considering the graph walks as sentences to compute the embedding.

Graph Neural Network Models One of the earliest work that proposes a framework for consuming graph data by neural networks is GNN [55]. Deep autoencoders can be used for dimensionality reduction. Deep

graph embedding techniques use this ability to reduce the dimension of the matrix representing the graph. The authors of [56] propose a Graph Convolutional Network (GCN) model which is a variant of convolutional neural networks that operates on graphs. [57] applies variational autoencoders— where the encoder part is a graph convolutional network— in order to improve the embedding quality of unsupervised techniques.

8.2.4. Embedding of Knowledge Graphs

[44] classifies the embedding approaches of KG facts into:

- **Translational distance models.** In the translational model TransE [58], both the head of the fact h (subject in RDF terminology) and the tail of the fact t (object in RDF terminology) are embedded in the same vector space.

Let:

\mathbb{R}^d : be the embedding space where d is the embedding dimension.

\mathbf{h} : be the vector representation in \mathbb{R}^d of the head entity h .

\mathbf{t} : be the vector representation in \mathbb{R}^d of the tail entity t .

In these translational models, the relation r (predicate in RDF terminology) is represented as a translation vector \mathbf{r} such that $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$. A scoring function $f_r(h, t)$ is defined to assign a plausibility score to the facts of the KG. The TransE model [58] does not support facts with 1-N relations— such as a student taking many courses— as all the courses in this case will have very close embedding vectors. The literature contains variations of the TransE model that support 1-N relations such as TransH [32] which uses relation-specific hyperplanes, TransR [59] and TransD [60]. Gaussian embeddings in this class such as KG2E [61] aim to model uncertainty in the entities and relations.

- **Semantic matching models.** In the semantic matching models, the entities are represented by their latent semantic attributes and their relations “are encoded as bilinear operators on the entities [62, 3]”. In other terms, each relation is denoted as a matrix \mathbf{M}_r that represents the pairwise relations between the entities. The score of the fact plausibility in these models is computed by this bilinear map: $f_r(h, t) = \mathbf{h}^T \mathbf{M}_r \mathbf{t}$. This category includes RESCAL [63], DistMul [64] where \mathbf{M}_r

is simplified to a diagonal matrix, ComplEx [65] which extends DistMul by using complex valued embedding in order to support asymmetric relations. HolE (Holographic Embeddings) [34] also supports asymmetric relations through circular correlations between the entities’ embeddings.

Neural Network Architectures for KG embedding
The network models proposed in the literature for learning KG embeddings include:

- *Semantic matching energy* [66] which computes the energy by matching the embedding of a left hand side containing the head and the relation of the triple and the embedding of the right hand side containing the tail and the relation of the triple.
- *Neural tensor network (NTN)* [67] proposed an end-to-end deep neural network model that is parameterized by a 3-way tensor representing the relation in order to learn the plausibility of triples in a KG.
- *Relational Graph Convolutional Networks (R-GCNs)* [68] adapts GCN [56] to KGs by introducing transformations that are dependent on the type and direction of the edges.

Embedding of RDF Graphs RDF embedding techniques can be classified into:

- **Graph kernels for RDF.** One of the earliest work in this class was [69] where the authors apply general graph kernel methods on RDF graphs and propose two kernels that are specific to RDF, namely intersection graph kernels and intersection tree kernels. In [70], the authors consider state of the art graph kernels which are Weisfeiler-Lehman graph kernels [71] and adapt them to RDF graphs. [72] proposes an h-hop neighborhood-based graph kernel for LOD and they apply it in a linked data recommender system.
- **2vec RDF embedding.** These approaches use the following generic method: generate sequences of entities from the RDF graph using graph walks or other graph kernels and then apply a technique similar to word2vec [73]— where each entity in the sequence is treated as a word in a sentence. In RDF2Vec [74], the sequences are generated using graph walks and using the Weisfeiler-Lehman adaptation to RDF graphs [70] mentioned previously. [75] improves RDF2Vec by using biased graph walks to generate the entities’ sequences. In order to explore the global patterns of the RDF

graph instead of the local patterns as in RDF2Vec, [76] substitute word2vec with a technique similar to GloVe (Global Vectors) [77]. The authors report similar embedding quality as RDF2Vec but with the ability to incorporate larger portions of the graph.

Knowledge graph completion One of the main application of KG embedding is Knowledge Graph Completion (KGC). Data-driven KGC literature includes [33, 78–82]. Logic Tensor Networks [81] allow the definition of logical constraints to improve the KGC. [82] aims not only at predicting missing relations in a KG but also at inducing the logical rules from it.

8.3. Approximate semantic reasoning

In 2010, Hitzler and van Harmelen called in [83] for questioning the model-theoretic semantic reasoning and investigation of machine learning (ML) for semantic reasoning since ML techniques are more tolerant to noisy data.

8.3.1. Type Inference

Type inference consists of inferring the corresponding classes from the TBox for resources in the ABox. It can be considered as a main step towards full RDFS reasoning as almost half of the rules in RDFS – namely *RDFS1*, *RDFS2*, *RDFS3*, *RDFS4a*, *RDFS4b* and *RDFS9* [7]– generate type inference.

SDType algorithm [37, 42] (mentioned previously) used a statistical distribution of types to predict the type of object and subject in a triple given that they are connected with a certain property. Their statistical approach makes this type inference mechanism robust to noise in RDF data. [84] targeted inferring the missing types in DBpedia resources through an inductive and an abductive approach. In the inductive approach, the k-Nearest Neighbors algorithm is used to determine the closest concepts from the DBpedia ontology to which the resource should be linked. In the abductive approach, the Encyclopedia Knowledge Paths [85] are used in a similarity metric.

8.3.2. Consistency Checking

In [86], the authors aimed to detect systematic errors in DBpedia by aligning the DBpedia ontology and the upper level ontology DOLCE-Zero [87]. By clustering the reasoning results, they found that 40 clusters cover 96% of the inconsistencies. This observation was among the motivations that approximate semantic reasoning can cover most of the use cases where

ontological reasoning is required. In order to speedup the process of ABox consistency checking, [88] used an approximate semantic reasoning approach based on machine learning. The authors formalized the problem as a binary classification problem where each classifier *C* is trained for a specific TBox to decide if any ABox is consistent or not with respect to the TBox. In order to transform the RDF graphs into feature vectors, graph walks [89] were used. The decision tree model achieves 95% accuracy within 2% of the time required by a semantic reasoner.

In [90], the authors extend the clash queries [91] for *DL-Lite* [92] and caching to reduce the required calls to a semantic reasoner in order to check ABox inconsistency. This approach had better running time and empirical accuracy than [88].

8.4. Deep Learning for semantic reasoning

One of the closest research efforts to the scope of this research is [93]. Besides the used neural network model, the main difference between their approach and ours is that they consider only learning from intact data and do not focus on noise-tolerance capabilities. In this work, Relational Tensor Networks (RTN) are proposed as an adaptation of Recursive Neural Tensor Networks (RNTN) [94] for relational learning. RNTNs were originally designed by Socher to support learning from tree-structured data such as sentences' parse trees and they were used successfully to improve sentiment analysis results. In [93], the authors start by building a Directed Acyclic Graph (DAG) representation of the RDF input. Every resource in the graph is initially represented as an incidence vector that indicates the set of *rdf:type(s)* of the resource. Then the embeddings of the resources are computed using the RTN model that takes into consideration the type or the relation that each resource has. Two types of targets are considered: a unary target for type prediction and a binary target for predicate classification. The input for the binary targets are the embeddings of two resources— to which the predicates are being classified.

9. Conclusions, Discussions and Future Work

The main contribution of this paper is the empirical evidence that deep learning (neural networks translation in particular) can in fact be used to learn semantic reasoning— RDFS rules specifically. The goal was not to reinvent the wheel and design a Yet another Seman-

tic Reasoner (YaSR) using a new technology; it was rather to fill a gap that existing rule-based semantic reasoners could not satisfy, which is noise-tolerance.

While the current approach proves empirically that RDFS rules are learnable by sequence-to-sequence models with noise-tolerant reasoning capabilities, it is barely a scratch on the surface of noise-tolerant reasoning in general. This research can be extended in the following directions:

9.1. Generative adversarial model for graph words

The experiments on controlled noisy datasets from LUBM1 showed that the noise-tolerance capability of the deep reasoner depends on the type of noise—specifically the noise-tolerance on noisy type assertions is better than the noise-tolerance on noisy property assertions. In the propagable noise cases—where Jena or any rule-based reasoner generates noisy inferences—the deep reasoner showed noise-tolerance with varying degrees of accuracy (from 93% to 46%). However, for the non-propagable noise cases—that do not affect rule reasoners inference—Jena performed better than the deep reasoner. For the special case of *WOAD* noise, both Jena and the deep reasoner have the worst accuracy of 0%. In these experiments, the training was performed on intact data and noise was seen only during the test phase. One way to improve the noise-tolerance capability for these cases is to induce a small percentage of noise in the training set as well. Our previous experiments on the naive sequence-to-sequence learning for RDFS reasoning [95] proved that training with a small percentage of noise improves the noise-tolerance capability dramatically. Instead of generating noise of a specific type—which assumes the prior knowledge of the type of noise encountered during the test phase—we propose designing generative adversarial models for graph words. Generative adversarial models, described in [96], are being used successfully in other fields to add robustness to unknown types of noise. In these models, two networks were trained while competing with each other: the generator is trained to generate the most difficult sample that can fool the discriminator into thinking that the sample is not noisy, and the discriminator is trained to distinguish between noisy and intact samples. The deep reasoner will then learn from the ground truth graph words as well as the corrupted graph words generated by the adversarial generator.

9.2. OWL Reasoning

In this work, we tackled the problem of noise-tolerant RDFS reasoning. Web Ontology Language (OWL) reasoning with noise-tolerant capability is also a very promising research track that can find its applications in the biological and biomedical fields for example. We investigated some use cases using ontologies from the Open Biological and Biomedical Ontology (OBO) Foundry [97], specifically using the Human Disease Ontology [98]. In this use case, some patients' descriptions would contain misdiagnoses and the goal is to generate correct inferences with the presence of these misdiagnoses. The hurdle that we faced in proceeding with this use case was ground truthing, as we needed patients' data with tagged noise. In this context, tagged noise means that the misdiagnosed cases are known. This is required to compare the inference from intact data versus the inference from noisy data.

In [95], we tested the naive sequence-to-sequence learning approach on a subset of OWL-RL rules. This subset includes what we call *generative rules* that generate inference triples and exclude the consistency checking rules. The performance of the naive sequence-to-sequence approach on OWL-RL rules was comparable to its performance on RDFS rules. This is a preliminary indication that the graph words translation approach can also be applicable to learning *OWL-RL* rules.

9.3. Training with multiple "ABoxes"

Another limitation to the current approach is that the training is done on a dataset that uses only one ontology for the inference. After training the graph-to-graph model on the LUBM1 dataset, we needed to adapt the model hyperparameters for the scientists' dataset and start the training from scratch. We propose exploring transfer learning: Instead of starting the training process from scratch when training to infer using a new ontology, the neural network weights from the previous training can be used to initialize the new model. Transfer learning [99] aims to capitalize on the knowledge learned from one domain and adapt it to a new domain. The adaptation phase in neural networks consists of tuning the model weights after initializing them using the previous models' weights. Research in this direction looks promising especially when transferring weights between models of different width. The width of the model is determined by the length of the graph words sequence.

9.4. Towards the trust layer

In a recent positional paper titled “Semantic Web: Learning from Machine Learning” [100], Brickley describes his vision of how deep learning and SW fields can communicate and learn from each other. In this paper, we initiated the communication in one direction which is: deep learning for SW. The other direction, SW for deep learning, is also equally important and very promising with lots of opportunities for research and subsequent discovery. One such research effort in that direction is [101] where the authors use SW technologies to describe the inputs and outputs of neural networks.

We believe that our deep learning for noise-tolerant semantic reasoning contribution can be extended into a hub where both fields can communicate and benefit from each other. One way to create this hub is through provenance-based reasoning. Imagine that the deep reasoner will not only have access to the erroneous triple in DBpedia but to the provenance of that triple i.e. the person who originally edited the Wikipedia page and input the wrong information. By detecting that most of the triples provenant from that user causes the reasoner to be in noise-tolerance mode, it should not only ignore the triples generated by that user but also assign a trust level to its “facts”. This can be a step towards the trust layer in the SW layers cake.

Acknowledgements

We would like to thank DARPA SMISC, AFRL NS-CTA and IBM HEALS for sponsoring different stages of this research.

Appendix A. RDFS rules

Table 5
RDFS rules (from [7])

Rule	Premise	Conclusion
RDFS1	any IRI aaa in D	aaa RDF:type RDFS:Datatype .
RDFS2	aaa RDFS:domain xxx . yyy aaa zzz .	yyy RDF:type xxx .
RDFS3	aaa RDFS:range xxx . yyy aaa zzz .	zzz rdf:type xxx .
RDFS4a	xxx aaa yyy .	xxx rdf:type RDFS:Resource .
RDFS4b	xxx aaa yyy.	yyy rdf:type RDFS:Resource .
RDFS5	xxx RDFS:subPropertyOf yyy . yyy RDFS:subPropertyOf zzz .	xxx RDFS:subPropertyOf zzz .
RDFS6	xxx rdf:type rdf:Property .	xxx RDFS:subPropertyOf xxx .
RDFS7	aaa RDFS:subPropertyOf bbb . xxx aaa yyy .	xxx bbb yyy .
RDFS8	xxx rdf:type RDFS:Class .	xxx RDFS:subClassOf RDFS:Resource .
RDFS9	xxx RDFS:subClassOf yyy . zzz rdf:type xxx .	zzz rdf:type yyy .
RDFS10	xxx rdf:type RDFS:Class .	xxx RDFS:subClassOf xxx .
RDFS11	xxx RDFS:subClassOf yyy . yyy RDFS:subClassOf zzz .	xxx RDFS:subClassOf zzz .
RDFS12	xxx rdf:type RDFS:ContainerMembershipProperty .	xxx RDFS:subPropertyOf RDFS:member .
RDFS13	xxx rdf:type RDFS:Datatype .	xxx RDFS:subClassOf RDFS:Literal .

Appendix B. Pattern types of RDFS rules' premises

Table 6
Pattern types of RDFS rules' premises

Rule	Premise	Pattern type
RDFS2	aaa RDFS:domain xxx .	TBox
	yyy aaa zzz .	ABox
RDFS3	aaa RDFS:range xxx .	TBox
	yyy aaa zzz .	ABox
RDFS5	xxx RDFS:subPropertyOf yyy .	TBox
	yyy RDFS:subPropertyOf zzz .	TBox
RDFS6	xxx rdf:type rdf:Property .	TBox
RDFS7	aaa RDFS:subPropertyOf bbb .	TBox
	xxx aaa yyy .	ABox
RDFS8	xxx rdf:type RDFS:Class .	TBox
RDFS9	xxx RDFS:subClassOf yyy .	TBox
	zzz rdf:type xxx .	ABox
RDFS10	xxx rdf:type RDFS:Class .	TBox
RDFS11	xxx RDFS:subClassOf yyy .	TBox
	yyy RDFS:subClassOf zzz .	TBox

Appendix C. Propagable noise by rule-based RDFS reasoners

Table 7
Propagable noise by rule-based RDFS reasoners

RDFS rule	Triple corruption	Conditions	Noisy inference
RDFS2		(aaa' rdfs:domain xxx' .)	
	yyy aaa zzz .	$\wedge ((\neg \exists \text{xxx}, \text{aaa rdfs:domain xxx} .))$	
	\rightarrow	\vee	yyy rdf:type xxx' .
	yyy aaa' zzz .	$(\forall \text{xxx}, \text{aaa rdfs:domain xxx} \implies \neg(\text{xxx} = \text{xxx'}))$	
RDFS3		(aaa' rdfs:range xxx' .)	
	yyy aaa zzz .	$\wedge ((\neg \exists \text{xxx}, \text{aaa rdfs:range xxx} .))$	
	\rightarrow	\vee	zzz rdf:type xxx' .
	yyy aaa' zzz .	$(\forall \text{xxx}, \text{aaa rdfs:range xxx} \implies \neg(\text{xxx} = \text{xxx'}))$	
RDFS7		(aaa' rdfs:subPropertyOf bbb' .)	
	xxx aaa yyy .	$\wedge ((\neg \exists \text{bbb}, \text{aaa rdfs:subPropertyOf bbb} .))$	
	\rightarrow	\vee	xxx bbb' yyy .
	xxx aaa' yyy .	$(\forall \text{bbb}, \text{aaa rdfs:subPropertyOf bbb} . \implies \neg(\text{bbb} = \text{bbb'}))$	

Appendix D. Input graph g

Listing 1: Input graph g

```

Publication2 ub:publicationAuthor GraduateStudent9 .
Publication6 ub:publicationAuthor GraduateStudent9 .
Publication17 ub:publicationAuthor GraduateStudent9 .
Publication11 ub:publicationAuthor GraduateStudent9 .
Publication15 ub:publicationAuthor GraduateStudent9 .

GraduateStudent9 a ub:GraduateStudent ;
    ub:advisor FullProfessor7 ;
    ub:emailAddress
        "GraduateStudent9@Department5.University0.edu" ;
    ub:memberOf <http://www.Department5.University0.edu> ;
    ub:name "GraduateStudent9" ;
    ub:takesCourse GraduateCourse39 ;
    ub:telephone "xxx-xxx-xxxx" ;
    ub:undergraduateDegreeFrom
        <http://www.University718.edu> .

```

Appendix E. Inference graph of the input graph in Listing 1

Listing 2: Inference graph of the input graph in listing 1

```

Publication2 a ub:Publication .
Publication6 a ub:Publication .
Publication17 a ub:Publication .
Publication11 a ub:Publication .
Publication15 a ub:Publication .

FullProfessor7 a ub:Employee,
    ub:Faculty,
    ub:Professor .

GraduateStudent9 a ub:Person ;
    ub:degreeFrom <http://www.University718.edu> .

<http://www.University718.edu> a ub:Organization,
    ub:University .

```

Appendix F. RDF Graph Formalism

An RDF graph can be defined using these formalisms from [17, 102, 103] (that is updated in this paper to conform to the more recent RDF 1.1 recommendation [15]):

Let:

I be an infinite set of Internationalized Resource Identifier (IRI) (which is an extension of Uniform Resource Identifier (URI) that supports Unicode characters).

B be an infinite set of Blank nodes

L be an infinite set of RDF literals

A tuple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an RDF triple where s denotes the triple's subject, p denotes its predicate and o denotes its object.

An RDF graph is a set of RDF triples.

$T = \{ (s, p, o) \mid (s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L) \}$

Let:

$\text{Subj}(T)$ be the set of subjects from $(I \cup B)$ that occur in the triples of T
 $\text{Pred}(T)$ be the set of predicates from I that occur in the triples of T
 $\text{Obj}(T)$ be the set of objects from $(I \cup B \cup L)$ that occur in the triples of T
 $\text{SubjObj}(T) = \text{Subj}(T) \cup \text{Obj}(T)$

Appendix G. Layered Graph Examples

Let the tuple of properties in P^+ for LUBM have the following order: (*rdf:type*, *ub:takesCourse*, *ub:advisor*, *ub:emailAddress*, *ub:memberOf*, *ub:name*, *ub:telephone*, *ub:undergraduateDegreeFrom*, *ub:publicationAuthor*, *ub:degree*).

The RDF graph in Listing 1 has the corresponding layered graph in Listing 3 and its inference graph has the layered graph listed in Listing 4.

Listing 3: Layered graph (indented and commented for readability) of the RDF graph in Listing 1

```

(
  ((GraduateStudent9, ub:GraduateStudent)), #Layer of rdf:type
  ((GraduateStudent9, GraduateCourse39)), #Layer of ub:takesCourse
  ((GraduateStudent9, FullProfessor7)), #Layer of ub:advisor
  ((GraduateStudent9, GraduateStudent9@Department5.University0.edu)), #Layer of ub:emailAddress
  ((GraduateStudent9, http://www.Department5.University0.edu)), #Layer of ub:memberOf
  ((GraduateStudent9, "GraduateStudent9")), #Layer of ub:name
  ((GraduateStudent9, "xxx-xxx-xxxx")), #Layer of ub:telephone
  ((GraduateStudent9, http://www.University718.edu)), #Layer of ub:undergraduateDegreeFrom
  ((Publication11, GraduateStudent9),
    (Publication15, GraduateStudent9),
    (Publication2, GraduateStudent9),
    (Publication17, GraduateStudent9),
    (Publication6, GraduateStudent9)), #Layer of ub:publicationAuthor
  () #Layer of ub:degree
)

```

Listing 4: Layered graph of the inference graph in Listing 2

```

(
  ((GraduateStudent9, ub:Person),
    (Publication6, ub:Publication),
    (Publication17, ub:Publication),
    (Publication11, ub:Publication),
    (Publication15, ub:Publication),
    (FullProfessor7, ub:Employee),
    (FullProfessor7, ub:Faculty),
    (FullProfessor7, ub:Professor),
    (http://www.University718.ed, ub:Organization),
    (http://www.University718.ed, ub:University)), #Layer of rdf:type
  (), #Layer of ub:takesCourse
  (), #Layer of ub:advisor
  (), #Layer of ub:emailAddress
  (), #Layer of ub:memberOf
  (), #Layer of ub:name
  (), #Layer of ub:telephone
  (), #Layer of ub:undergraduateDegreeFrom
  (), #Layer of ub:publicationAuthor
  ((GraduateStudent9, http://www.University718.edu)) #Layer of ub:degree
)

```

Appendix H. Proof of the Tensor Creation Requirement

Proposition 1. The encoding dictionaries of the input graph and its corresponding inference graph must be equal.

Proof by contradiction. Assuming that during the training phase the input graph and its corresponding inference graph are encoded independently (allowing their encoding dictionaries to be different):

In the inference phase the test input graph is encoded then the trained model is used to predict the encoded version of the inference graph. Because there is access to only one encoding dictionary— which is the input graph encoding dictionary— it has to be used in the decoding algorithm to obtain the inference graph.

This proves that the encoding dictionary for the input graph and inference graph in the training phase must be the same. \square

Corollary 1. The encoding dictionary of the input graph should contain all the possible resources of the inference graph.

Proof. For the encoding dictionaries of the input graph and the inference graph to be equal, the encoding algorithm of the inference graph should only use lookups from the encoding dictionary without adding any new resources. This means that all the resources of the inference graph were already added to the encoding dictionary when encoding the input graph. \square

Hence, it is mandatory that the encoding dictionary for a given graph g contains all the possible resources that might be used in its corresponding inference graph i .

Appendix I. Tensor Creation Detailed Algorithms

In order to fulfill the requirement established in Proposition 1, it is mandatory that the *global_resources_dictionary* and the *local_resources_dictionary* for a given graph g contain all the possible resources that might be used in its corresponding inference graph i . To create the properties dictionary *Properties_dictionary*, the list of properties is collected using the following SPARQL Protocol and RDF Query Language (SPARQL) query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
select distinct ?property where {
  ?property a rdf:Property .
  ?subject ?property ?object .
}
```

which returns all the properties in the ontology that were used at least once. An ID is then assigned to each property. In the LUBM1 dataset, this query gives 32 properties, which means that the 3D adjacency matrix will have 32 layers.

For the global resources dictionary, the list of RDFS classes are collected from the ontology using this SPARQL query:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select distinct ?class where {
  ?class a rdfs:Class .
  filter(isuri(?class))
}
```

A filter is used to eliminate blank nodes. In the LUBM1 dataset, this query returns 57 classes where each class is assigned an ID in a *global_resources_dictionary*.

The *local_resources_dictionary* is created incrementally during the encoding routine for each graph g in G . It holds the IDs of the resources that are not present in the *global_resources_dictionary*. The *local_resources_dictionary* is populated with an offset equal to the length of *global_resources_dictionary* i.e. 57 in the case of LUBM1. The largest ID in the *local_resources_dictionary* for every graph in G is less than 80. This value will be used to initialize the size of the 3D adjacency matrix.

1.1. Encoding Algorithm

Once the *properties_dictionary* and the *global_resources_dictionary* are created, they are used in the encoding routine listed in Algorithm 1. The function ZEROS in Algorithm 1 creates a 3D matrix of the desired shape filled with zeros, and the function SORTEDTRIPLESBYPROPERTY lists the triples of the input graph sorted by the property attribute. When encoding an RDF graph, the triples having a property not listed in the *properties_dictionary* are ignored because they will not have any effect on the inference generation. Subsequent to encoding the input graph *g* and creating the *local_resources_dictionary*, the latter is used to encode the corresponding inference graph *i*. The inference graph encoding algorithm is very similar to the input graph encoding. It is crucial not to update the *local_resources_dictionary* since all the resources in the inference graph should already be either in the *global_resources_dictionary* or in the *local_resources_dictionary*.

Algorithm 1 Simplified encoding algorithm

```

Input: rdf_graph,                                /* The RDF graph to be encoded */
        properties_dictionary,                      /* A dictionary containing the IDs of the properties */
        global_resources_dictionary,               /* A dictionary containing the IDs of the subjects and objects resources in the
        ontology */
        local_resources_dictionary,               /* If encoding an input graph, this dictionary is empty and will be filled during the
        encoding, and if encoding an inference graph, this dictionary contains the IDs of the local subjects' and objects'
        resources */
Parameter: is_inference                        /* A boolean set to True if encoding the inference graph and to False otherwise */
        max_local_dictionary_size                  /* The size of the biggest local_resources_dictionary */
Output: adjacency_matrix                       /* 3D adjacency matrix containing an encoded representation of the RDF graph */
        local_resources_dictionary                 /* The filled local_resources_dictionary if encoding an input graph */

Begin:
    number_of_properties ← SIZE(properties_dictionary)
    max_size ← max_local_dictionary_size + SIZE(global_resources_dictionary)
    adjacency_matrix ← ZEROS(number_of_properties, max_size, max_size)
    function ADD_RESOURCE(resource)
    |   if resource in global_resources_dictionary ↵
    |       or resource in local_resources_dictionary then
    |       |   return
    |   else
    |       |   local_resources_dictionary[resource] ← ↵
    |       |       SIZE(local_resources_dictionary) + SIZE(global_resources_dictionary)
    |       /* We offset the IDs in the local_resources_dictionary with the size of the global_resources_dictionary so their
    |       IDs do not overlap */
    function LOOKUP_RESOURCE(resource)
    |   if resource in global_resources_dictionary then
    |       |   return global_resources_dictionary[resource]
    |   else if resource in local_resources_dictionary then
    |       |   return local_resources_dictionary[resource]
    |   else
    |       |   ERROR, EXIT
    function ENCODE(rdf_graph, global_resources_dictionary, local_resources_dictionary, properties_dictionary,
    is_inference)
    |   for all (s,p,o) in SORTED_TRIPLES_BY_PROPERTY(rdf_graph) do
    |       |   if p not in properties_dictionary then
    |       |       |   continue
    |       |       p_id ← properties_dictionary[p]

```

```

1      |   if not is_inference then
2      |       |   ADD_RESOURCE(s)
3      |       |   ADD_RESOURCE(o)
4      |       s_id ← LOOKUP_RESOURCE(s)
5      |       o_id ← LOOKUP_RESOURCE(o)
6      |       adjacency_matrix[p_id, s_id, o_id] ← 1
7      |   return adjacency_matrix, local_resources_dictionary
8  End

```

1.2. Decoding Algorithm

The decoding algorithm takes a 3D adjacency matrix and the resources dictionaries as inputs, and regenerates the original RDF graph. To be more precise, the decoding algorithm will regenerate the original RDF graph for every graph i in I . Nevertheless, the graphs in G can be different from their decoded graphs because the properties that are not present in the *properties_dictionary* are disregarded during the encoding process. This is irrelevant to the process of inference learning: both the regenerated graph and the original RDF graph should have the same inference graph because the ignored properties are not from the ontology.

The NON_ZEROS routine in Algorithm 2 returns the list of 3-tuples containing the indices of the non-zeros values. The INVERT method for dictionaries returns the dictionary with the values as keys and vice versa.

Algorithm 2 Simplified decoding algorithm

```

21 Input: adjacency_matrix,                               /* 3D adjacency matrix containing */
22         global_resources_dictionary, local_resources_dictionary, properties_dictionary
23 Output: rdf_graph
24 Begin:
25     rdf_graph ← GRAPH()                                /* Creating an empty RDF graph */
26     inverted_properties_dictionary ← INVERT(properties_dictionary)
27     inverted_global_resources_dictionary ← INVERT(global_resources_dictionary)
28     inverted_local_resources_dictionary ← INVERT(local_resources_dictionary)
29     function REVERSE_LOOKUP(resource_id)
30     |   if resource_id in inverted_global_resources_dictionary then
31     |       |   return inverted_global_resources_dictionary[resource_id]
32     |   else if resource_id in inverted_local_resources_dictionary then
33     |       |   return inverted_local_resources_dictionary[resource_id]
34     |   else
35     |       |   ERROR, EXIT
36
37     function DECODE(adjacency_matrix, inverted_global_resources_dictionary, inverted_local_resources_dictionary,
38     inverted_properties_dictionary)
39     |   for all (p_id,s_id,o_id) in NON_ZEROS(adjacency_matrix) do
40     |       |   p ← inverted_properties_dictionary[p_id]
41     |       |   s ← REVERSE_LOOKUP(s_id)
42     |       |   o ← REVERSE_LOOKUP(o_id)
43     |       |   ADD_TRIPLE(rdf_graph, s, p, o)
44     |   return rdf_graph
45 End

```

Appendix J. Advanced Tensor Creation Technique

According to Corollary 1, for the encoding dictionaries of the input graph and the inference graph to be equal, the encoding algorithm of the inference graph should only use lookups from the encoding dictionary without adding

any new resources. The simplified encoding technique achieved this because all the layers share the same *local_resources_dictionary*. However, by having a *local_resources_dictionary* per layer (i.e. per property) the following issues arise:

3.1.1. Type Inference Challenges

When the type inference rule, *RDFS9*, is applied to this input graph:

```
dbo:Scientist rdfs:subClassOf dbo:Person .
dbr:James_Hendler a dbo:Scientist .
```

it infers the following:

```
dbr:James_Hendler a dbo:Person .
```

The input graph contains the following subject-object resources: *dbo:Scientist*, *dbo:Person* and *dbr:James_Hendler*. When encoding the input graph and populating the resources dictionaries, the first two resources will be already in the *global_resources_dictionary* as they are concepts in the DBpedia ontology and the *dbr:James_Hendler* resource will populate the *local_resources_dictionary* of the layer RDFt.

The inference graph has two subject-object resources: *dbr:James_Hendler* and *dbo:Person*. As they appear in a triple with the property RDFt, first look into the *global_resources_dictionary* and find the ID of the resource *dbo:Person* then in the *local_resources_dictionary* of the property RDFt and find the ID of the resource *dbr:James_Hendler*. In this case all the required resources when encoding the inference graph were inserted in the corresponding dictionaries during the encoding of the input graph. However, this will not be the case for the rules *RDFS2* and *RDFS3*.

When the type inference rule *RDFS3* is applied to this input graph:

```
dbo:almaMater rdfs:range dbo:EducationalInstitution .
dbr:James_Hendler dbo:almaMater dbr:Brown_University .
dbr:James_Hendler dbo:almaMater dbr:Southern_Methodist_University .
dbr:James_Hendler dbo:almaMater dbr:Yale_University .
```

it infers that:

```
dbr:Brown_University a dbo:EducationalInstitution .
dbr:Southern_Methodist_University a dbo:EducationalInstitution .
dbr:Yale_University a dbo:EducationalInstitution .
```

The input graph has the following subject-object resources: *dbr:Brown_University*, *dbr:James_Hendler*, *dbo:EducationalInstitution*, *dbr:Yale_University* and *dbr:Southern_Methodist_University*. When encoding the input graph, the resource *dbo:EducationalInstitution* is found in the *global_resources_dictionary* and the rest of the resources are added to the *local_resources_dictionary* of the layer *dbo:almaMater*. And when encoding the inference graph, in the first triple, the resource *dbr:Brown_University* is looked-up in the *local_resources_dictionary* of the property RDFt but it will not be found as this resource was only added to the layer of the property *dbo:almaMater*. The same problem occurs with the *RDFS2* rule.

Solution: Six out of the fourteen *RDFS* rules are type inference rules i.e. infer a conclusion in the form:

```
yyy rdf:type xxx .
```

Consequently, there is a high chance that any resource *r* in the input graphs will be used in a triple with the pattern

```
r rdf:type xxx .
```

in the inference graph.

The solution to this issue is to simply add all the local resources to the *local_resources_dictionary* of the RDFS property. Whenever any resource is added to the *local_resources_dictionary* of any property when encoding the input graphs, it should be added to the *local_resources_dictionary* of the RDFS property as well. This way, when the corresponding inference graph is encoded, all the required resources will be found in the respective *local_resources_dictionary*.

J.2. SubProperty Challenges

When a property appears only in the inference graph but not in the input graph, the *local_resources_dictionary* for this property will be empty. As a result, all the resources seen in the inference graph will be unknown. For instance, this can happen when the *RDFS7* rule is applied. Consider the following input graph:

```
dbo:field rdfs:subPropertyOf dul:isDescribedBy .
dbr:James_Hendler dbo:field dbr:Artificial_intelligence .
dbr:James_Hendler dbo:field dbr:Semantic_web.
```

which has this inference:

```
dbr:James_Hendler dul:isDescribedBy dbr:Artificial_intelligence .
dbr:James_Hendler dul:isDescribedBy dbr:Semantic_web .
```

When encoding the input graph, the resources *dbr:James_Hendler*, *dbr:Artificial_intelligence* and *dbr:Semantic_web* are added to the *local_resources_dictionary* of the layer *dbo:field*. Subsequently, when encoding the inference graph, lookup these resources in the *local_resources_dictionary* of the layer *dul:isDescribedBy*, but they will not be found as its *local_resources_dictionary* is still empty and will not contain the required resources.

J.2.1. Solution:

The same fix used to solve the type inference case by adding all the resources to every *local_resources_dictionary* results exactly in using the simplified version of the encoding/decoding technique and having a shared *local_resources_dictionary* between all the properties; thus, this fix cannot be applied.

By analyzing the root cause of the issue at hand, it seems logical when encoding the inference graph generated by the *RDFS7* rule to lookup the unknown resources in the *local_resources_dictionary* of the corresponding sub-properties. For example, while encoding the inference graph in this section, when the resource *dbr:Artificial_intelligence* is not found in the *local_resources_dictionary* of the layer *dul:isDescribedBy*, it is looked-up in the *local_resources_dictionary* of its subProperty *dbo:field*. Nonetheless, the property in question can have more than one subProperty, which makes the resources lookup process ambiguous. For instance, the property *dul:isDescribedBy* has two sub-properties: *dbo:field* and *dbo:knownFor*. If a larger excerpt of Professor Hendler's DBpedia graph is considered:

```
dbr:James_Hendler dbo:field dbr:Artificial_intelligence .
dbr:James_Hendler dbo:field dbr:Semantic_web.
dbr:James_Hendler dbo:knownFor dbr:Semantic_Web .
```

it generates the inference:

```
dbr:James_Hendler dul:isDescribedBy dbr:Artificial_intelligence .
dbr:James_Hendler dul:isDescribedBy dbr:Semantic_web.
dbr:James_Hendler dul:isDescribedBy dbr:Semantic_Web.
```

When the input graph is encoded, the resource *dbr:James_Hendler* will have an ID in the *local_resources_dictionary* of the layers *dbo:field* and *dbo:knownFor*. When the inference graph is encoded and lookup of the resources'

IDs for the property *dul:isDescribedBy* is performed, if its sub-properties dictionaries were to be searched two sub-properties dictionaries containing the resource in question probably having different IDs in each dictionary will be found.

This attempt is obviously an unsuccessful fix that one can imagine improving in the following way: Instead of having a *local_resources_dictionary* per property, the *local_resources_dictionary* can be shared between sibling properties (i.e. properties having the same super-property) and their super-properties. In the previous example, the properties *dbo:field*, *dbo:knownFor* and *dul:isDescribedBy* will share the same *local_resources_dictionary*. Again, this is not a fix because some properties can have more than one super-property. For example, in the DBpedia ontology, the property *dbo:capital* is a subPropertyOf *dul:isLocationOf* and *dbo:administrativeHeadCity*. In this case the property *dbo:capital* will have to share its *local_resources_dictionary* with its sibling properties from the super-property *dul:isLocationOf* and also from the super-property *dbo:administrativeHeadCity*.

When the network of the property *rdfs:subPropertyOf* is drawn as shown in Fig. 16, a set of disconnected sub-graphs can be observed— where each subgraph contains the properties having a path connecting them.

By sharing the *local_resources_dictionary* between the properties of each subgraph, the issue at hand is solved. This is because every subgraph contains sibling properties, their super-properties recursively and their sub-properties recursively also. To get the list of these subgraphs, the following SPARQL query is run:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
select ?property1 ?property2
where {
    ?property1 rdfs:subPropertyOf ?property2
    filter(?property1 != ?property2)
}
```

An undirected graph is then created using the Python library networkX [104]. Finally, the connected components of the resultant graph are computed to get the subgraphs which were previously mentioned. In the DBpedia ontology case, 53 connected components were found and in the LUBM ontology only 4 were found. The final working solution for the advanced encoding challenges consists of having a separate *local_resources_dictionary* per group of properties— these groups being the result of the connected components' computation. If a property does not belong to any group, it will automatically have its own *local_resources_dictionary*.

3.2.2. Scaling Challenge

Besides the preceding issues, which are inherent to the advanced encoding technique, scaling is a matter that needs to be taken care of. When using the simplified encoding technique on small graphs and small ontologies, 3D adjacency matrices can be created with the desired dimensions. On the other hand, when dealing with large ontologies with a big number of properties and a large set of subject and object resources, the size of the 3D adjacency matrices becomes unmanageable. For example, the DBpedia ontology contains 3006 properties and 1576 subject and object resources. Thus, even when encoding the smallest possible RDF graph with only one triple containing two local resources, the total size of the matrix becomes $3006 * 1578 * 1578$ which requires approximately 6 Gigabytes of memory for a single RDF graph.

Solution: Even though a large ontology such as DBpedia contains a big number of properties, a smaller number of these properties are usable in a restricted domain dataset such as the Scientists dataset. If the dataset is encoded with the simplified encoding technique, most of the layers throughout the dataset in the 3D adjacency matrices will be empty. The only thing this achieves is the slowing down of the training without having any impact on the training results.

Instead of using a layer for each property from the ontology by utilizing the full *properties_dictionary*, a dictionary of the usable properties needs to be maintained— denoted *usable_properties_dictionary*. The *usable_properties_dictionary* is populated while encoding the dataset. Similarly for the *global_resources_dictionary*, not all the resources in this dictionary will be used in a restricted domain dataset. A *usable_global_resources_dictionary* containing the resources from the *global_resources_dictionary* that are used in the dataset should be maintained.

In the simplified encoding technique, the size of the *local_resources_dictionary* should be known prior to encoding the dataset, because this size should be used to offset the IDs in the *local_resources_dictionary* so that the IDs

in both dictionaries do not overlap. However, as the *usable_global_resources_dictionary* is populated incrementally while encoding the graphs in the dataset, the final size of the *usable_global_resources_dictionary* cannot be known until the whole dataset is encoded. Thus, the IDs of the *local_resources_dictionary* cannot be offset during the encoding in the same way they can be offset in the simplified encoding technique. Instead, the IDs of the local resources dictionaries and the *usable_global_resources_dictionary* should be incremented in opposite directions. For instance, whenever a new resource is added to the *usable_global_resources_dictionary* a positive value equal to the size of *usable_global_resources_dictionary* should be assigned to it. When a new resource is added to the local resources dictionary, a negative ID equal to minus the size of that local dictionary should be assigned to it. After encoding the whole dataset, the IDs in the local resources dictionaries are adjusted using the final size of *usable_global_resources_dictionary* so that no overlaps occur.

The final adjustment that should be applied to the simplified encoding technique to make it more scalable is to apply sparse encoding: instead of creating huge sparse matrices, only the list of indices where these matrices contain the value 1 are maintained.

J.2.3. Advanced Encoding Technique Algorithm

The full algorithm of the advanced encoding technique is detailed in Algorithm 3. The decoding algorithm for the advanced version is very similar to the simplified decoding algorithm.

Appendix K. Advanced Encoding Algorithm

Algorithm 3 Advanced encoding algorithm

Input: *rdf_graph*, *properties_dictionary*, *properties_groups*,
global_resources_dictionary, *usable_properties_dictionary*,
usable_global_resources_dictionary,
local_resources_dictionaries */* The list of local resources dictionaries that will be populated if encoding an input graph */*

Parameter: *is_inference*

Output: *sparse_encoding*,
local_resources_dictionaries */* Not modified if encoding an inference graph */*

Begin:

```

function LOOKUP_RESOURCE(resource, property)
    property_group ← properties_groups[property]
    if resource in usable_global_resources_dictionary then
        return usable_global_resources_dictionary [resource]
    else if resource in local_resources_dictionaries [property_group] then
        return local_resources_dictionaries [property_group][resource]
    else
        ERROR, EXIT

function ADD_RESOURCE(resource, property)
    property_group ← properties_groups[property]
    if resource in usable_global_resources_dictionary then
        return
    else if resource in global_resources_dictionary then
        usable_global_resources_dictionary [resource] ←
            ← SIZE(usable_global_resources_dictionary)
    else if resource not in local_resources_dictionaries [property_group] then
        local_resources_dictionaries [property_group][resource] ←
            ← -(SIZE(local_resources_dictionaries [property_group][resource]))

```



```


1  for all (s,p,o) in SORTED_TRIPLES_BY_PROPERTY(rdf_graph) do
2      if p not in properties_dictionary then
3          continue
4      else if p not in usable_properties_dictionary then
5          usable_properties_dictionary[p] ← SIZE(usable_properties_dictionary)
6          p_id ← usable_properties_dictionary[p]
7          if not is_inference then
8              ADD_RESOURCE(s,p)
9              ADD_RESOURCE(o,p)
10         s_id ← LOOKUP_RESOURCE(s,p)
11         o_id ← LOOKUP_RESOURCE(o,p)
12         APPEND(sparse_encoding, p_id, s_id, o_id)
13 End

```

Appendix L. Graph words creation algorithm

Algorithm 4 From RDF dataset to graph words corpus

```

19 Input: G, /* The set of input RDF graphs */
20 I /* The set of inference RDF graphs */
21 global_resources_dictionary, properties_dictionary
22 Output: X, /* Input corpus */
23 Y, /* Target corpus */
24 G_Catalog, /* Layouts catalog of the input corpus */
25 I_Catalog, /* Layouts catalog of the target corpus */
26 Local_Resources_Dictionaries /* A list containing the local_resources_dictionary */
27 Begin:
28 dataset_size ← SIZE(G)
29 index ← 0
30 X ← [ ]
31 Y ← [ ]
32 G_Catalog ← [ ]
33 I_Catalog ← [ ]
34 Local_Resources_Dictionaries ← [ ]
35 while index < dataset_size do
36     rdf_input ← G[index]
37     inference ← I[index]
38     local_resources_dictionary ← [ ]
39     x_graph_sentence ← [ ]
40     y_graph_sentence ← [ ]
41
42     /* First we encode the input graph */
43     adjacency_matrix, local_resources_dictionary ← 
44     ENCODE(rdf_input, global_resources_dictionary, local_resources_dictionary, properties_dictionary,
45     is_inference=False)
46     Local_Resources_Dictionaries[index] ← local_resources_dictionary
47     for all layer in adjacency_matrix do
48         if LAYOUT(layer) not in G_Catalog then
49             APPEND(G_Catalog, LAYOUT(layer))
50             graph_word ← G_Catalog[LAYOUT(layer)]
51             APPEND(x_graph_sentence, graph_word)

```

```

1   |   X[index] ← x_graph_sentence
2   |   /* Then we encode the inference graph using local_resources_dictionary */
3   |   adjacency_matrix, local_resources_dictionary ←  $\leftarrow \sqsupset$ 
4   |   ENCODE(inference, global_resources_dictionary, local_resources_dictionary, properties_dictionary,
5   |   is_inference=True)
6   |   for all layer in adjacency_matrix do
7   |   |   if LAYOUT(layer) not in I_Catalog then
8   |   |   |   APPEND(I_Catalog, LAYOUT(layer))
9   |   |   |   graph_word ← I_Catalog[LAYOUT(layer)]
10  |   |   |   APPEND(y_graph_sentence, graph_word)
11  |   |   Y[index] ← y_graph_sentence
12  |   |   index ← index + 1
13  |   return X, Y, G_Catalog, I_Catalog, Local_Resources_Dictionaries
14 End

```

Appendix M. Possible Number of Links per Properties per Classes in LUBM1

Table 8
Possible number of links per properties per classes in LUBM1

Classes \ Properties	rdf:type	ub:advisor	ub:teacherOf	ub:researchInterest
ub:GraduateStudent	1, 2	1	0	0
ub:Publication	1	0	0	0
ub:TeachingAssistant	2	1	0	0
ub:ResearchAssistant	2	1	0	0
ub:AssistantProfessor	1	0	2, 3, 4	1
ub:AssociateProfessor	1	0	2, 3, 4	1
ub:Lecturer	1	0	2, 3, 4	0
ub:Course	1	0	0	0
ub:GraduateCourse	1	0	0	0
ub:FullProfessor	1	0	2, 3, 4	1
ub:ResearchGroup	1	0	0	0
ub:Department	1	0	0	0
ub:University	1	0	0	0

Appendix N. The network of the relation *RDFS:subPropertyOf* in the DBpedia ontology (depicted without labels for visibility)

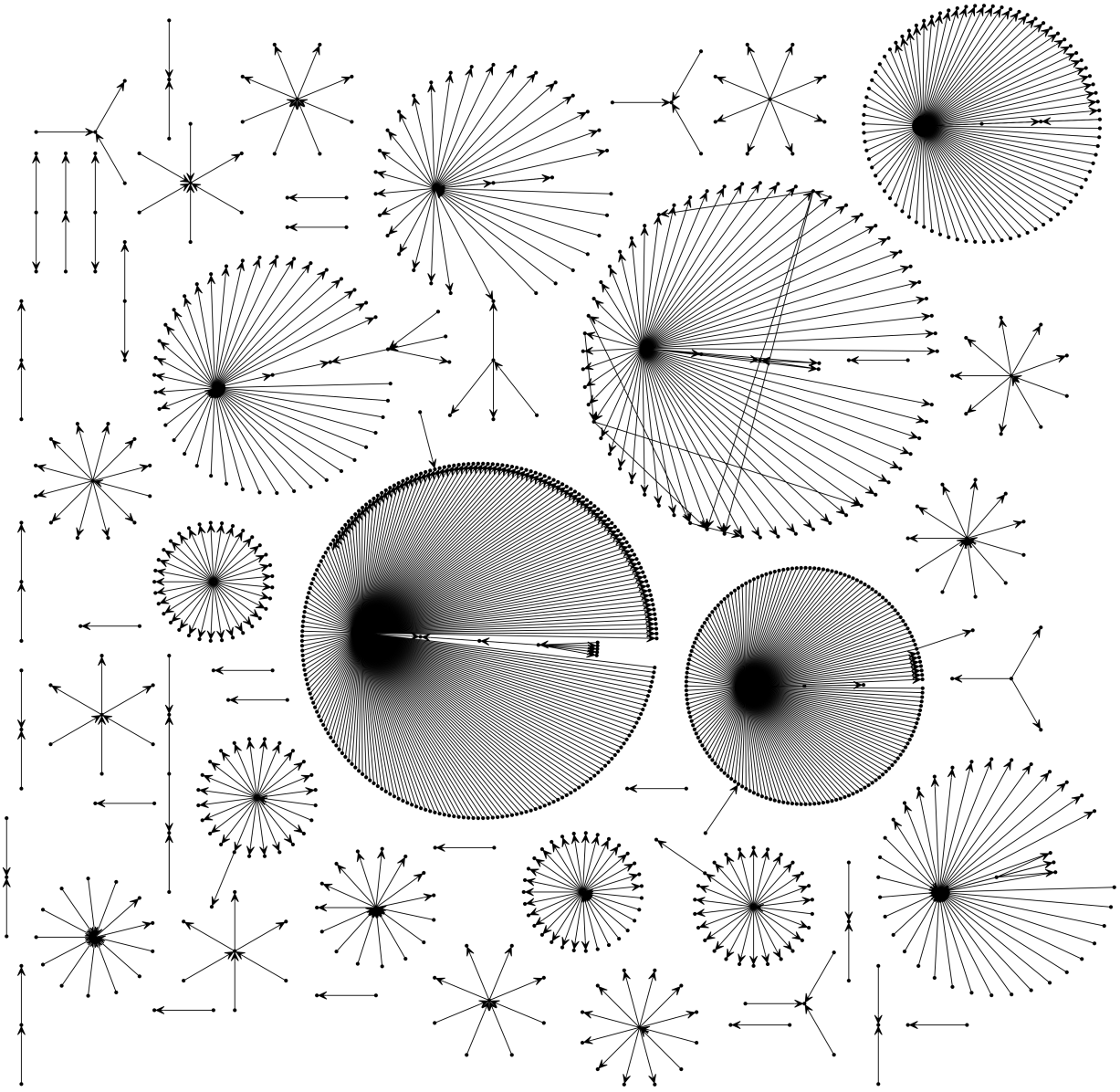


Fig. 16. The network of the relation *RDFS:subPropertyOf* in the DBpedia ontology (depicted without labels for visibility)

References

- [1] T. Berners-Lee, J. Hendler and O. Lassila, The Semantic Web, *Scientific American* **284**(5) (2001), 34–43. doi:<https://doi.org/10.1038/scientificamerican0501-34>. <http://www.nature.com/scientificamerican/journal/v284/n5/pdf/scientificamerican0501-34.pdf>.
- [2] L.G. Valiant, Knowledge Infusion: In Pursuit of Robustness in Artificial Intelligence, in: *IARCS Annu. Conf. Foundations Software Technol. Theoretical Computer Science*, Vol. 2, R. Hariharan, M. Mukund and V. Vinay, eds, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2008, pp. 415–422. doi:10.4230/LIPIcs.FSTTCS.2008.1770. <http://drops.dagstuhl.de/opus/volltexte/2008/1770>.
- [3] A.S. d’Avila Garcez, T.R. Besold, L.D. Raedt, P. Földiák, P. Hitzler, T. Icard, K. Kühnberger, L.C. Lamb, R. Miikkulainen and D.L. Silver, Neural-Symbolic Learning and Reasoning: Contributions and Challenges, in: *2015 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 22-25, 2015*, AAAI Press, 2015. <http://www.aaai.org/ocs/index.php/SSS/SSS15/paper/view/10281>.
- [4] B. Shi and T. Weninger, Open-World Knowledge Graph Completion, in: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S.A. McIlraith and K.Q. Weinberger, eds, AAAI Press, 2018, pp. 1957–1964. <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16055>.
- [5] H. Cai, V.W. Zheng and K.C. Chang, A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications, Vol. abs/1709.07604, 2017. <http://arxiv.org/abs/1709.07604>.
- [6] A. Bordes, J. Weston, R. Collobert and Y. Bengio, Learning Structured Embeddings of Knowledge Bases, in: *Proc. Twenty-Fifth AAAI Conf. Artificial Intelligence*, W. Burgard and D. Roth, eds, AAAI Press, 2011, pp. 301–306. <http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3659>.
- [7] P.J. Hayes and P.F. Patel-Schneider, RDF 1.1 Semantics, W3C Recommendation 25 February 2014, W3C, 2014. <https://www.w3.org/TR/rdf11-mt/>.
- [8] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak and Z. Ives, DBpedia: A Nucleus for a Web of Open Data, in: *Semantic Web*, K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber and P. Cudré-Mauroux, eds, Springer Berlin Heidelberg, Berlin, Germany, 2007, pp. 722–735.
- [9] J.J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne and K. Wilkinson, Jena: Implementing the Semantic Web Recommendations, in: *Proc. 13th Int. World Wide Web Conf. Alternate Track Papers Posters*, ACM, New York, NY, USA, 2004, pp. 74–83. doi:10.1145/1013367.1013381.
- [10] A. Hogan, A. Harth, A. Passant, S. Decker and A. Polleres, Weaving the Pedantic Web, in: *Proc. Linked Data Web Workshop (LDOW2010)*, Vol. 628, C. Bizer, T. Heath, T. Berners-Lee and M. Hausenblas, eds, CEUR-WS.org, Raleigh, NC, USA, 2010, pp. 1–10. http://ceur-ws.org/Vol-628/ldow2010_paper04.pdf.
- [11] A. Zaveri, D. Kontokostas, M.A. Sherif, L. Bühmann, M. Morsey, S. Auer and J. Lehmann, User-driven Quality Evaluation of DBpedia, in: *Proc. 9th Int. Conf. Semantic Systems*, ACM, New York, NY, USA, 2013, pp. 97–104. doi:10.1145/2506182.2506195.
- [12] Y. Guo, Z. Pan and J. Heflin, LUBM: A benchmark for OWL knowledge base systems, *Web Semantics: Science, Services Agents World Wide Web* **3**(2–3) (2005), 158–182. doi:10.1016/j.websem.2005.06.005.
- [13] M. Zhu, Z. Gao and Z. Quan, Noisy Type Assertion Detection in Semantic Datasets, in: *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, P. Mika, T. Tudorache, A. Bernstein, C. Welty, C.A. Knoblock, D. Vrandečić, P.T. Groth, N.F. Noy, K. Janowicz and C.A. Goble, eds, Lecture Notes in Computer Science, Vol. 8796, Springer, 2014, pp. 373–388. doi:10.1007/978-3-319-11964-9_24.
- [14] C. Bizer, T. Heath and T. Berners-Lee, Linked Data - The Story So Far, *Int. J. Semantic Web Inform. Syst.* **5**(3) (2009), 1–22. doi:10.4018/jswis.2009081901.
- [15] M.L. Richard Cyganiak David Wood, RDF 1.1 Concepts and Abstract Syntax, W3C, W3C, 2014. <https://www.w3.org/TR/rdf11-concepts/>.
- [16] J. Hayes and C. Gutiérrez, Bipartite Graphs as Intermediate Model for RDF, in: *Semantic Web - ISWC 2004: Third Int. Semantic Web Conference*, Vol. 3298, S.A. McIlraith, D. Plexousakis and F. van Harmelen, eds, Springer, Berlin, Germany, 2004, pp. 47–61. doi:10.1007/978-3-540-30475-3_5.
- [17] A.A.M. Morales, A Directed Hypergraph Model for RDF, in: *Proc. KWEPSY 2007 Knowledge Web PhD Symp. 2007*, Vol. 275, E.P.B. Simperl, J. Diederich and G. Schreiber, eds, CEUR-WS.org, Innsbruck, Austria, 2007, pp. 1–2. <http://ceur-ws.org/Vol-275/paper24.pdf>.
- [18] H. Liu, D. Dou, R. Jin, P. LePendu and N. Shah, Mining Biomedical Ontologies and Data Using RDF Hypergraphs, in: *12th Int. Conf. Mach. Learning Applications, ICMLA 2013*, IEEE, Miami, FL, USA, 2013, pp. 141–146. doi:10.1109/ICMLA.2013.31.
- [19] G. Wu, J. Li, J. Hu and K. Wang, System π : A native RDF repository based on the hypergraph representation for RDF data model, *J. Computer Sci. Technology* **24**(4) (2009), 652–664. doi:10.1007/s11390-009-9265-9.
- [20] V. Chernenkiy, Y. Gapanyuk, A. Nardid, M. Skvortsova, A. Gushcha, Y. Fedorenko and R. Picking, Using the meta-graph approach for addressing RDF knowledge representation limitations, in: *2017 Internet Technologies Appl. (ITA)*, 2017, pp. 47–52. doi:10.1109/ITECHA.2017.8101909.
- [21] DBpedia, About: James Hendler, 2016. http://dbpedia.org/page/James_Hendler.
- [22] DBpedia, About: Yoshua Bengio, 2016. http://dbpedia.org/page/Yoshua_Bengio.
- [23] M. Ou, P. Cui, J. Pei, Z. Zhang and W. Zhu, Asymmetric Transitivity Preserving Graph Embedding, in: *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery Data Mining*, ACM, New York, NY, USA, 2016, pp. 1105–1114. doi:10.1145/2939672.2939751.
- [24] K. Cho, B. van Merriënboer, D. Bahdanau and Y. Bengio, On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, in: *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure*

- in *Statistical Translation*, Doha, Qatar, 25 October 2014, D. Wu, M. Carpuat, X. Carreras and E.M. Vecchi, eds, Association for Computational Linguistics, 2014, pp. 103–111. ISBN 978-1-937284-96-1. <http://aclweb.org/anthology/W/W14/W14-4012.pdf>.
- [25] J. Gehring, M. Auli, D. Grangier, D. Yarats and Y.N. Dauphin, Convolutional Sequence to Sequence Learning, in: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, D. Precup and Y.W. Teh, eds, Proceedings of Machine Learning Research, Vol. 70, PMLR, 2017, pp. 1243–1252. <http://proceedings.mlr.press/v70/gehring17a.html>.
- [26] F. Chollet et al., Keras, GitHub, 2015. <https://github.com/keras-team/keras>.
- [27] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu and X. Zheng, TensorFlow: A System for Large-Scale Machine Learning, in: *12th USENIX Symp. Operating Syst. Design Implementation (OSDI 16)*, USENIX Association, Savannah, GA, USA, 2016, pp. 265–283. <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>.
- [28] I. Sutskever, O. Vinyals and Q.V. Le, Sequence to Sequence Learning with Neural Networks, in: *Proc. 27th Int. Conf. Neural Inform. Process. Syst. - Volume 2*, MIT Press, Cambridge, MA, USA, 2014, pp. 3104–3112. <http://dl.acm.org/citation.cfm?id=2969033.2969173>.
- [29] M. Schuster and K.K. Paliwal, Bidirectional recurrent neural networks, *IEEE Trans. Signal Process* **45**(11) (1997), 2673–2681. doi:10.1109/78.650093.
- [30] J. Chung, Ç. Gülçehre, K. Cho and Y. Bengio, Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, Vol. abs/1412.3555, 2014. <http://arxiv.org/abs/1412.3555>.
- [31] D.P. Kingma and J. Ba, Adam: A Method for Stochastic Optimization, in: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, eds, 2015. <http://arxiv.org/abs/1412.6980>.
- [32] Z. Wang, J. Zhang, J. Feng and Z. Chen, Knowledge Graph Embedding by Translating on Hyperplanes, in: *Proc. Twenty-Eighth AAAI Conf. Artificial Intelligence*, C.E. Brodley and P. Stone, eds, AAAI Press, Cambridge, MA, USA, 2014, pp. 1112–1119. <http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8531>.
- [33] T. Trouillon, C.R. Dance, É. Gaussier, J. Welbl, S. Riedel and G. Bouchard, Knowledge graph completion via complex tensor factorization, *Journal of Machine Learning Research (JMLR)* **18**(130) (2017), 1–38.
- [34] M. Nickel, L. Rosasco and T. Poggio, Holographic Embeddings of Knowledge Graphs, in: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI'16*, AAAI Press, 2016, pp. 1955–1961. <http://dl.acm.org/citation.cfm?id=3016100.3016172>.
- [35] R.X.Z.L.M.S. Xu Han Yankai Lin, OpenKE, THUNLP, 2017. <http://openke.thunlp.org/home>.
- [36] B. Motik, B.C. Grau, I. Horrocks, Z. Wu, A. Fokoue and C. Lutz, OWL 2 Web Ontology Language Profiles, W3C Recommendation 27 October 2009, W3C, 2012. <https://www.w3.org/TR/owl2-profiles/>.
- [37] H. Paulheim and C. Bizer, Improving the Quality of Linked Data Using Statistical Distributions, *Int. J. Semantic Web Inform. Syst.* **10**(2) (2014), 63–86. doi:10.4018/ijswis.2014040104.
- [38] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E.R.H. Jr. and T.M. Mitchell, Toward an Architecture for Never-Ending Language Learning, in: *Proc. Twenty-Fourth AAAI Conf. Artificial Intelligence, AAAI, M. Fox and D. Poole, eds*, AAAI Press, Cambridge, MA, USA, 2010, pp. 1306–1313. <http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1879>.
- [39] D. Fleischhacker, H. Paulheim, V. Bryl, J. Völker and C. Bizer, Detecting Errors in Numerical Linked Data Using Cross-Checked Outlier Detection, in: *Semantic Web – ISWC 2014*, P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. Knoblock, D. Vrandečić, P. Groth, N. Noy, K. Janowicz and C. Goble, eds, Springer International Publishing, New York, NY, USA, 2014, pp. 357–372. doi:10.1007/978-3-319-11964-9_23.
- [40] D. Wienand and H. Paulheim, Detecting Incorrect Numerical Data in DBpedia, in: *Semantic Web: Trends Challenges*, V. Presutti, C. d'Amato, F. Gandon, M. d'Aquin, S. Staab and A. Tordai, eds, Springer International Publishing, New York, NY, USA, 2014, pp. 504–518. doi:10.1007/978-3-319-07443-6_34.
- [41] E. Parzen, On Estimation of a Probability Density Function and Mode, *Ann. Math. Statistics* **33**(3) (1962), 1065–1076. <http://www.jstor.org/stable/2237880>.
- [42] H. Paulheim and C. Bizer, Type Inference on Noisy RDF Data, in: *Semantic Web – ISWC 2013*, H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J.X. Parreira, L. Aroyo, N. Noy, C. Welty and K. Janowicz, eds, Springer Berlin Heidelberg, Berlin, Germany, 2013, pp. 510–525.
- [43] P. Goyal and E. Ferrara, Graph Embedding Techniques, Applications, and Performance: A Survey, Vol. abs/1705.02801, 2017. <http://arxiv.org/abs/1705.02801>.
- [44] Q. Wang, Z. Mao, B. Wang and L. Guo, Knowledge Graph Embedding: A Survey of Approaches and Applications, *IEEE Trans. Knowl. Data Eng* **29**(12) (2017), 2724–2743. doi:10.1109/TKDE.2017.2754499.
- [45] S. Bhagat, G. Cormode and S. Muthukrishnan, Node Classification in Social Networks, Vol. abs/1101.3291, 2011. <http://arxiv.org/abs/1101.3291>.
- [46] D. Liben-Nowell and J. Kleinberg, The Link-prediction Problem for Social Networks, *J. Assoc. Inform. Sci. Technology* **58**(7) (2007), 1019–1031. doi:10.1002/asi.v58:7.
- [47] M. Belkin and P. Niyogi, Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering, in: *Advances in Neural Inform. Process. Syst. 14*, T.G. Dietterich, S. Becker and Z. Ghahramani, eds, MIT Press, Cambridge, MA, USA, 2002, pp. 585–591. <http://papers.nips.cc/paper/1961-laplacian-eigenmaps-and-spectral-techniques-for-embedding-and-clustering.pdf>.
- [48] L. Katz, A new status index derived from sociometric analysis, *Psychometrika* **18**(1) (1953), 39–43. doi:10.1007/BF02289026.
- [49] S.T. Roweis and L.K. Saul, Nonlinear Dimensionality Reduction by Locally Linear Embedding, *Science* **290**(5500) (2000), 2323–2326. doi:10.1126/science.290.5500.2323. <http://science.sciencemag.org/content/290/5500/2323>.

- [50] A. Ahmed, N. Shervashidze, S.M. Narayanamurthy, V. Josifovski and A.J. Smola, Distributed large-scale natural graph factorization, in: *Proc. 22Nd Int. Conf. World Wide Web*, D. Schwabe, V.A.F. Almeida, H. Glaser, R.A. Baeza-Yates and S.B. Moon, eds, ACM, New York, NY, USA, 2013, pp. 37–48. doi:10.1145/2488388.2488393.
- [51] M. Ou, P. Cui, J. Pei, Z. Zhang and W. Zhu, Asymmetric Transitivity Preserving Graph Embedding, in: *Proc. 22Nd ACM SIGKDD Int. Conf. Knowledge Discovery Data Mining*, B. Krishnapuram, M. Shah, A.J. Smola, C.C. Aggarwal, D. Shen and R. Rastogi, eds, ACM, New York, NY, USA, 2016, pp. 1105–1114. doi:10.1145/2939672.2939751.
- [52] L. Lovász, Random walks on graphs: A survey, *Combinatorics, Paul Erdos Is Eighty* 2 (1993), 1–46. <http://web.cs.elte.hu/~lovasz/erdos.pdf>.
- [53] A. Grover and J. Leskovec, node2vec: Scalable Feature Learning for Networks, in: *Proc. 22Nd ACM SIGKDD Int. Conf. Knowledge Discovery Data Mining*, B. Krishnapuram, M. Shah, A.J. Smola, C.C. Aggarwal, D. Shen and R. Rastogi, eds, ACM, New York, NY, USA, 2016, pp. 855–864. doi:10.1145/2939672.2939754.
- [54] T. Mikolov, K. Chen, G. Corrado and J. Dean, Efficient Estimation of Word Representations in Vector Space, Vol. abs/1301.3781, 2013. <http://arxiv.org/abs/1301.3781>.
- [55] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner and G. Monfardini, The Graph Neural Network Model, *IEEE Trans. Neural Netw.* 20(1) (2009), 61–80. doi:10.1109/TNN.2008.2005605.
- [56] T.N. Kipf and M. Welling, Semi-Supervised Classification with Graph Convolutional Networks, *CoRR* abs/1609.02907 (2016). <http://arxiv.org/abs/1609.02907>.
- [57] T.N. Kipf and M. Welling, Variational Graph Auto-Encoders, Vol. abs/1611.07308, 2016. <http://arxiv.org/abs/1611.07308>.
- [58] A. Bordes, N. Usunier, A. García-Durán, J. Weston and O. Yakhnenko, Translating Embeddings for Modeling Multi-relational Data, in: *Advances in Neural Inform. Process. Syst.* 26, C.J.C. Burges, L. Bottou, Z. Ghahramani and K.Q. Weinberger, eds, Curran Associates, Inc., Red Hook, NY, USA, 2013, pp. 2787–2795. <http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf>.
- [59] Y. Lin, Z. Liu, M. Sun, Y. Liu and X. Zhu, Learning Entity and Relation Embeddings for Knowledge Graph Completion, in: *Proc. Twenty-Ninth AAAI Conf. Artificial Intelligence*, B. Bonet and S. Koenig, eds, AAAI Press, Cambridge, MA, USA, 2015, pp. 2181–2187. <http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9571>.
- [60] G. Ji, S. He, L. Xu, K. Liu and J. Zhao, Knowledge Graph Embedding via Dynamic Mapping Matrix, in: *Proc. 53rd Annu. Meeting Assoc. Computational Linguistics 7th Int. Joint Conf. Natural Language Process. (Volume 1: Long Papers)*, The Association for Computer Linguistics, Beijing, China, 2015, pp. 687–696. <http://aclweb.org/anthology/P/P15/P15-1067.pdf>.
- [61] S. He, K. Liu, G. Ji and J. Zhao, Learning to Represent Knowledge Graphs with Gaussian Embedding, in: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, ACM, New York, NY, USA, 2015, pp. 623–632. ISBN 978-1-4503-3794-6. doi:10.1145/2806416.2806502.
- [62] R. Jenatton, N.L. Roux, A. Bordes and G. Obozinski, A latent factor model for highly multi-relational data, in: *Advances in Neural Inform. Process. Syst.* 25, P.L. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou and K.Q. Weinberger, eds, Curran Associates, Inc., Red Hook, NY, USA, 2012, pp. 3167–3175. <http://papers.nips.cc/paper/4744-a-latent-factor-model-for-highly-multi-relational-data.pdf>.
- [63] M. Nickel, V. Tresp and H.-P. Kriegel, A Three-way Model for Collective Learning on Multi-relational Data, in: *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, Omnipress, USA, 2011, pp. 809–816. ISBN 978-1-4503-0619-5. <http://dl.acm.org/citation.cfm?id=3104482.3104584>.
- [64] B. Yang, W. Yih, X. He, J. Gao and L. Deng, Embedding Entities and Relations for Learning and Inference in Knowledge Bases, *CoRR* abs/1412.6575 (2014). <http://arxiv.org/abs/1412.6575>.
- [65] T. Trouillon, J. Welbl, S. Riedel, E. Gaussier and G. Bouchard, Complex Embeddings for Simple Link Prediction, in: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, JMLR.org, 2016, pp. 2071–2080. <http://dl.acm.org/citation.cfm?id=3045390.3045609>.
- [66] A. Bordes, X. Glorot, J. Weston and Y. Bengio, A semantic matching energy function for learning with multi-relational data - Application to word-sense disambiguation, *Machine Learning* 94(2) (2014), 233–259. doi:10.1007/s10994-013-5363-6.
- [67] R. Socher, D. Chen, C.D. Manning and A.Y. Ng, Reasoning With Neural Tensor Networks For Knowledge Base Completion, in: *Advances in Neural Information Processing Systems* 26, 2013.
- [68] M. Schlichtkrull, T.N. Kipf, P. Bloem, R. van den Berg, I. Titov and M. Welling, Modeling Relational Data with Graph Convolutional Networks, in: *The Semantic Web*, A. Gangemi, R. Navigli, M.-E. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai and M. Alam, eds, Springer International Publishing, Cham, 2018, pp. 593–607. ISBN 978-3-319-93417-4.
- [69] U. Lösch, S. Bloehdorn and A. Rettinger, Graph Kernels for RDF Data, in: *Semantic Web: Res. Applications*, E. Simperl, P. Cimiano, A. Polleres, Ó. Corcho and V. Presutti, eds, Springer Berlin Heidelberg, Berlin, Germany, 2012, pp. 134–148. doi:10.1007/978-3-642-30284-8_16.
- [70] G.K.D. de Vries, A Fast Approximation of the Weisfeiler-Lehman Graph Kernel for RDF Data, in: *Mach. Learning Knowledge Discovery in Databases*, H. Blockeel, K. Kersting, S. Nijssen and F. Zelezny, eds, Springer Berlin Heidelberg, Berlin, Germany, 2013, pp. 606–621. doi:10.1007/978-3-642-40988-2_39.
- [71] N. Shervashidze, P. Schweitzer, E.J. van Leeuwen, K. Mehlhorn and K.M. Borgwardt, Weisfeiler-Lehman Graph Kernels, *J. Mach. Learning Research* 12 (2011), 2539–2561. <http://dl.acm.org/citation.cfm?id=1953048.2078187>.
- [72] V.C. Ostuni, T.D. Noia, R. Mirizzi and E.D. Sciascio, A Linked Data Recommender System Using a Neighborhood-Based Graph Kernel, in: *E-Commerce Web Technologies*, M. Hepp and Y. Hoffner, eds, Springer International Publishing, New York, NY, USA, 2014, pp. 89–100. doi:10.1007/978-3-319-10491-1_10.

- [73] T. Mikolov, I. Sutskever, K. Chen, G. Corrado and J. Dean, Distributed Representations of Words and Phrases and Their Compositionality, in: *Proc. 26th Int. Conf. Neural Inform. Process. Syst. - Volume 2*, Curran Associates Inc., Red Hook, NY, USA, 2013, pp. 3111–3119. <http://dl.acm.org/citation.cfm?id=2999792.2999959>.
- [74] P. Ristoski and H. Paulheim, RDF2Vec: RDF Graph Embeddings for Data Mining, in: *Semantic Web – ISWC 2016*, Vol. 9981, P.T. Groth, E. Simperl, A.J.G. Gray, M. Sabou, M. Krötzsch, F. Lécué, F. Flöck and Y. Gil, eds, Springer International Publishing, New York, NY, USA, 2016, pp. 498–514. doi:10.1007/978-3-319-46523-4_30.
- [75] M. Cochez, P. Ristoski, S.P. Ponzetto and H. Paulheim, Biased graph walks for RDF graph embeddings, in: *Proc. 7th Int. Conf. Web Intelligence, Mining Semantics*, R. Akerkar, A. Cuzzocrea, J. Cao and M. Hacid, eds, ACM, New York, NY, USA, 2017, pp. 21:1–21:12. doi:10.1145/3102254.3102279.
- [76] M. Cochez, P. Ristoski, S.P. Ponzetto and H. Paulheim, Global RDF Vector Space Embeddings, in: *Semantic Web - ISWC 2017 - 16th Int. Semantic Web Conference, Part I*, Vol. 10587, C. d’Amato, M. Fernández, V.A.M. Tamma, F. Lécué, P. Cudré-Mauroux, J.F. Sequeda, C. Lange and J. Heflin, eds, Springer, New York, NY, USA, 2017, pp. 190–207. doi:10.1007/978-3-319-68288-4_12.
- [77] J. Pennington, R. Socher and C.D. Manning, Glove: Global Vectors for Word Representation, in: *Proc. 2014 Conf. Empirical Methods in Natural Language Process. (EMNLP)*, A. Moschitti, B. Pang and W. Daelemans, eds, Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1532–1543. <http://www.aclweb.org/anthology/D14-1162>.
- [78] R. Socher, D. Chen, C.D. Manning and A.Y. Ng, Reasoning With Neural Tensor Networks for Knowledge Base Completion, in: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, C.J.C. Burges, L. Bottou, Z. Ghahramani and K.Q. Weinberger, eds, 2013, pp. 926–934. <http://papers.nips.cc/paper/5028-reasoning-with-neural-tensor-networks-for-knowledge-base-completion>.
- [79] Y. Lin, Z. Liu, M. Sun, Y. Liu and X. Zhu, Learning Entity and Relation Embeddings for Knowledge Graph Completion, in: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, B. Bonet and S. Koenig, eds, AAAI Press, 2015, pp. 2181–2187. ISBN 978-1-57735-698-1. <http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9571>.
- [80] G. Ji, K. Liu, S. He and J. Zhao, Knowledge Graph Completion with Adaptive Sparse Transfer Matrix, in: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.*, D. Schuurmans and M.P. Wellman, eds, AAAI Press, 2016, pp. 985–991. ISBN 978-1-57735-760-5. <http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11982>.
- [81] L. Serafini and A.S. d’Avila Garcez, Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge, in: *Proceedings of the 11th International Workshop on Neural-Symbolic Learning and Reasoning (NeSy’16) co-located with the Joint Multi-Conference on Human-Level Artificial Intelligence (HLAI 2016), New York City, NY, USA, July 16-17, 2016.*, T.R. Besold, L.C. Lamb, L. Serafini and W. Tabor, eds, CEUR Workshop Proceedings, Vol. 1768, CEUR-WS.org, 2016. http://ceur-ws.org/Vol-1768/NESY16_paper3.pdf.
- [82] T. Rocktäschel and S. Riedel, End-to-end Differentiable Proving, in: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, I. Guyon, U. von Luxburg, S. Bengio, H.M. Wallach, R. Fergus, S.V.N. Vishwanathan and R. Garnett, eds, 2017, pp. 3791–3803. <http://papers.nips.cc/paper/6969-end-to-end-differentiable-proving>.
- [83] P. Hitzler and F. van Harmelen, A reasonable Semantic Web, *Semantic Web* 1(1–2) (2010), 39–44. doi:10.3233/SW-2010-0010.
- [84] A.G. Nuzzolese, A. Gangemi, V. Presutti and P. Ciancarini, Type inference through the analysis of Wikipedia links, in: *WWW2012 Workshop Linked Data Web*, Vol. 937, C. Bizer, T. Heath, T. Berners-Lee and M. Hausenblas, eds, CEUR-WS.org, Lyon, France, 2012, pp. 1–9. <http://ceur-ws.org/Vol-937/ldow2012-paper-13.pdf>.
- [85] A.G. Nuzzolese, A. Gangemi, V. Presutti and P. Ciancarini, Encyclopedic Knowledge Patterns from Wikipedia Links, in: *Semantic Web – ISWC 2011*, Vol. 7031, L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N.F. Noy and E. Blomqvist, eds, Springer Berlin Heidelberg, Berlin, Germany, 2011, pp. 520–536. doi:10.1007/978-3-642-25073-6_33.
- [86] H. Paulheim and A. Gangemi, Serving DBpedia with DOLCE - More than Just Adding a Cherry on Top, in: *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*, M. Arenas, Ó. Corcho, E. Simperl, M. Strohmaier, M. d’Aquin, K. Srinivas, P.T. Groth, M. Dumontier, J. Heflin, K. Thirunarayan and S. Staab, eds, Lecture Notes in Computer Science, Vol. 9366, Springer, 2015, pp. 180–196. ISBN 978-3-319-25006-9. doi:10.1007/978-3-319-25007-6_11.
- [87] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari and L. Schneider, Sweetening Ontologies with DOLCE, in: *Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, 13th International Conference, EKAW 2002, Sigüenza, Spain, October 1-4, 2002, Proceedings*, A. Gómez-Pérez and V.R. Benjamins, eds, Lecture Notes in Computer Science, Vol. 2473, Springer, 2002, pp. 166–181. ISBN 3-540-44268-5. doi:10.1007/3-540-45810-7_18.
- [88] H. Paulheim and H. Stuckenschmidt, Fast Approximate A-Box Consistency Checking Using Machine Learning, in: *Semantic Web. Latest Advances New Domains*, H. Sack, E. Blomqvist, M. d’Aquin, C. Ghidini, S.P. Ponzetto and C. Lange, eds, Springer International Publishing, New York, NY, USA, 2016, pp. 135–150.
- [89] U. Lösch, S. Bloehdorn and A. Rettinger, Graph Kernels for RDF Data, in: *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*, E. Simperl, P. Cimiano, A. Polleres, Ó. Corcho and V. Presutti, eds, Lecture Notes in Computer Science, Vol. 7295, Springer, 2012, pp. 134–148. ISBN 978-3-642-30283-1. doi:10.1007/978-3-642-30284-8_16.

- [90] C. Meilicke, D. Ruffinelli, A. Nolle, H. Paulheim and H. Stuckenschmidt, Fast ABox Consistency Checking Using Incomplete Reasoning and Caching, in: *Rules and Reasoning - International Joint Conference, RuleML+RR 2017, London, UK, July 12-15, 2017, Proceedings*, S. Costantini, E. Franconi, W.V. Woensel, R. Kontchakov, F. Sadri and D. Roman, eds, Lecture Notes in Computer Science, Vol. 10364, Springer, 2017, pp. 168–183. ISBN 978-3-319-61251-5. doi:10.1007/978-3-319-61252-2_12.
- [91] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi and D.F. Savo, Query Rewriting for Inconsistent DL-Lite Ontologies, in: *Web Reasoning and Rule Systems - 5th International Conference, RR 2011, Galway, Ireland, August 29-30, 2011. Proceedings*, S. Rudolph and C. Gutiérrez, eds, Lecture Notes in Computer Science, Vol. 6902, Springer, 2011, pp. 155–169. ISBN 978-3-642-23579-5. doi:10.1007/978-3-642-23580-1_12.
- [92] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini and R. Rosati, Tractable Reasoning and Efficient Query Answering in Description Logics: The *DL-Lite* Family, *J. Autom. Reasoning* **39**(3) (2007), 385–429. doi:10.1007/s10817-007-9078-x.
- [93] P. Hohenecker and T. Lukasiewicz, Deep Learning for Ontology Reasoning, Vol. abs/1705.10342, 2017. <http://arxiv.org/abs/1705.10342>.
- [94] R. Socher, A. Perelygin, J. Wu, J. Chuang, C.D. Manning, A. Ng and C. Potts, Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank, in: *Proc. 2013 Conf. Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Seattle, WA, USA, 2013, pp. 1631–1642. <http://www.aclweb.org/anthology/D13-1170>.
- [95] B. Makni and J. Hendler, Deep Learning of RDFS rules, 2016, IJCAI Workshop “Semantic Machine Learning”. http://ist.gmu.edu/~hpurohit/events/sml16/pdf/SML2016_submission6_Bassem_revised.pdf.
- [96] I.J. Goodfellow, J. Shlens and C. Szegedy, Explaining and Harnessing Adversarial Examples, Vol. abs/1412.6572, 2014. <http://arxiv.org/abs/1412.6572>.
- [97] B. Smith, M. Ashburner, C. Rosse, J. Bard, W. Bug, W. Ceusters, L.J. Goldberg, K. Eilbeck, A. Ireland, C.J. Mungall, T.O. Consortium, N. Leontis, P. Rocca-Serra, A. Ruttenberg, S.-A. Sansone, R.H. Scheuermann, N. Shah, P.L. Whetzel and S. Lewis, The OBO Foundry: Coordinated evolution of ontologies to support biomedical data integration, *Nature Biotechnology* **25** (2007), 1251. <http://dx.doi.org/10.1038/nbt1346>.
- [98] W.A. Kibbe, C. Arze, V. Felix, E. Mitraka, E. Bolton, G. Fu, C.J. Mungall, J.X. Binder, J. Malone, D. Vasant, H. Parkinson and L.M. Schriml, Disease Ontology 2015 update: an expanded and updated database of human diseases for linking biomedical knowledge through disease data, *Nucleic Acids Research* **43**(D1) (2015), 1071–1078. doi:10.1093/nar/gku1011.
- [99] S.J. Pan and Q. Yang, A Survey on Transfer Learning, *IEEE Trans. Knowl. Data Eng* **22**(10) (2010), 1345–1359. doi:10.1109/TKDE.2009.191.
- [100] D. Brickley, Semantic Web: Learning from Machine Learning, in: *Joint Proc. Int. Workshops Hybrid Statistical Semantic Understanding Emerging Semantics, Semantic Statistics co-located with 16th Int. Semantic Web Conference, Hybrid-SemStats*, S. Capadisli, F. Cotton, X.L. Dong, R.V. Guha, A. Haller, P. Hitzler, E. Kalampokis, M. Kejriwal, F. Lécué, D. Sivakumar, P. Szekely, R. Troncy and M.J. Witbrock, eds, CEUR-WS.org, Vienna, Austria, 2017, pp. 1–6. <http://ceur-ws.org/Vol-1923/article-08.pdf>.
- [101] M.K. Sarker, N. Xie, D. Doran, M. Raymer and P. Hitzler, Explaining Trained Neural Networks with Semantic Web Technologies: First Steps, in: *Proc. Twelfth Int. Workshop Neural-Symbolic Learning Reasoning*, Vol. 2003, T.R. Besold, A.S. d’Avila Garcez and I. Noble, eds, CEUR-WS.org, London, UK, 2017, pp. 1–10. http://ceur-ws.org/Vol-2003/NeSy17_paper4.pdf.
- [102] C. Gutiérrez, C.A. Hurtado and A.O. Mendelzon, Formal aspects of querying RDF databases, in: *Proc. First Int. Conf. Semantic Web Databases*, I.F. Cruz, V. Kश्यap, S. Decker and R. Eckstein, eds, CEUR-WS.org, Aachen, Germany, 2003, pp. 279–293. <http://dl.acm.org/citation.cfm?id=2889905.2889924>.
- [103] A. Hogan, M. Arenas, A. Mallea and A. Polleres, Everything you always wanted to know about blank nodes, *Web Semantics: Science, Services Agents World Wide Web* **27** (2014), 42–69. doi:10.1016/j.websem.2014.06.004.
- [104] A.A. Hagberg, D.A. Schult and P.J. Swart, Exploring Network Structure, Dynamics, and Function using NetworkX, in: *Proc. 7th Python in Sci. Conference*, G. Varoquaux, T. Vaught and J. Millman, eds, Pasadena, CA, USA, 2008, pp. 11–15.