

Deep learning for noise-tolerant RDFS reasoning

Bassem Makni^a, James Hendler^a and

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

E-mails: maknib@rpi.edu, hendler@cs.rpi.edu

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

Solicited reviews: First Solicited Reviewer, University or Company name, Country; Second Solicited Reviewer, University or Company name, Country

Open reviews: First Open Reviewer, University or Company name, Country; Second Open Reviewer, University or Company name, Country

Abstract. Since the introduction of the Semantic Web vision in 2001 as an extension to the Web, the main research focus in semantic reasoning was on the soundness and completeness of the reasoners. While these reasoners assume the veracity of the input data, the reality is that the Web of data is inherently noisy. Recent research work on semantic reasoning with noise-tolerance focuses on type inference and does not aim for full RDFS reasoning. This paper documents a novel approach that takes previous research efforts in noise-tolerance in the Semantic Web to the next level of full RDFS reasoning by utilizing advances in deep learning research. This is a stepping stone towards bridging the *Neural-Symbolic gap* for RDFS reasoning which is accomplished through layering RDF graphs and encoding them in the form of 3D adjacency matrices where each layer layout forms 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 and real-world Semantic Web data while showing noise-tolerance capabilities as opposed to rule-based reasoners.

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

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, the noise can be as 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) tools—whose accuracy is not perfect—they generate erroneous triples. Thus, the integrity of the inference becomes questionable.

1.1. A Realistic approach for an idealistic vision

It is foolish to expect that the Web or the Semantic Web 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.

This paper documents a novel approach that takes previous research efforts in noise-tolerance to the next level of full RDF Schema (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.

1.2. Neural-Symbolic Gap

Humans are able to learn from very few examples while providing explanations for their decision making 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 [1]. “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.” ([1]) However, connectionist learning uses low-level representations- such as embeddings- rather than “symbolic representations used in knowledge representation” ([2]). 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 Semantic Web field and RDFS reasoning in particular.

1.3. Hypothesis and outline

The research hypothesis are:

1. RDFS rules are learnable by connectionist models
2. A deep reasoner for RDFS will be noise-tolerant

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. Then, the creation process is described for two noisy datasets— a synthetic dataset and a real world dataset—that are used as models to describe the design of the overall approach. The creation of the graph words as well as the description of the graph words translation are de-

scribed respectively in ?? and Section 6. The results of the experiments are described in the Evaluation chapter. Finally the learned lessons, main contributions and future work are illustrated.

2. State of the Art

2.1. Handling Noise in Semantic Web Data

We classify the strategies of handling noise in Semantic Web 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 the following. Giving the unrealistic expectation of cleansing every type of noise in Semantic Web 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.

2.1.1. 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 [3], 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 [4], the authors describe two algorithms that they designed to improve the quality of LOD. *SDType* algorithm falls into the category of adaptive noise handling and will be described in the corresponding section. *SD-Validate* 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 (NELL) [5] knowledge bases.

In [6], the authors focus on detecting noisy type assertions. They built a few synthetic noisy datasets

based on Lehigh University Benchmark (LUBM). Then a multi-class classifier is trained to learn disjoint classes.

In [7, 8], the focus is on incorrect numerical data in LOD datasets. [7] 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. [8] uses a few unsupervised learning techniques including Kernel Density Estimation (KDE) [9] combined with semantic grouping to identify the outliers.

2.1.2. Adaptive Noise Handling

In *SDType* algorithm [4, 10], 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, all the previous work in the literature about reasoning with noisy Semantic Web data focuses on type inference. This research is the first to aim full RDFS reasoning with noise-tolerance capability.

2.2. Deep Learning and Semantic Web

The literature that tries to combine deep learning and Semantic Web can be classified into two categories:

Deep learning for Semantic Web In this category, deep learning methods are used to solve research problems and/or improve current solutions to Semantic Web challenges.

Semantic Web for deep learning In this category, Semantic Web technologies are used to improve deep learning methods by making the classification algorithms more aware of the semantics of the data they are learning from.

2.2.1. Deep Learning for Semantic Web

Deep learning methods are being used mainly to serve two goals in the Semantic Web: *Ontology building* and *Ontological reasoning*.

Deep Learning for Ontology Building Ontologies are the backbone of the Semantic Web. They are most commonly defined using Gruber's definition: "An ontology is an explicit specification of a conceptualization" ([11]). They can be either upper level ontologies that describe high level concepts such as *Agent*, *Process*, *Spacial region* etc. or domain ontologies that describe domain specific concepts and their relations, or hybrid ontologies. Domain ontologies are usually built by capitalizing on the domain experts' knowledge from the domain texts for example. Ontology learning consists of building ontologies automatically or semi-automatically from texts. The literature contains extensive work [12–15] on building or bootstrapping ontologies from texts using Natural Language Processing (NLP) tools. One of the most important steps in the methodologies of building ontologies from texts is the extraction of *candidate terms*. In [16], the authors apply recent advances in word embedding—namely Continuous Bag of Words (CBOW) and Continuous Skip-gram [17]—in order to extract candidate terms from a corpus of PubMed articles' titles and abstracts. They also compare the results with two traditional distributional semantic models namely Latent Semantic Analysis (LSA) [18] and Latent Dirichlet Allocation (LDA) [19]. In [20], the authors apply a similar approach to build an ontology from Arabic texts.

Deep Learning for Ontological Reasoning One of the closest research efforts to the scope of this research is [21]. 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) [22] for relational learning. RNTN 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 [21], 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.

2.2.2. Semantic Web for Deep Learning

Deep learning models are often described as black box models because they are not built in a way that permits backtracking their decision-making process. This constitutes a major drawback, especially in safety-critical systems. Imagine a self-driving car getting into an accident by avoiding a pothole and hitting a tree on the side. Understanding the root cause of the pothole mis-classification is critical to improve the safety awareness of autonomous cars. Demystifying the internals of neural network models is not a recent challenge. In [23], the authors use propositional rules to generate explanations from neural network models. [24] builds on top of these approaches by describing an early proof of concept that uses Suggested Upper Merged Ontology (SUMO) [25] as background knowledge and description logic formalism to provide human readable explanations of neural network decisions.

3. Ground Truthing and Noise Induction

In supervised machine learning, the process of collecting the data—ground truthing—is crucial because the ground truth is used in the training phase to learn a mapping between the inputs and their corresponding targets in order to predict the targets of unseen inputs. 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. The target for these datasets is set using a state of the art Semantic Web reasoner (Jena [26]). Essentially, the goal for the deep reasoner is to learn the mapping between input RDF graphs and their inference graphs in the presence of noise. Thus, noise was induced in the synthetic dataset to test the noise-tolerance of the deep reasoner.

In this section, a taxonomy of noise types in Semantic Web data is outlined, then the process of ground truthing and noise induction in the LUBM dataset is described, and finally the ground truthing of the DBpedia dataset is detailed.

3.1. Taxonomy of Semantic Web Noise Types

The literature contains a few taxonomies for the types of noise that can impact RDF graphs. These taxonomies are drawn with respect to different goals. For example [27] focuses on noise types that can occur when publishing RDF graphs on the Web in the form of LOD—such as accessibility and dereferability. On

the other hand, in [28] the authors consider a particular dataset in the LOD—which is DBpedia—and enumerate 17 different types of noise that can impact this dataset. Furthermore, [6] considers two types of noisy assertions: noisy type assertions and noisy property assertions.

A noisy type assertion is a corruption of a triple in the form:

```
subject RDF:type concept1 .
```

into a triple of the form:

```
subject RDF:type concept2 .
```

given that $\text{concept1} \neq \text{concept2}$. Similarly, a property noise assertion is a corruption of a triple in the form

```
subject property1 object .
```

into a triple of the form:

```
subject property2 object .
```

given that $\text{property1} \neq \text{property2}$.

These types of noises can either impact the inference or have no effect on it. For instance, if concept1 and concept2 in the previous example are two sibling concepts and concept3 being their super-class, both the original triple and the corrupted type assertion will generate—according to RDFS rule *RDFS9*—the same inference which is:

```
subject RDF:type concept3 .
```

Accordingly, a new taxonomy of noise types in Semantic Web data that classifies the impact of the noise on the inference is proposed in this paper. This taxonomy is divided into the following classes: (Figure 1)

3.1.1. TBox Noise

This is the type noise that resides within the ontology, such as in the classes hierarchy or domain and range classes. This type of noise impacts the inference of 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 of type *dbo:Artist* in the inference. To be fair, it can be argued that this is not noisy inference as every scientist is an artist in the sense of creative thinking. Reasoning with tolerance to TBox noise is outside the scope of this research because of the following:

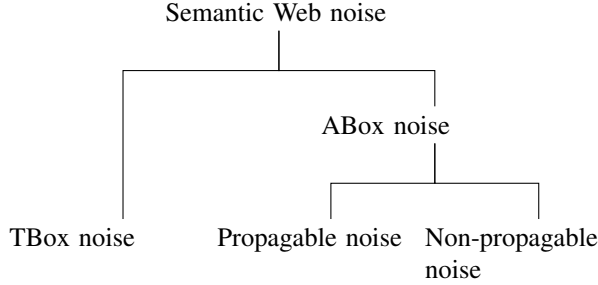


Fig. 1. Semantic Web noise taxonomy

- Rule-based reasoners are used for ground truthing and the goal is to learn the mapping between the input and inference graphs in the ground truth.
- The presence of TBox noise biases the whole ground truth and noisy inferences are not the only anomalies that will be detected and fixed. In the *dbo:field* example, when all the scientists' inference graphs assert that they are artists, a learning algorithm will also learn this mapping between scientists and artists.

As a result of this, 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. In other words, the ontology is not noisy and only assertions can be noisy.

3.1.2. ABox Noise

The noise in the ABox can either be non-propagable or propagable.

Definition 1. Triple corruption:

Triple corruption is the process of morphing an existing triple in an RDF graph by changing one of the triples' resources

For example, a triple

```
subject property object1 .
```

can be corrupted into

```
subject property object2 .
```

given that $\text{object1} \neq \text{object2}$

Definition 2. Non-propagable noise

Non-propagable noise is any corrupted triple in the input graph that does not have any impact on the inference graph.

This can occur in one of three 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. For example, in the LUBM ontology:

```
doctoralDegreeFrom rdfs:subPropertyOf
degreeFrom.
mastersDegreeFrom rdfs:subPropertyOf
degreeFrom.
```

If the following triple from LUBM1:

```
_:FullProfessor7 mastersDegreeFrom <
http://www.University901.edu>
```

is corrupted to:

```
_:FullProfessor7 doctoralDegreeFrom <
http://www.University901.edu>
```

both triples will generate the inference:

```
_:FullProfessor7 degreeFrom <http://www.
University901.edu>
```

Definition 3. Propagable noise

Propagable noise is any corrupted triple in the input graph that changes the inference graph.

This occurs when the corrupted triple does not fall under any of the three cases mentioned previously. The necessary conditions for RDFS rules to generate noisy inference from a corrupted triple are discerned in the following:

First, the input patterns of the premises of the RDFS rules ([29]) are classified as TBox pattern or ABox pattern. Table 1 shows the results of this classification. The rules that have only TBox type patterns such

Table 1
Patterns types of RDFS rules premises

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

as *RDFS5*—which defines the properties hierarchy—and *RDFS11*—which defines the classes hierarchy—are excluded because any corruption of the triples that match these patterns will induce TBox noise. Again, this lies outside the scope of this research. 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 each of these rules to generate a noisy inference from the corrupted triple are identified. In plain English, if we corrupt the triple

subject property1 object .

into

subject property2 object .

the *RDFS2* rule will generate a noisy inference if and only if *property2* has a domain defined in the ontology and *property1* either does not have a specified domain or has a domain that is different than the domain of *property2*.

3.2. Ground-Truthing and Noise Induction

In the synthetic dataset obtained from LUBM, the noise was fabricated using the concepts that were in-

troduced in the taxonomy. Real world data does not necessitate noise induction as the noise is an inherent feature within this data genre.

3.2.1. Ground-Truthing in LUBM1

Lehigh University Benchmark (LUBM) [30] is a benchmark for Semantic Web repositories. The LUBM ontology conceptualizes 42 classes from the academic domain and 28 properties describing these classes' relationships. Using Univ-Bench Artificial Data Generator (UBA) [31], LUBM1—an RDF graph of one hundred thousand triples—was generated. This graph contains 17189 subject-resources within 15 classes. Table 3 lists the number of resources per class. Some of the resources have more than one class such as a student being a *GraduateStudent* and a *TeachingAssistant* at the same time.

Let R be the set of these subject-resources. For each resource r in R , a graph g is built by running the following SPARQL Protocol and RDF Query Language (SPARQL) query:

```
DESCRIBE <r>
```

which constructs a graph description of the resource r . Listing 1 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 inference is generated according to the LUBM ontology. For this step, Jena [26]—a state of the art toolkit for Semantic Web technologies that has RDFS and OWL reasoning capabilities—was used. Let I be the set of inference graphs obtained from the Jena reasoner. Listing 2 contains the inference graph of the input graph in Listing 1.

Finally, G and I are split into 60% training, 20% validation and 20% testing sets using a stratified splitting technique. The resource class is used as the label for the stratification. The goal of the stratification is to have the same 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. Let G_{train} , G_{val} , G_{test} , I_{train} , I_{val} and I_{test} be the training, validation and test sets. The input of the supervised learning algorithm is the set G_{train} , the target is I_{train} and the goal is to learn the inference generation.

3.2.2. Noise Induction in LUBM1

In [6], the authors proposed a methodology for noise induction in LUBM. They construct three datasets by

Table 2
Propagable noise by rule-based RDFS reasoners

RDFS rule	Original triple	Corrupted triple	Conditions	Noisy inference
RDFS2	yyy aaa zzz .	yyy aaa' zzz .	$(aaa' \text{ rdfs:domain } xxx' .)$ $\wedge ((\neg \exists xxx, aaa \text{ rdfs:domain } xxx .)$ \vee $(\forall xxx, aaa \text{ rdfs:domain } xxx$ $\implies \neg(xxx = xxx'))))$	yyy rdf:type xxx' .
RDFS3	yyy aaa zzz .	yyy aaa' zzz .	$(aaa' \text{ rdfs:range } xxx' .)$ $\wedge ((\neg \exists xxx, aaa \text{ rdfs:range } xxx .)$ \vee $(\forall xxx, aaa \text{ rdfs:range } xxx$ $\implies \neg(xxx = xxx'))))$	zzz rdf:type xxx' .
RDFS7	xxx aaa yyy .	xxx aaa' yyy .	$(aaa' \text{ rdfs:subPropertyOf } bbb' .)$ $\wedge ((\neg \exists bbb, aaa \text{ rdfs:subPropertyOf } bbb .)$ \vee $(\forall bbb, aaa \text{ rdfs:subPropertyOf } bbb .$ $\implies \neg(bbb = bbb'))))$	xxx bbb' yyy .
RDFS9	zzz rdf:type xxx .	zzz rdf:type xxx' .	$(xxx' \text{ rdfs:subClassOf } yyy' .)$ $\wedge ((\neg \exists yyy, xxx \text{ rdfs:subClassOf } yyy .)$ \vee $(\forall yyy, xxx \text{ rdfs:subClassOf } yyy .$ $\implies \neg(yyy = yyy'))))$	zzz rdf:type yyy' .

Table 3
Number of resources per class in LUBM1

Class	Number of resources
ub:Publication	5999
ub:UndergraduateStudent	5916
ub:GraduateStudent	1874
ub:University	979
ub:Course	828
ub:GraduateCourse	799
ub:ResearchAssistant	547
ub:TeachingAssistant	407
ub:ResearchGroup	224
ub:AssociateProfessor	176
ub:AssistantProfessor	146
ub:FullProfessor	125
ub:Lecturer	93
ub:Department	15

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> .

```

corrupting type assertions according to a given noise level. The three datasets are described as follows:

RATA In this dataset, the instances of type *TeachingAssistant* are 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

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 .

```

Person. In theory, this type of noise will not affect the inference thus a rule-based reasoner should not be affected. It is reasonable to check if a deep reasoner is also resilient to this type of noise.

UGS In this dataset, the instances of type *GraduateStudent* are 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 In this dataset, the instances of type *Course* are corrupted to be of type *GraduateCourse*. This type of noise is also non-propagable.

In this research, this technique for noise induction in LUBM is adopted by creating three types of datasets: *RATA_p*, *UGS_p* and *GCC_p* where the noise level *p* is varied within {0, 15, 50, 100} percent.

As the authors of [6] focus only on noisy type assertions, two additional datasets were created with noisy property assertions for the purpose of this research as well.

TEPA In this dataset 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 In this dataset the property *advisor* is corrupted to be *worksFor*. This noise is non-propagable as the property *worksFor* does not have any do-

main or range specification in the LUBM ontology, but by removing the property *advisor* the type inference that was deducted about the student and his advisor is lost.

The datasets *TEPA_p* and *WOAD_p* were also created by varying the noise level within {0, 15, 50, 100} percent. More precisely in *TEPA₁₀₀* and *WOAD₁₀₀*, every graph in the dataset that is being corrupted will have only one corrupted triple. For example if a publication has more than one publication author, only one of them will have a corrupted property assertion.

3.3. Ground Truthing the Scientists Dataset from DBpedia

[3] defines DBpedia as: “a community effort to extract structured information from Wikipedia and to make this information available on the Web” in the form of LOD [32]. From the DBpedia cloud, a dataset of scientists’ descriptions was built. To obtain the list of scientists in DBpedia, the following SPARQL query was run against DBpedia endpoint.

```

prefix dbo: <http://dbpedia.org/ontology/>
select distinct ?scientist
where {
    ?scientist a dbo:Scientist .
}

```

This query returns 25760 URIs for scientists’ descriptions. According to the LOD principles, these URIs are dereferenceable. The scientists’ RDF descriptions were obtained by fetching these URIs. 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*, *University*, *Place* and *Award*. To get the list of places in DBpedia that are related to scientists—either by being their place of birth or work etc.—and not all the places in DBpedia, this SPARQL query was run:

```

prefix dbo: <http://dbpedia.org/ontology/>
select distinct ?place
where {
    ?scientist a dbo:Scientist .
    ?scientist ?property ?place .
    ?place a dbo:Place .
}

```

For the classes *EducationalInstitution* and *Award* similar queries to this one were used. Table 4 lists the number of resources per class in the scientists dataset. The

Table 4

Number of resources per class in the scientists dataset

Class	Number of resources
dbo:Scientist	25760
dbo:Place	22035
dbo:EducationalInstitution	6048
dbo:Award	1166

total number of triples obtained in the scientists dataset is $\simeq 5.5$ million triples (5578576 precisely).

No artificial noise was induced in this dataset as it already has pre-existing noise. Using an empirical assessment of a DBpedia sample, [28] identified 17 quality issue types and estimated that 11.93% of DBpedia triples have at least one of these quality issues. These issues can manifest in noisy type assertions or noisy property assertions. An example of noisy type assertion in DBpedia is the resource *dbo:United_States* being of type *dbo:Person*. This is not one of a kind example. There are 1761 resources in DBpedia that are of types *dbo:Person* and *dbo:Place* simultaneously, which obviously indicates that one of them is a noisy triple. In the scientists dataset, there are 94 of these resources, which can be enumerated using this query:

```
prefix dbo: <http://dbpedia.org/ontology/>
select distinct ?place_person
where {
  ?scientist a dbo:Scientist .
  ?scientist ?property ?place_person .
  ?place_person a dbo:Place .
  ?place_person a dbo:Person .
}
```

4. Layered Graph Model for RDF

Despite its effectiveness as a standardized “framework for representing information in the Web” ([33]) and as an essential building block for the Semantic Web, the graph representation for the RDF model remains an open question in the Semantic Web 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 detailed later in this section. The motivation behind the research efforts to represent the RDF model as a graph that conforms to the graph theory can be summed up in the following:

1. Taking advantage of the well established mathematical foundations of the graph theory

2. Exploiting the recent advances in graph databases which inherently support graph querying—such as finding the shortest path—as opposed to storing RDF graphs in relational databases that were not designed for graph algorithms

Other motivations that are closely related to the theme of this research are:

3. Representing RDF graphs in a format suitable for neural network input
4. Capitalizing on the emerging research on deep learning for graphs
5. Stating the RDFS reasoning problem as a graph completion problem

4.1. Preliminary Notions

[34] defines graphs as: “A graph G is an ordered triple $(V(G), E(G), \Psi_G)$ consisting of a nonempty set $V(G)$ of vertices, a set $E(G)$, **disjoint** from $V(G)$, of edges, and an incidence function Ψ_G that associates with each edge of G an unordered pair of (not necessarily distinct) vertices of G . [34]”

An RDF graph can be defined using these formalisms from [35–37] (that is updated in this paper to conform to the more recent RDF 1.1 recommendation [33]):

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:

$Subj(T)$ be the set of subjects from $(I \cup B)$ that occur in the triples of T

$Pred(T)$ be the set of predicates from I that occur in the triples of T

$Obj(T)$ be the set of objects from $(I \cup B \cup L)$ that occur in the triples of T

$Subj-obj(T) = Subj(T) \cup Obj(T)$

4.2. Existing Graph Models for RDF

There are three main structural differences between RDF graphs and simple graphs which are outlined as follows:

Multigraph Unlike simple graphs, two RDF nodes can be linked with more than one link of different labels.

Heterogeneous Nodes in RDF graphs are connected by different types of edges (labeled edges).

Multivalent nodes and edges a node in RDF graphs can also be an edge at the same time.

4.2.1. Labeled Directed Graph Model

This is the graph-model described in the *RDF 1.1 Concepts and Abstract Syntax* [33]. In this model, the nodes $V(T)$ of an RDF graph T are the elements of the set $Subj-obj(T)$ and the edges $E(T)$ are labeled edges with elements from $Pred(T)$ (Fig. 2).

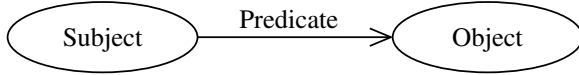


Fig. 2. A triple in the labeled directed graph model [33]

When triples from the ontology are described according to this graph model, two main challenges arise:

- If the predicate in one triple becomes the subject in another triple (this case occurs when describing the domain and range of properties such as in the example in Figure 3), then the predicate will become a node as well.

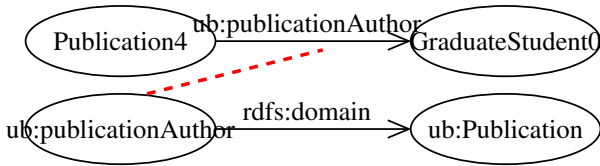


Fig. 3. Intersecting nodes and edges sets in RDF

In this example

$$V(T) \cap E(T) = \{ub:publicationAuthor\}$$

which contradicts the requirement in the graph definition from [34] that E and V should be disjoint.

- If both the subject and object of the triple are predicates in other triples (this case occurs when defining a property hierarchy such as in the example in Figure 4), then this relation is represented by a labeled directed edge between two edges. This means that the incidence function Ψ_T has values inside the set $E(T)$, which again contradicts the definition from [34] that restricts the range of Ψ_T to pairs of vertices in $V(T) \times V(T)$

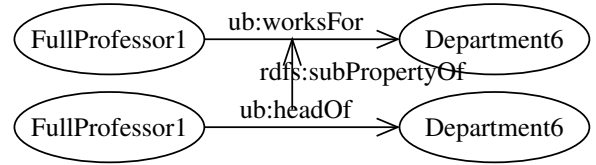


Fig. 4. Edge between edges in RDF graph model

To overcome these challenges, the existing literature proposes several graph models to represent the RDF data model. These graph models can be categorized as follows: Bipartite graph, Hypergraph based approaches and Metagraph approach.

4.2.2. Bipartite Graph Model

One of the earliest proposed graph-based models for RDF was the bipartite graph model [38]. In this model, an RDF graph T is represented as two disjoint sets of nodes: the first set contains the union of resources $Subj(T) \cup Obj(T) \cup Pred(T)$ and the second set of nodes is equal to the set of triples. Each resource in the first set of nodes is connected with a labeled edge to a node in the second set. The label of the edge indicates the role of the resource in the triple i.e. subject, predicate or object.

This model was adopted in some of the preliminary experiments, which showed that it is not suitable for the graph-to-graph based RDFS reasoning approach. The reason is that the representation of the inference graph in this model will contain new nodes denoting the inferred triples. By stating the RDFS reasoning problem as a graph completion problem, the introduction of new nodes—and not only new edges—makes the problem even more difficult to solve.

4.2.3. Hypergraph Models

The Mathworld [39] encyclopedia defines a hypergraph as: “a graph in which generalized edges (called hyperedges) may connect more than two nodes.”

In the class of models that represent RDF as a hypergraph [36, 40–42], each hyperedge connects three nodes which are the subject, the predicate and the ob-

ject of the triple. This forms a 3-uniform hypergraph. The authors of [36] use a directed hypergraph in order to reduce the size complexity and the load time of RDF graphs.

4.2.4. Metagraph Model

A more recent research effort ([43]) uses the metagraph model proposed in [44]. In metagraphs, an edge connects a set of nodes (called metavertex) to another set of nodes. This model is useful in representing RDF reification and N-ary relations [43]. RDF reification consists of expressing statements about RDF statements—for example, to record their provenance, time of collection, validity span etc. In the metagraph model, the reification can be expressed as an edge between the reification subject and the set of nodes representing the original statement.

4.3. Layered Graph Model for RDF

Even though the models described previously were suggested to propose a graph model for the RDF data model, they 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 neither coincide with RDFS reasoning nor neural network input. The reason why these models are not suitable for neural network input is that they use complex graph models beyond simple directed graphs. 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.

4.3.1. Notations and Definitions

In Table 1, 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)

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:

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

$$(e_i, e_j) \in E_l \iff \begin{cases} (e_i, p_l, e_j) \in T \\ e_i \in G(Subj-obj(T)) \\ e_j \in G(Subj-obj(T)) \end{cases}$$

Again, 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 two RDF graphs have the same layered directed graph representation if and only if their RDFS inference graphs according to the ontology O are isomorphic.

5. Encoding and Embedding of RDF Graphs

The encoding and embedding stages aim to transform RDF graphs into a format suitable for neural network input. Firstly, a simplified version of the encoding/decoding technique that is applicable to small RDF graphs is described. This will lay the ground to present a more complex approach that can be used for both small and large RDF graphs. Finally, the embedding phase is illustrated.

5.1. RDF Graphs Encoding Requirement

The way the generic methodology of supervised machine learning is utilized in this work is depicted in Figure 5 where the pair of (input, target) is the input graph and its corresponding inference. In a nutshell, the input graph g and its corresponding inference i are encoded. The encoded representations of these graphs are then used in the training phase. The encoding algorithm outputs an encoded version of the graph and its encoding dictionary that will be used in the decoding phase to regenerate the original graph.

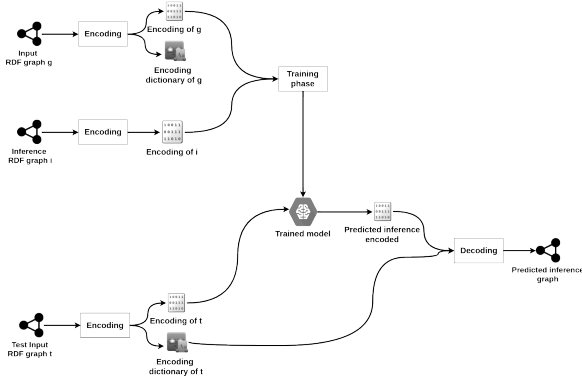


Fig. 5. Encoding/decoding in training and inference phases

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

Fig. 6. 3D Adjacency matrix

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 .

5.2. Simplified Encoding/Decoding Technique

In addition to preparing the RDF graph for input into a neural network, the main goal of the encoding phase is to capture the pattern similarities between graphs in such a way that “similar” graphs will have similar encodings. 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).

The size of the input RDF graphs plays an important role in determining how it will be encoded. While the “complex encoding/decoding technique” works on both small and large RDF graphs, it is salient to detail the “simplified encoding/decoding technique”—which works only on small RDF graphs—in order to comprehensively describe the complex technique.

5.2.1. 3D Adjacency Matrix

The first step of the multi-layer encoding approach is to create a 3D adjacency matrix, where each layer is the adjacency matrix relative to one property (Fig. 6). An ID must be assigned to each resource in the RDF graph to model it as 3D adjacency matrix.

In this approach, two dictionaries were created; one for the subject and objects IDs—which is split into a

Table 5
Properties_dictionary sample

Property	ID
rdf:type	0
...	...
ub:memberOf	10
ub:takesCourse	11
ub:telephone	12
ub:name	13
ub:emailAddress	14
ub:publicationAuthor	15
ub:undergraduateDegreeFrom	16
ub:advisor	17
...	...
ub:degreeFrom	31

global and a local dictionary—and one for the properties IDs. The global resources dictionary contains the subjects and objects resources that are used throughout the G set (which are basically the RDFS classes in the ontology). The local dictionary contains the resources that are specific to a particular graph g in G .

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* shown in Table 5, the list of properties is collected using the following 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 .
}
```

Table 6
Global_resources_dictionary sample

Resource	ID
...	...
ub:ResearchAssistant	18
ub:GraduateStudent	19
ub:University	20
ub:Employee	31
ub:Professor	33
ub:Person	36
ub:Organization	39
...	...

Table 7
Local_resources_dictionary sample for the graph g in Table ??

Resource	ID
_:GraduateStudent9	57
"GraduateStudent9"	58
< http : //www.Department5.University0.edu >	59
_:Professor7	60
...	...

```
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* shown in Table 6.

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. For example, the resources in the graph g of Listing 1 will have the *local_resources_dictionary* in Table 7.

5.2.2. 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 to not 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*.

5.2.3. 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 than 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.

5.2.4. Limitations

As previously stated, the main goal of the encoding phase is to capture the pattern similarities between graphs describing resources of the same type. This can be achieved by the simplified encoding technique when the cardinality of each property within these graphs is variable within a small range. For example, 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. Table 11 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 [45] has 40 objects for the property *RDF:type* 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 [46] has only 12 links for the property *RDF:type* and all of the objects are in *global_resources_dictionary*. This implies that the *RDF:type* 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 *dbp:Brown_University*, *dbp:Southern_Methodist_University* and *dbp: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 *dbp: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 encoding/decoding technique is proposed in the next section.

5.3. Advanced Encoding/Decoding Technique

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.

5.3.1. Challenges

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:

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 *RDF:type*.

The inference graph has two subject-object resources: *dbr:James_Hendler* and *dbo:Person*. As they appear in a triple with the property *RDF:type*, 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 *RDF:type* 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 *RDF:type* 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 *RDF:type* property. Whenever any resource is added to the *local_resources_dictionary* of any property when encoding the inputs graphs, it should be added to the *local_resources_dictionary* of the *RDF:type* 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*.

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.

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. 7 (zoomed in Fig. 14), a set of disconnected subgraphs 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)
}

```

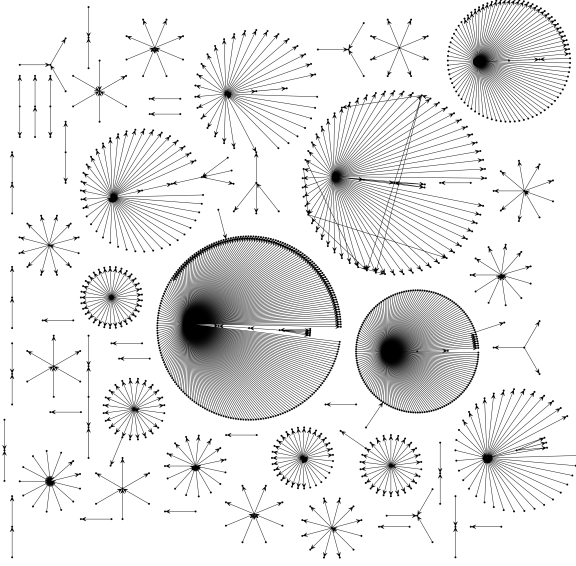



Fig. 7. The network of the relation *RDFS:subPropertyOf* in the DBpedia ontology (depicted without labels for visibility)

An undirected graph is then created using the Python library networkX [47]. 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*.

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 subjects and objects resource, the size of the 3D adjacency matrices becomes unmanageable. For example, the DBpedia ontology contains 3006 properties and 1576 subjects and objects 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 re-

quires 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 through out 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.

5.3.2. 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.

5.4. Graph Words

At this stage, every RDF graph is encoded as a 3D adjacency matrix of size: $(nb_properties, max_nb_resources, max_nb_resources)$ where each layer represents an adjacency matrix according to one property. $nb_properties$ being the number of usable properties and $max_nb_resources$ the maximum number of resources in the encoded graphs.

Proposition 2. The maximum number of possible layouts of layers in a dataset of size $dataset_size$ is equal to:

$$\min \begin{cases} 2^{max_nb_resources^2} \\ dataset_size * nb_properties \end{cases}$$

Proof. Let lp be the layer of the property p in the 3D adjacency matrix and lp_size be the number of elements in lp . lp is a matrix of size $max_nb_resources \times max_nb_resources$ so

$$lp_size = max_nb_resources^2$$

Each element $lp_{i,j}$ in the layer has two possible values:

- 1 if the resource r_i having the ID i in the encoding dictionaries is linked to the resource r_j with ID j according to the property p (i.e. if the encoded graph contains the triple $(r_i \ p \ r_j \ .)$)
- 0 otherwise

Which means that there are 2^{lp_size} possible layers layouts.

In a dataset of size $dataset_size$ there are $dataset_size * nb_properties$ layers.

If $(dataset_size * nb_properties) \leq 2^{lp_size}$ and all the layers in the dataset have different layouts then the dataset contains $(dataset_size * nb_properties)$ layouts. Otherwise if $(dataset_size * nb_properties) > 2^{lp_size}$ then the layers in the dataset cannot be all different and the maximum number of layouts is equal to 2^{lp_size} . \square

When encoding an RDF graph from the LUBM1 dataset—which contains 17189 RDF graphs—a 3D adjacency matrix of size $(32, 80, 80)$ is obtained. According to Proposition 2, the maximum number of layers layouts is equal to:

$$\min(2^{64^2} \simeq 10^{1233}, 32 * 17189) = 550048 \text{ possible layouts.}$$

The actual number of layouts when encoding LUBM1 is much smaller than the theoretical boundary of possible layouts. When using the simplified encoding technique, 547 different layouts are obtained in the encodings of the input graphs set G and 739 layouts in the encodings of the inference graphs set I . And, when using the advanced encoding technique, an even smaller number of different layouts is obtained: 107 and 469 for the sets G and I respectively. This observation is a good indication that the encoding algorithm 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. 8. 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: instead of saving redundant layers’ layouts across the dataset, only the ID of the layer’s layout along with a catalog of layouts is saved.
2. Having an RDF graph represented as a sentence of graph words allows for the exploitation of the research results in neural machine translation—which is detailed in the next chapter.

The full algorithm for converting a dataset of RDF graphs to a corpus of graph words is detailed in Algorithm 4.

5.5. Layered Embedding of RDF Graphs

At this stage, there is a parallel corpus of graph words for the input and inference graphs. Every input graph and its corresponding inference graph are represented as a sequence of graph words. This representation has the following drawbacks:

- Handling of “unknown” graph words
- Insensitivity to graph words similarities
- Graph words embedding

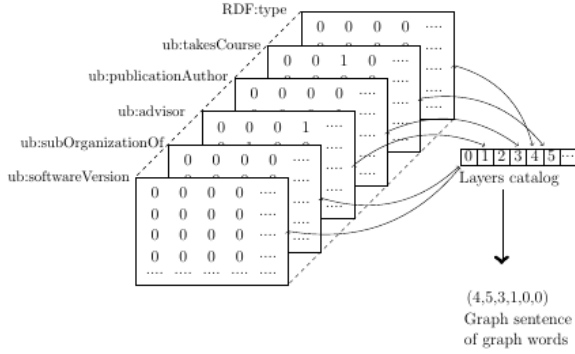


Fig. 8. From a 3D adjacency matrix to a sentence of graph words

5.5.1. Handling of “Unknown” Graph Words

When encoding a sentence from a natural language corpus, there is a special token for *unknown* words. The reason there are *unknown* words is that usually the dictionary of words is limited in size and contains only the words that are most relevant to the learning task. Every *unknown* word is assigned the same ID. This is not a deterrent to most learning tasks in natural language, such as topic categorization or sentiment analysis.

In the parallel corpus of graph words, *unknown* graph words can be encountered for one of two reasons:

1. A graph word seen only in the test set but not previously during the training phase.
2. When inducing noise, most of the graph words will be unknown.

Proposition 3. 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.

Proof. To prove this, two graphs having the same input representations but having different targets should be built.

Let

- *graph1* and *graph2* be two input RDF graphs from the test set.
- *inference1* and *inference2* be the inference graphs generated using a rule reasoner from *graph1* and *graph2* respectively.
- $Catalog_G$ be the catalog of graph words of the input graphs.
- n be the number of properties in the dataset.
- $[gw1_1, ..gw1_n]$, $[gw2_1, .., gw2_n]$, $[iw1_1, .., iw1_n]$ and $[iw2_1, .., iw2_n]$ be the sequence of graph words

representing respectively the graphs *graph1*, *graph2*, *inference1* and *inference2*

Assuming that *graph1* and *graph2* layers are equal except in the first layer, there are two possible cases:

1. The layout of the first layer in *graph1* and/or the layout of the first layer in *graph2* are in $Catalog_G$, in which case $gw1_1 \neq gw2_1$ —which implies that $[gw1_1, ..gw1_n] \neq [gw2_1, .., gw2_n]$.
2. Both layouts of the first layers in *graph1* and *graph2* are not present in $Catalog_G$, in which case $gw1_1 = gw2_1 = \text{UNK}$ (UNK being the ID assigned to unknown graph words). In this case $[gw1_1, ..gw1_n] = [gw2_1, .., gw2_n]$ as the rest of the layers are equal. Contrarily, The output sequences $[iw1_1, .., iw1_n]$ and $[iw2_1, .., iw2_n]$ can be different because *graph1* and *graph2* are not isomorphic.

□

5.5.2. Insensitivity to Graph Words Similarities

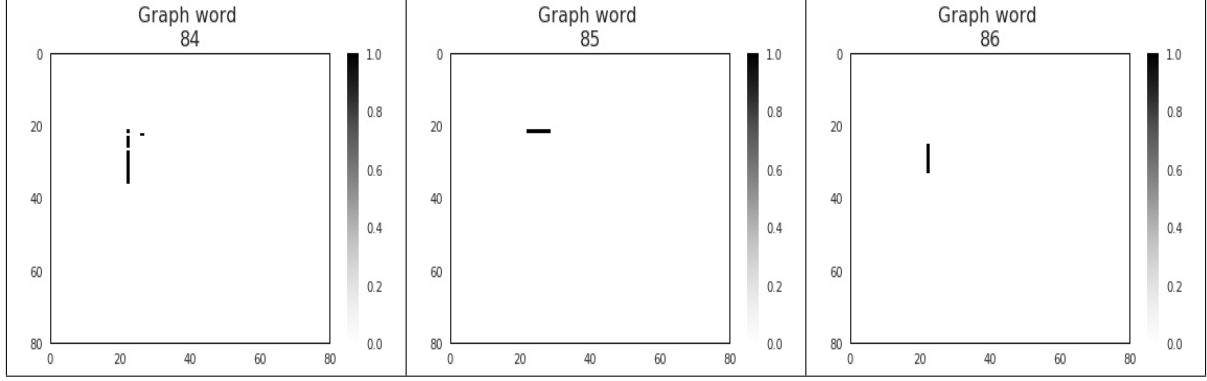
When incrementing the IDs of the graph words in the catalogs arbitrarily as was done in Algorithm 4, the similarity information between graph words is lost. Consequently, two similar graph words can have very different IDs and vice versa. Table 8 shows how the graph words with IDs 84,85 and 86 have very different layouts but their IDs are consecutive. This can be a deterrent to the learning process because similar sequences of graph words should have similar sequences for their target graph words.

5.5.3. Graph Words Embedding

In order to overcome the drawbacks of representing an RDF graph as a sequence of IDs of graph words, namely handling of unknown graph words and the lack in capturing the similarities between graph words, a technique used in natural language processing can be utilized. This technique is word embedding—except that in this case, instead of embedding words, graphs should be embedded. The goal of words embedding is to represent words as vectors in a continuous space that captures semantic and grammatical similarities, instead of using arbitrary IDs for words and encoding them using huge one-hot vectors.

As previously mentioned, RDF graphs are heterogeneous graphs where each node can be an edge. This hinders the use of most graph embedding algorithms that work on homogeneous graphs. The main reason of encoding an RDF graph as a 3D adjacency matrix is to obtain a sequence of layers where each layer contains

Table 8
Sample from the catalog of graph words in LUBM1 dataset



an adjacency matrix of a directed graph according to one property. Thus, graph embedding algorithms can be used on each layer of the 3D adjacency matrix. This method is termed “layered embedding”. Graph embedding algorithms can have many purposes including—but not limited to—node classification, clustering, link prediction, graph completion and graph reconstruction etc.

Proposition 3 proves that if two different graph words have the same ID, the learning process is compromised. Similarly, for the embedding of graph words: if two different graph words end up having the same embeddings, the learning process will be compromised as well. To ensure that different graph words can never have the same embeddings, the embedding algorithm must have perfect reconstruction accuracy.

Proposition 4. To ensure that different graph words have different embeddings, it is sufficient that the embedding algorithm has a perfect reconstruction accuracy.

Proof. By contradiction:

Let:

- GW be the group of graph words
- E be the group of graph words embeddings
- EMBED be the function that transforms a graph word g from G into its embedding e in E.
- RECONSTRUCT be the function that transforms an embedding e in E into a graph word g in G.
- gw_1 and gw_2 be two different graph words in GW.
- e_1 and e_2 be the embeddings of gw_1 and gw_2 respectively.

The embedding algorithm have a perfect reconstruction accuracy

$\Rightarrow \text{RECONSTRUCT} \circ \text{EMBED} = \text{Id}$ (The Identity function)

$$\Rightarrow \begin{cases} \text{RECONSTRUCT} \circ \text{EMBED}(gw_1) = gw_1 \\ \text{RECONSTRUCT} \circ \text{EMBED}(gw_2) = gw_2 \end{cases}$$

$$\Rightarrow \begin{cases} \text{RECONSTRUCT}(e_1) = gw_1 \\ \text{RECONSTRUCT}(e_2) = gw_2 \end{cases}$$

Assuming that $e_1 = e_2$

$$\Rightarrow \text{RECONSTRUCT}(e_1) = \text{RECONSTRUCT}(e_2)$$

$$\Rightarrow gw_1 = gw_2 \text{ (Contradiction)} \Rightarrow e_1 \neq e_2 \quad \square$$

In order to satisfy the criteria of perfect reconstruction accuracy, High-Order Proximity preserved Embedding (HOPE) algorithm[48] was used as it had the best reconstruction accuracy when tested on the catalog of graph words. From an adjacency matrix A of a directed graph with size m by m (where m is the number of nodes), HOPE embedding algorithm creates two vectors U and V of size m by d where d is the embedding dimension. The dot product of the vector U and the transpose of V produces a matrix P of size m by m where each element $P_{i,j}$ in P represents the probability of the node i being linked to the node j . When using a threshold to round these probabilities, the original adjacency matrix A should be reconstructed. Figure 15 shows the embedding and reconstruction results of the graph word with ID 84. In this example the embedding dimension is 2.

The graph words embedding also solves the problem of capturing the similarities between graph words. Table 13 shows four examples of graph words and their embedding vectors. Graph words 61 and 85 are similar and so are their embedding vectors. Graph words 100 and 104 are also similar but different from 61 and 85 and so are their respective embeddings.

6. Graph Words Translation for RDFS Reasoning

The graph words translation model is basically a sequence-to-sequence model with a Bidirectional Recurrent Neural Network (BRNN) [49] encoder. The designers of BRNN define it as: “The BRNN can be trained without the limitation of using input information just up to a preset future frame. This is accomplished by training it simultaneously in positive and negative time direction.” ([49]) In practice, this is achieved by training two Recurrent Neural Network (RNN)s: one on the input sequence and one on the reversed input sequence and combining the hidden states of both RNNs. This has been proven to improve the accuracy results for sequence learning [50].

6.1. Graph Words Translation for LUBM Inference

In designing the sequence-to-sequence model for graph words translation, keras [51]—a state of the art neural network library that can run the designed models on a variety of backends including TensorFlow [52]—was used. The model architecture is illustrated in Fig. 9 and its layers and hyper-parameters are described as follows:

- 17** is the length of the input sequence and the output sequence, which is equal to the number of usable properties in the *usable_properties_dictionary* obtained when encoding LUBM1 using the advanced encoding technique.
- cuDNNGRU** layer is an implementation of the Gated Recurrent Unit (GRU) [53] layer that uses the cuDNN [54] primitives. By switching to this implementation, the training time improved dramatically.
- 256** in the input layer is the size of the graph embedding.
- 0.2** In order to avoid over-fitting, three dropout layers of 0.2 are used: for input, encoder and decoder.
- 470** in the output layer is the size of the inference graph words catalog.

6.2. Graph Words Translation for DBpedia Inference

When using the advanced encoding technique for the Scientists dataset, the size of the graph words catalog was more than seventeen thousands graph words. This was an indication that learning to translate the sequence of these graph words will not be successful as most graph words appear only once in the training set.

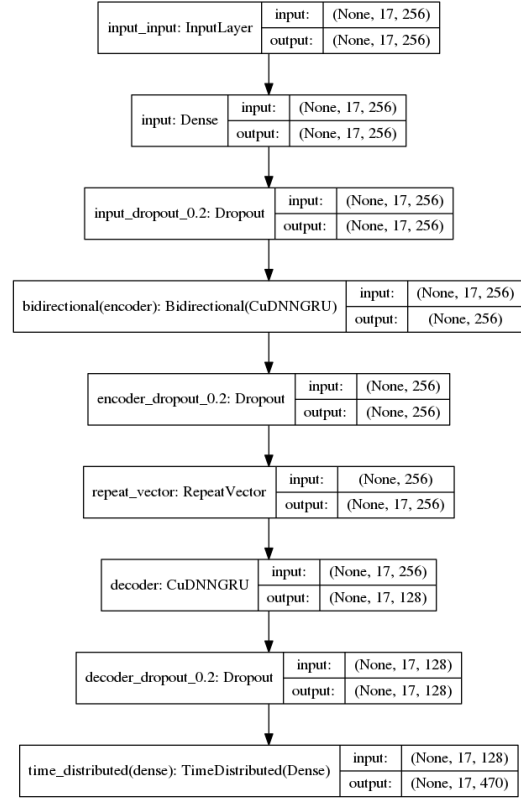


Fig. 9. Graph words translation model for LUBM

By examining the size of the graph words per property, it was clear that most of the graph words appear in the layer of the *RDF:type* property.

In order to reduce the number of graph words, the following simplification was necessary: instead of inferring all the types of resources that appear in the graph, only the types of the main resources are inferred. For instance, for the graph of Professor Hendler, instead of inferring the types of his universities and his birth town, only the types of his resource are inferred. This is actually what is returned by the state of the art SPARQL endpoints such as GraphDB [55] when running the *DESCRIBE* query with inference. By applying this simplification, the number of graph words dropped to 4393 for the input graphs and to 286 for the inference graphs, which is manageable.

The sequence-to-sequence model for the Scientists graph words translation has the same architecture as the LUBM1 model. Few hyper-parameters needed to be changed though to adapt the model to the data input and output sizes. Another distinction between this model and the LUBM1 model is the difference be-

tween the length of the input sequence and the length of the output sequence. In the LUBM1 model, both sequences had length of 17—which is the size of the usable properties in the LUBM ontology. In the scientists dataset, the size of the usable properties dictionary is 779. If 779 is used for the sequence output too, this slows down the training. Only 33 of these 779 properties are actually used in the inference graphs. This value can be obtained and calculated quickly by adding all the elements in the target array in the first axis and searching the indices where the sum is non zero. In the evaluation section, the training speeds for both models are compared.

The network model is depicted in Figure 16 and its main hyper-parameters are described as follows:

779 Size of the usable properties dictionary

791 Size of the embedding of the input graph words.
The original size of the embedding of the graphs from the Scientists dataset was 2048—compared to 256 in the case of LUBM1. The value 791 was obtained after dropping the indices that always contain zeros.

33 Size of the usable properties in the inference

286 Size of the catalog of the inference graph words

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.

7.2. Evaluation on LUBM1 Dataset

In this section, the training process and the inference results on intact as well as on noisy data are described.

7.2.1. Training Phase

Fig. 10 describes the training process on the LUBM1 dataset. After approximately 12 minutes of training, 98.4% validation accuracy was achieved. The best model weights are saved to be used for the test phase.

7.2.2. Evaluation on Intact LUBM1 Dataset

When testing the trained model on the intact LUBM1 test set, an overall accuracy of 98% was obtained. A break down of the accuracy per class as shown

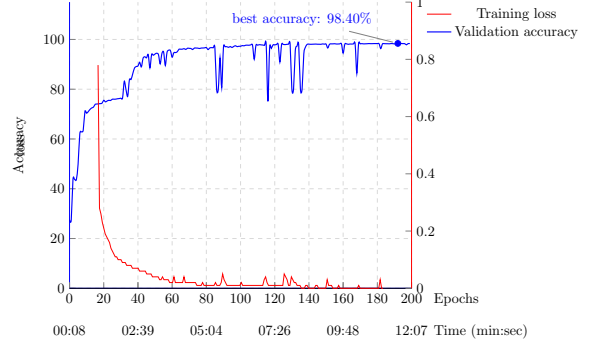


Fig. 10. Training results on intact LUBM1 data

in Figure 17, shows 100% accuracy for most classes that have over one thousand samples in the training. On the other hand, for the small classes, such as *ub:Department*—that has only 13 samples from which 7 will be used in training and 3 for validation—the test accuracy is only 13%. This confirms the rule of thumb that deep learning techniques require a large amount of data for training in order to obtain decent test accuracy.

7.2.3. Evaluation on Noisy LUBM1 Data

In this experiment, the trained model is tested on the noisy datasets which were created as described in Section 3. For the evaluation of the noise-tolerance capability, two metrics were designed:

Macroscopic metric: Per-graph accuracy This metric consists of measuring the percentage of corrupted graphs where their inference is equal to the inference of their corresponding intact graphs. As depicted in Fig. 11, for every graph g in the testing dataset, the corrupted graph g' is created according to the selected corruption method (i.e. UGS, TEPA, etc.). Then, the inference of the intact graph i as well as that of the corrupted graph i' are generated using Jena. The deep reasoner is used to generate the inference dr from the corrupted graph. Correct inferences are 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 In the previous metric, an inference is considered to be wrong if it is not isomorphic to the Jena inference of the intact graph. This overlooks how different the deep reasoner inference is to that of Jena. For instance, the per-graph accuracy metric does

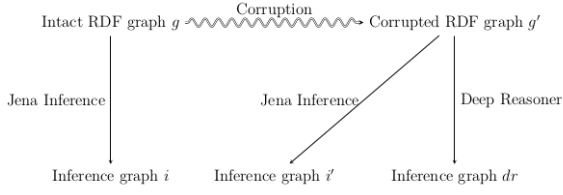


Fig. 11. Macroscopic metric: Per-graph accuracy

not indicate if the wrong inference contains only one triple that is different than Jena's inference or if the whole graph is different. Moreover, some triples generated by the deep reasoner, which were not generated by Jena, were in fact valid. To address these cases, a microscopic metric was designed where two materialization graphs are generated: the first materialization graph contains all the triples generated by Jena from the intact graphs and the second materialization graph contains all the triples generated by the deep reasoner from the corrupted graphs. Let J and DR be the set of triples in the graphs generated by Jena and the deep reasoner respectively. Then the quasi-confusion matrix with respect to Jena (depicted in Table 9) is computed as follows:

- True positives are the triples generated by the deep reasoner and by Jena as well—which are the triples in the set: $DR \cap J$
- False positives are the triples generated by the deep reasoner but not by Jena—which are the triples in the set: $DR \setminus J$
- False negatives are the triples generated by Jena but missed by the deep reasoner—which are the triples in the set: $J \setminus DR$
- True negatives are not defined because the deep reasoner is not a classifier that separates between valid and invalid triples. Rather, it is a generative model and any triple that is not generated by the deep reasoner can either be valid or invalid according to the open-world assumption. This is the reason the term quasi-confusion matrix was used. The precision and recall can be computed from this quasi-confusion matrix as they do not require the *true negative* value.

As mentioned previously, some of the triples generated by the deep reasoner and not by Jena—i.e. the false positives—can in fact be valid triples. In order to assess the validity of these triples, the SPARQL query:

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

Table 9

Confusion Matrix with respect to Jena inference

		Deep reasoner inference		total
		p	n	
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	

Table 10

Refined confusion matrix

ASK {subject property object .}

is run against a triple store containing an OWL-RL materialization of LUBM1. Let V be the subset of valid triples from the set $DR \setminus J$. The refined quasi-confusion matrix becomes as referenced in Table 10.

UGS UGS_p datasets were created by corrupting—according to the noise level p —the resources of type *GraduateStudent* to be of type *University*. This constitutes propagable type-assertion noise as defined in Definition 3. Every *GraduateStudent* resource in the corrupted graphs will be inferred to be of type *Organization* by a rule reasoner. Thus, it is not surprising that Jena reasoner has 0% per-graph accuracy in the UGS_{100} case. In other terms, all the inference graphs from the corrupted graphs i' are different than the inference graphs from the intact graphs i . The deep reasoner, on the other hand, achieved a per-graph accu-

racy of 93% on the test set of UGS_{100} , which shows exceptional noise tolerance capability.

Listing 3 shows an example of *GraduateStudent* graph that was distorted according to UGS noise into the graph shown in Listing 4. The correct deep reasoner inference is shown in Listing 5 and the Jena wrong inference is shown in Listing 6. An example of an input graph where the deep reasoner and Jena make wrong inference is shown in Listing 7, Listing 8, Listing 9 and Listing 10 for the original graph, distorted graph, deep reasoner inference and Jena inference respectively. It should be noted that the deep reasoner got the inference that Jena made a mistake on correctly, but made another mistake.

In the microscopic metric, the set J (the Jena inference from the intact graph) contains 10820 triples. The set DR (the deep reasoner inference from the UGS_{100}) also contains 10820 triples from which 129 triples are different than the Jena inference. When assessing the validity of the 129 false positive triples using the ASK SPARQL queries, 25 of these triples turn out to be valid. Using the quasi-confusion matrix, the precision for UGS_{100} is equal to:

$$Precision_{UGS_{100}} = \frac{10691 + 25}{(10691 + 25) + (129 - 25)} \simeq 98.04\%$$

and the recall is equal to:

$$Recall_{UGS_{100}} = \frac{10691 + 25}{(10691 + 25) + 129} \simeq 98.81\%$$

RATA $RATA_p$ datasets were created by corrupting—according to the noise level p —the resources of type *ub:TeachingAssistant* to be of type *ub:ResearchAssistant*. This constitutes non propagable type-assertion noise as defined in Definition 2. Even though this type of noise should not affect a rule based reasoner, our metrics show that Jena has an accuracy of only 78% on $RATA_{100}$ when compared to the Jena inference of the intact graphs. This happens when the inference of the corrupted graph contains triples of the form:

```
s RDF:type ub:TeachingAssistant .
```

while the inference of the intact graph does not contain such triple because it was already in the input graph. The deep reasoner per-graph accuracy is comparative to that of Jena in the $RATA_{100}$ case. The total number of triples in the set J generated by Jena from the intact graphs is 10820. The deep reasoner generated

9826 correct triples in DR from the $RATA_{100}$ dataset, missed 994 triples and generated 814 invalid triples. This results in the following precision and recall:

$$Precision_{RATA_{100}} \simeq 92.35\%$$

$$Recall_{RATA_{100}} \simeq 90.81\%$$

GCC GCC_p datasets were created by corrupting—according to the noise level p —the resources of type *ub:Course* to be of type *ub:GraduateCourse*. This is also non-propagable type-assertion noise. In GCC_{100} , the per-graph accuracy of Jena is 100% as expected. The deep reasoner on the other hand, has 66.87% per-graph accuracy only. However, a big portion of the inferences that are counted to be wrong because they are different than Jena inferences are actually valid. The graph example illustrated in Listing 11 and its respective deep reasoner inference shown in Listing 12 show correct inferences that were considered wrong as they are different than the Jena inference.

The microscopic metric aims to quantify these cases. Jena generates 3913 triples from the intact graphs GCC_0 . The deep reasoner missed 249 of these triples in the GCC_{100} case and generated 1611 triples that were not generated by Jena from which 627 are valid. The following listing contains two valid triples that were generated by the deep reasoner but not by Jena:

```
UndergraduateStudent132 RDF:type ub:Person .
Course37 RDF:type ub:Course .
```

The precision and recall of the deep reasoner for GCC_{100} are:

$$Precision_{GCC_{100}} \simeq 81.35\%$$

$$Recall_{GCC_{100}} \simeq 94.52\%$$

TEPA $TEPA_p$ datasets were created by corrupting—according to the noise level p —the properties *ub:teacherOf* into *ub:publicationAuthor*. Unlike the previous noisy datasets—which contain type-assertion noisy triples—**TEPA** and **WOAD** contain property-assertion noisy triples. **TEPA** contain propagable property-assertion noise and Jena had 0% accuracy in $TEPA_{100}$ as expected. The deep reasoner got 45.82% per-graph accuracy in $TEPA_{100}$. Jena generated 2062 triples from

the intact graphs $TEPA_0$. The deep reasoner generated 1861 valid triples from the corrupted dataset $TEPA_{100}$, missed 201 triples and did not generate any invalid triples. This results in the following precision and recall for $TEPA_{100}$:

$$Precision_{TEPA_{100}} = 100\%$$

$$Recall_{TEPA_{100}} \simeq 90.25\%$$

$WOAD$ $WOAD_p$ datasets were created by corrupting—according to the noise level p —the properties $ub:advisor$ into $ub:worksFor$. This is non propagable property-assertion noise because the property $ub:worksFor$ does not have any domain or range in the LUBM ontology. However Jena accuracy was 0% on $WOAD_{100}$ compared to the inference from intact graphs $WOAD_0$. This is because even though the Jena inference does not contain any invalid triple, it missed the triples that were generated using the property $ub:advisor$ domain and range. For instance, the inference from the intact graph indicates that the type of the student is $ub:Person$ and that his advisor is of type $ub:Professor$ but the inference from the corrupted graph does not contain these two triples.

The per-graph accuracy of the deep reasoner on $WOAD_{100}$ is also 0%. Jena generated 13301 triples from the intact graphs and the deep reasoner inference from $WOAD_{100}$ missed 4772 triples and added 8252 false positive triples. This results in the following precision and recall for $WOAD_{100}$:

$$Precision_{WOAD_{100}} \simeq 50.83\%$$

$$Recall_{WOAD_{100}} \simeq 64.12\%$$

Even though the deep reasoner had poor noise-tolerance in this case, there is an interesting pattern that was captured. For instance, when corrupting the triple:

```
GraduateStudent0 ub:advisor
  AssistantProfessor3 .
```

into

```
GraduateStudent0 ub:worksFor
  AssistantProfessor3 .
```

the inference from the deep reasoner contains these two triples:

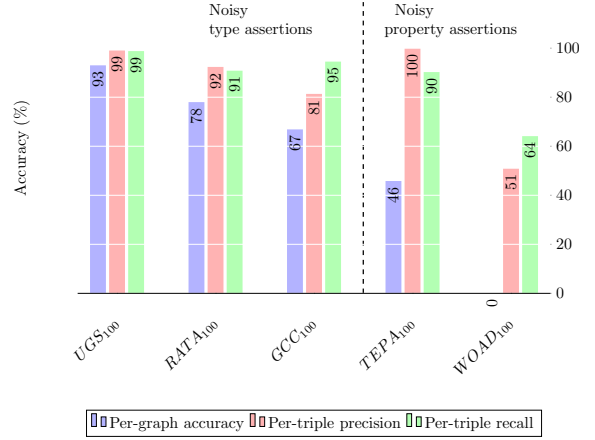


Fig. 12. Macro and micro evaluation on noisy LUBM1 datasets

```
AssistantProfessor3 RDF:type ub:University .
AssistantProfessor3 RDF:type ub:Organization
.
```

Both triples are obviously invalid but they show how the deep reasoner captured the implicit range of the property $ub:worksFor$. The range of this property is not encoded using $RDFS:range$ property in the ontology but rather using Web Ontology Language (OWL) rules as follows:

```
<owl:Class rdf:ID="Employee">
  <rdfs:label>Employee</rdfs:label>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Person"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#worksFor"/>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#Organization"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

In other words, even though the ground truth did not specifically contain the range of the $ub:worksFor$ property as we used RDFS reasoning, the deep reasoner captured that this property is connected to resources of types $ub:Organization$ and $ub:University$.

Section 7.2.3 summarizes the macro and micro evaluation on the 5 noisy datasets.

7.3. Evaluation on the Scientists Dataset

As detailed in Section 6, two models are trained on the scientists dataset. The first model is similar to the LUBM1 model—except for the hyper-parameters—having the same length of the input and output sequences. This model takes more than 16 hours to reach a validation accuracy of 87.76% (Section 7.3). As the output sequence is mainly filled with zeros except for 33 indices out of 779, we used a compressed version of the output containing these indices only.

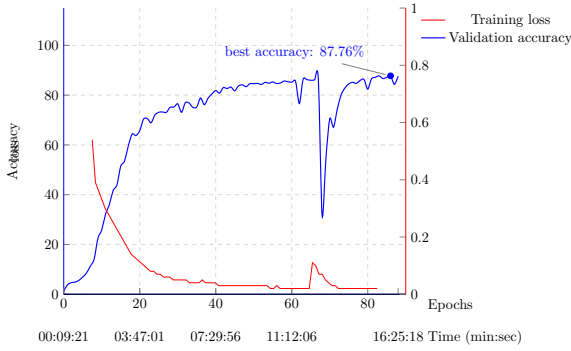


Fig. 13. Training results on DBpedia Scientists dataset

The optimized model reaches similar performance in approximately half the time.

7.3.1. Noise-Tolerance Evaluation on the Scientists Dataset

For the noise-tolerance evaluation, we used the person-place examples. Out of the 1761 person-place graph descriptions in DBpedia, the Scientists dataset contains 94. Unlike the LUBM1 case—where the training is performed on intact data and the test on controlled noisy data—the training on the scientists dataset is performed on noisy data. When an input graph contains a triple of the form:

```
r RDF:type dbo:Person .
```

the Jena reasoner infers that

```
r RDF:type dbo:Agent .
```

as the class *dbo:Person* is a subclass of *dbo:Agent*. For the person-place graphs, this constitutes a propagation of noise because *dbo:Agent* and *dbo:Place* are disjoint classes. In order to evaluate the noise-tolerance capability of the deep reasoner, we check if it inferred

that a person-place is of type *dbo:Agent*. Out of the 94 examples, 6 inferences only contain this noisy inference. However, some of the remaining 88 inference either contain false positive or missed valid triples that were inferred by Jena. 38 inference graphs are *perfect* in the sense that they contain exactly the inference from Jena minus the noisy triple. Some examples of these graphs are: *dbr:Socialist_Republic_of_Croatia*, *dbr:Teylers_Museum* and *dbr:Meta_River*. This makes up 40% of the noisy examples. For the remaining person-place graphs, a few inferences contain false positive triples that were not generated by Jena. For example, the deep reasoner inference from the *dbr:Big_Ben* graph, missed these two triples compared to Jena (The first triple should be missed):

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

and generated the following *extra* triple:

```
dbr:Big_Ben a dbo:HistoricPlace .
```

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

8. Conclusions 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 Semantic 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.

Towards achieving the main contribution of this research, the following sub-contributions were made:

Noise Intolerance Conditions In order to illustrate the intolerance of rule-based reasoners to noise in Semantic Web data, a taxonomy for the different types of noise that can be found in it was drawn. Additionally, the necessary conditions for a noise

type to be propagable (i.e. affect the inference) by any RDFS rule were discerned.

Layered Graph Model for RDF Even though the literature encompasses quite a few propositions for graph models for RDF, none of them is designed for RDFS reasoning specifically or for neural network input. We proposed a layered graph model for RDF data that fulfilled these requirements.

Graph Words Using the layered graph model, we proposed 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, which constitutes a novel approach for graph-to-graph learning.

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:

8.1. 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 [56] 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 adversarial generative models for graph words. Generative adversarial models, described in [57], 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.

8.2. OWL Reasoning

In this work, we tackled the problem of noise-tolerant RDFS reasoning. 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 [58], specifically using the Human Disease Ontology [59]. 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 [56], 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.

8.3. Training with multiple “A-Boxes”

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 LUBM1 dataset, we needed to adapt

the model hyper-parameters 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 [60] 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.

8.4. Towards the trust layer

In a recent positional paper titled "Semantic Web: Learning from Machine Learning" [61], Brickley describes his vision of how deep learning and Semantic Web fields can communicate and learn from each other. In this paper, we initiated the communication in one direction which is: deep learning for Semantic Web. The other direction, Semantic Web for deep learning, is also equally as important and very promising with lots of opportunities for research and subsequent discovery. One such research effort in that direction is [24] where the authors use Semantic Web 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 can 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 Semantic Web 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. Simplified Encoding Algorithm

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` \leftarrow `SIZE(properties_dictionary)`
`max_size` \leftarrow `max_local_dictionary_size` + `SIZE(global_resources_dictionary)`
`adjacency_matrix` \leftarrow `ZEROS(number_of_properties, max_size, max_size)`
function `ADD_RESOURCE(resource)`
| **if** `resource` in `global_resources_dictionary` \hookrightarrow
| **or** `resource` in `local_resources_dictionary` **then**
| | **return**
| **else**
| | `local_resources_dictionary[resource]` \leftarrow \hookrightarrow
| | `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` \leftarrow `properties_dictionary[p]`
| **if not** `is_inference` **then**
| | `ADD_RESOURCE(s)`
| | `ADD_RESOURCE(o)`
| `s_id` \leftarrow `LOOKUP_RESOURCE(s)`
| `o_id` \leftarrow `LOOKUP_RESOURCE(o)`
| `adjacency_matrix[p_id, s_id, o_id]` \leftarrow 1
| **return** `adjacency_matrix, local_resources_dictionary`

End

Appendix B. Simplified Decoding Algorithm

Algorithm 2 Simplified decoding algorithm

Input: adjacency_matrix, */* 3D adjacency matrix containing */*
 global_resources_dictionary, local_resources_dictionary, properties_dictionary
Output: rdf_graph
Begin:
 rdf_graph \leftarrow GRAPH() */* Creating an empty RDF graph */*
 inverted_properties_dictionary \leftarrow INVERT(properties_dictionary)
 inverted_global_resources_dictionary \leftarrow INVERT(global_resources_dictionary)
 inverted_local_resources_dictionary \leftarrow INVERT(local_resources_dictionary)
function REVERSE_LOOKUP(resource_id)
 | **if** resource_id in inverted_global_resources_dictionary **then**
 | | **return** inverted_global_resources_dictionary[resource_id]
 | **else if** resource_id in inverted_local_resources_dictionary **then**
 | | **return** inverted_local_resources_dictionary[resource_id]
 | **else**
 | | **ERROR, EXIT**

function DECODE(adjacency_matrix, inverted_global_resources_dictionary, inverted_local_resources_dictionary,
 inverted_properties_dictionary)
 | **for all** (p_id, s_id, o_id) in NON_ZEROS(adjacency_matrix) **do**
 | | p \leftarrow inverted_properties_dictionary[p_id]
 | | s \leftarrow REVERSE_LOOKUP(s_id)
 | | o \leftarrow REVERSE_LOOKUP(o_id)
 | | ADD_TRIPLE(rdf_graph, s, p, o)
 | **return** rdf_graph
End

Appendix C. 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 \leftarrow 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] ←  $\sqcup$ 
|       |   ← SIZE(usable_global_resources_dictionary)
|   else if resource not in local_resources_dictionaries [property_group] then
|       |   local_resources_dictionaries [property_group][resource] ←  $\sqcup$ 
|       |   ← -(SIZE(local_resources_dictionaries [property_group][resource]))

for all (s,p,o) in SORTED_TRIPLES_BY_PROPERTY(rdf_graph) do
|   if p not in properties_dictionary then
|       |   continue
|   else if p not in usable_properties_dictionary then
|       |   usable_properties_dictionary[p] ← SIZE(usable_properties_dictionary)
|   p_id ← usable_properties_dictionary[p]
|   if not is_inference then
|       |   ADD_RESOURCE(s,p)
|       |   ADD_RESOURCE(o,p)
|   s_id ← LOOKUP_RESOURCE(s,p)
|   o_id ← LOOKUP_RESOURCE(o,p)
|   APPEND(sparse_encoding, p_id, s_id, o_id)
End

```

Appendix D. Graph words creation algorithm

Algorithm 4 From RDF dataset to graph words corpus

Input: G, /* The set of input RDF graphs */
I /* The set of inference RDF graphs */
global_resources_dictionary, properties_dictionary

Output: X, /* Input corpus */
Y, /* Target corpus */
G_Catalog, /* Layouts catalog of the input corpus */
I_Catalog, /* Layouts catalog of the target corpus */
Local_Resources_Dictionaries /* A list containing the local_resources_dictionary */

Begin:
dataset_size ← SIZE(G)
index ← 0
X ← []
Y ← []
G_Catalog ← []
I_Catalog ← []
Local_Resources_Dictionaries ← []
while index < dataset_size **do**
| rdf_input ← G[index]
| inference ← I[index]
| local_resources_dictionary ← []

```

|   x_graph_sentence ← [ ]
|   y_graph_sentence ← [ ]
|
|   /* First we encode the input graph */
|   adjacency_matrix, local_resources_dictionary ← ENCODE(rdf_input, global_resources_dictionary, local_resources_dictionary,
properties_dictionary, is_inference=False)
|   Local_Resources_Dictionaries[index] ← local_resources_dictionary
|   for all layer in adjacency_matrix do
|       if LAYOUT(layer) not in G_Catalog then
|           APPEND(G_Catalog, LAYOUT(layer))
|           graph_word ← G_Catalog[LAYOUT(layer)]
|           APPEND(x_graph_sentence, graph_word)
|       X[index] ← x_graph_sentence
|
|   /* Then we encode the inference graph using local_resources_dictionary */
|   adjacency_matrix, local_resources_dictionary ← ENCODE(inference, global_resources_dictionary, local_resources_dictionary,
properties_dictionary, is_inference=True)
|   for all layer in adjacency_matrix do
|       if LAYOUT(layer) not in I_Catalog then
|           APPEND(I_Catalog, LAYOUT(layer))
|           graph_word ← I_Catalog[LAYOUT(layer)]
|           APPEND(y_graph_sentence, graph_word)
|       Y[index] ← y_graph_sentence
|       index ← index + 1
|   return X, Y, G_Catalog, I_Catalog, Local_Resources_Dictionaries
End

```

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

Table 11
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

Table 12
Possible number of links per properties in the Scientists dataset for the Scientist class

Property	Possible number of links
rdf:type	1,2,3, [5 .. 97], 99, 101, 103, 107, 110, 111, 112, 116, 118, 134, 135, 154
dbo:doctoralStudent	[1 .. 17], 19, 20, 21, 22, 26, 27, 28, 31, 32
dbo:knownFor	[1 .. 17], 19, 20, 23, 25, 28
dbo:field	[1 .. 12], 14, 17, 18, 22
dbo:influenced	[1 .. 11], 13, 16, 19, 21, 30
dbo:influencedBy	[1 .. 13], 15, 16, 18, 19

Listing 3: Original graph (LUBM1 description of GraduateStudent0)

```
_:Publication4 ub:publicationAuthor _:GraduateStudent0 .
_:Publication8 ub:publicationAuthor _:GraduateStudent0 .
_:Publication7 ub:publicationAuthor _:GraduateStudent0 .
_:Publication14 ub:publicationAuthor _:GraduateStudent0 .
_:Publication13 ub:publicationAuthor _:GraduateStudent0 .
_:GraduateStudent0 a ub:GraduateStudent,
    ub:ResearchAssistant ;
ub:advisor _:AssistantProfessor3 ;
ub:emailAddress "GraduateStudent0@Department0.University0.edu" ;
ub:memberOf <http://www.Department0.University0.edu> ;
ub:name "GraduateStudent0" ;
ub:takesCourse _:GraduateCourse16
    _:GraduateCourse50
    _:GraduateCourse64 ;
ub:telephone "xxx-xxx-xxxx" ;
ub:undergraduateDegreeFrom <http://www.University358.edu> .
```

Appendix F. Possible Number of Links per Properties in the Scientists Dataset for the Scientist Class

Appendix G. A zoom in on the network of the relation *RDFS:subPropertyOf* in the DBpedia ontology with labels

Appendix H. Graph Words Embedding and Reconstruction

Appendix I. Capturing graph words similarity via embedding

Appendix J. Graph words translation model for DBpedia

Appendix K. LUBM1 accuracy results per class

Appendix L. Examples of inference on noisy RDF graphs

Listing 4: Distorted graph (LUBM1 description of GraduateStudent0)

```

_:Publication4 ub:publicationAuthor _:GraduateStudent0 .
_:Publication8 ub:publicationAuthor _:GraduateStudent0 .
_:Publication7 ub:publicationAuthor _:GraduateStudent0 .
_:Publication14 ub:publicationAuthor _:GraduateStudent0 .
_:Publication13 ub:publicationAuthor _:GraduateStudent0 .
_:GraduateStudent0 a ub:University ,
    ub:ResearchAssistant;
ub:advisor _:AssistantProfessor3 ;
ub:emailAddress "GraduateStudent0@Department0.University0.edu" ;
ub:memberOf <http://www.Department0.University0.edu> ;
ub:name "GraduateStudent0" ;
ub:takesCourse _:GraduateCourse16
    _:GraduateCourse50
    _:GraduateCourse64 ;
ub:telephone "xxx-xxx-xxxx" ;
ub:undergraduateDegreeFrom <http://www.University358.edu> .

```

Listing 5: Deep reasoner correct inference from noisy input of type UGS

```

_:Publication4 a ub:Publication .
_:AssistantProfessor3 a ub:Employee,
    ub:Faculty,
    ub:Professor .
_:Publication8 a ub:Publication .
_:Publication7 a ub:Publication .
_:Publication14 a ub:Publication .
_:Publication13 a ub:Publication .
_:GraduateStudent0 a ub:Person ;
    ub:degreeFrom <http://www.University358.edu> .
<http://www.University358.edu> a ub:Organization,
    ub:University .

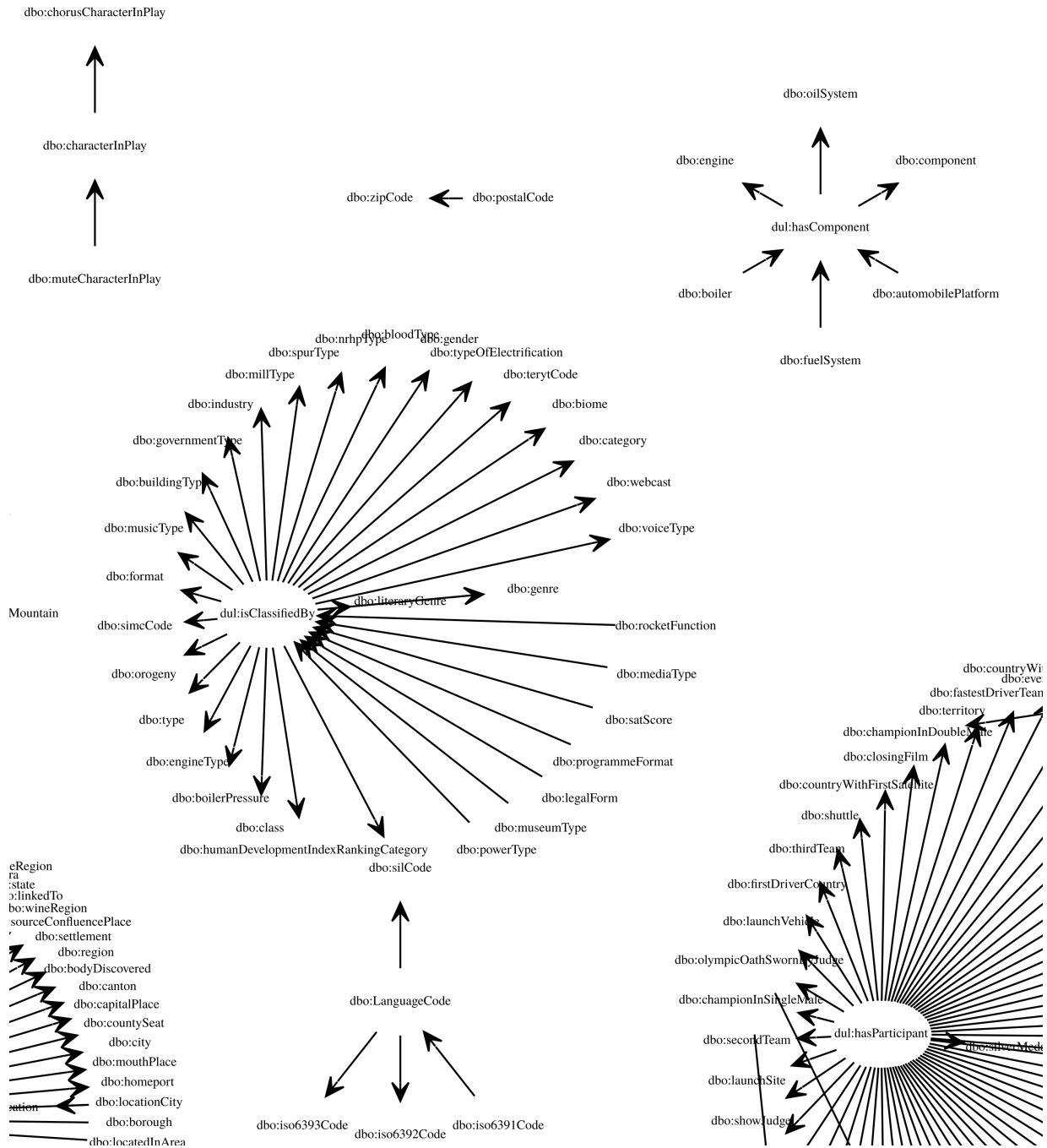
```

Listing 6: Jena inference from noisy input of type UGS

```

_:Publication4 a ub:Publication .
_:AssistantProfessor3 a ub:Employee,
    ub:Faculty,
    ub:Professor .
_:Publication8 a ub:Publication .
_:Publication7 a ub:Publication .
_:Publication14 a ub:Publication .
_:Publication13 a ub:Publication .
_:GraduateStudent0 a ub:Organization ,
    ub:Person ;
    ub:degreeFrom <http://www.University358.edu> .
<http://www.University358.edu> a ub:Organization,
    ub:University .

```

Fig. 14. A zoom in on the network of the relation *RDFS:subPropertyOf* in the DBpedia ontology with labels

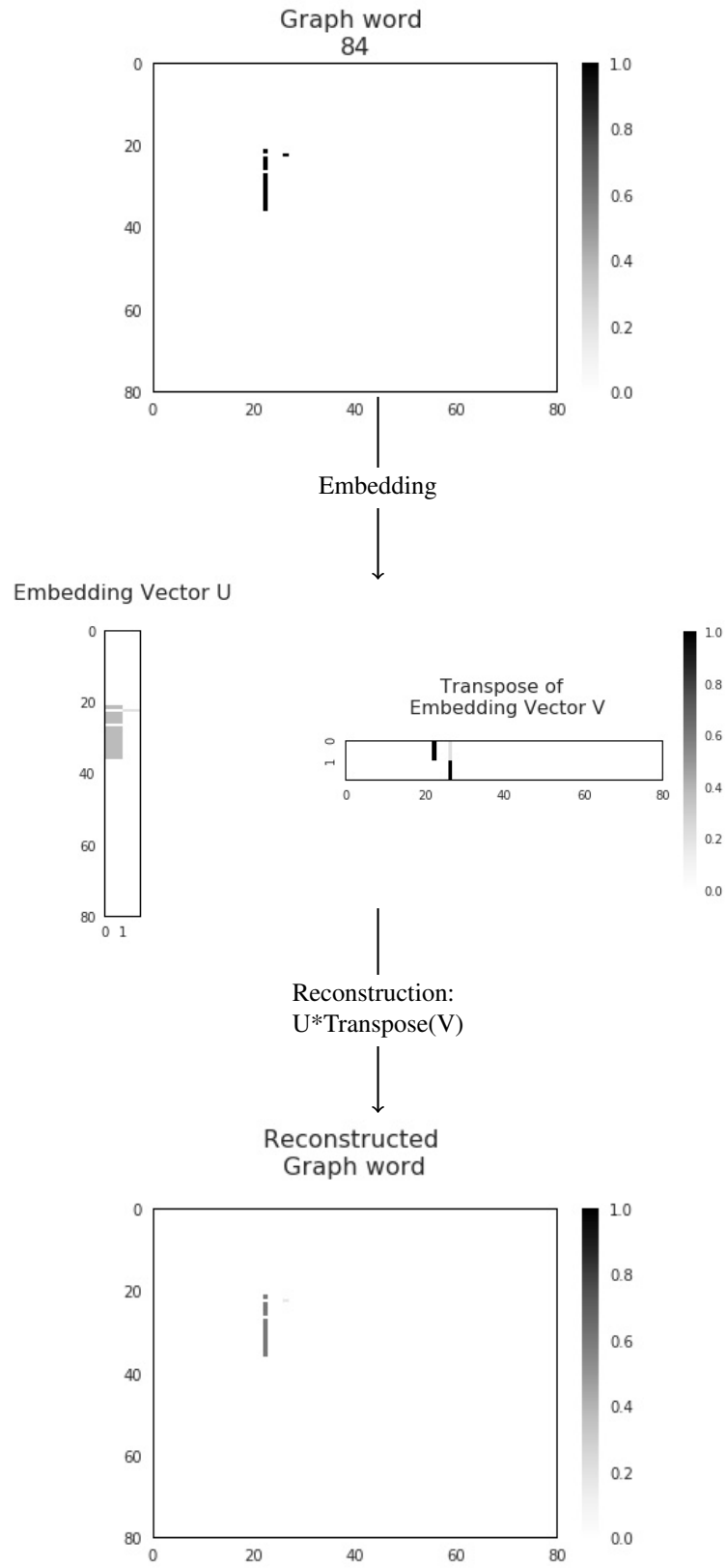
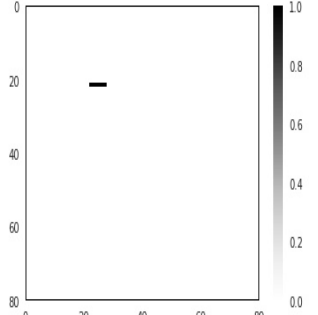
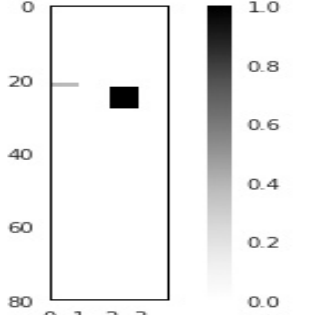
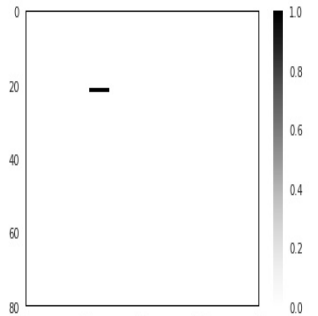
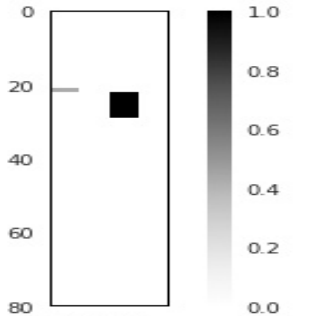
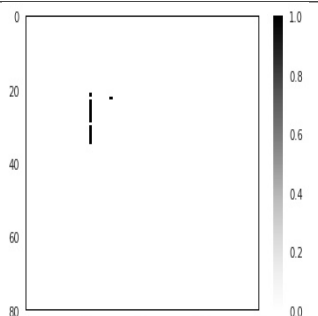
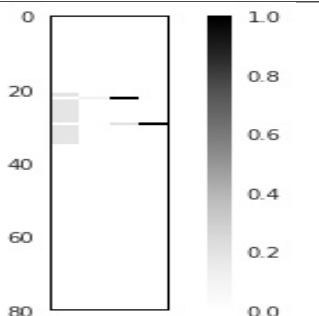
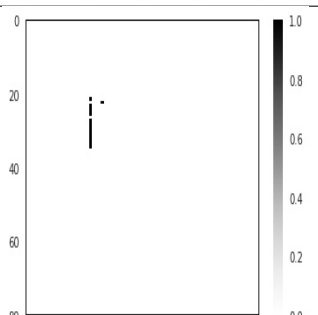
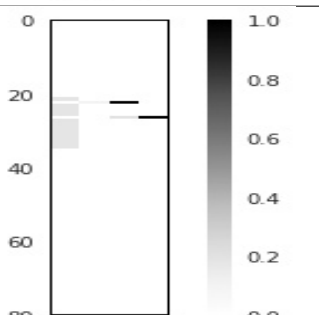


Fig. 15. Graph words embedding and reconstruction

Table 13
Capturing graph words similarity via embedding

ID	Graph word	Embedding vectors U and V
61		
85		
100		
104		

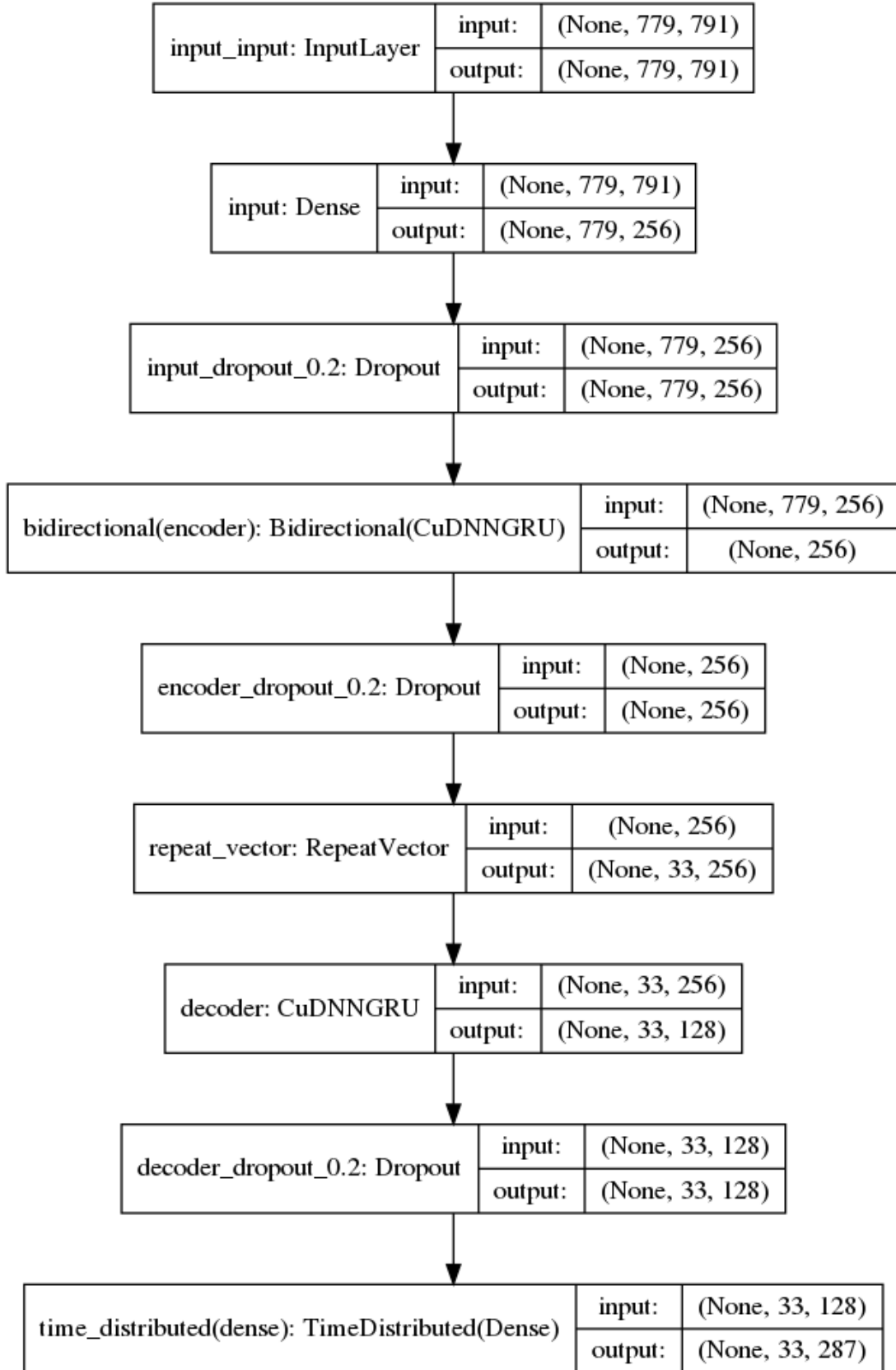


Fig. 16. Graph words translation model for DBpedia

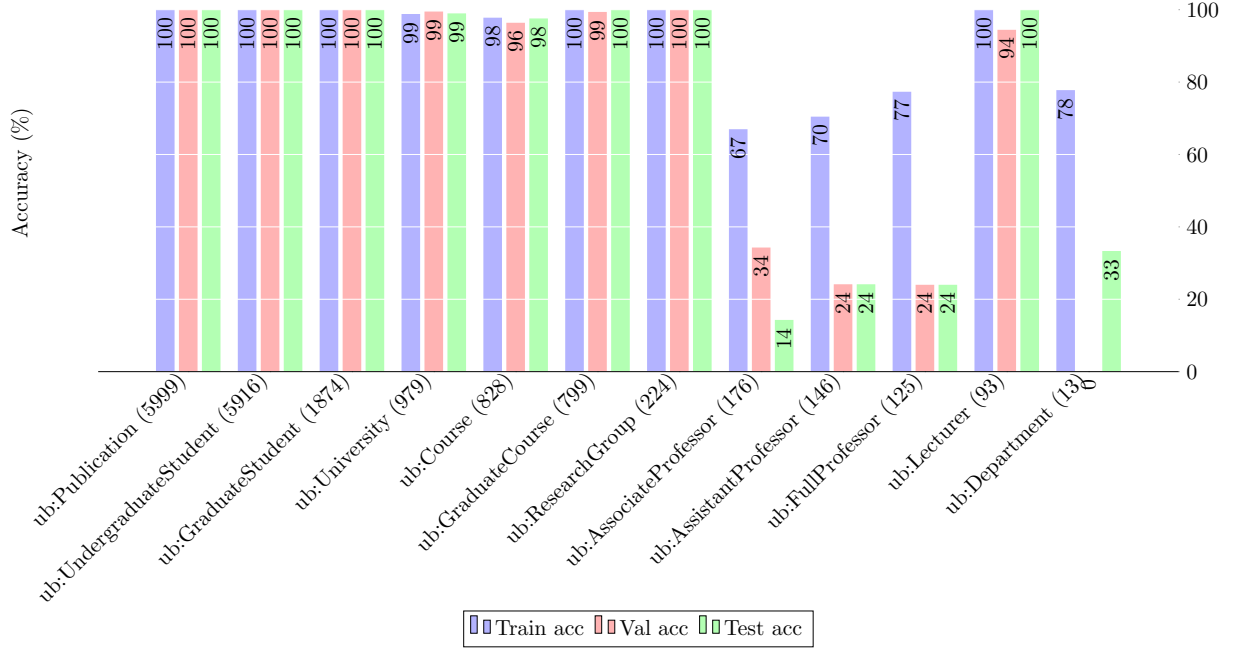


Fig. 17. LUBM1 accuracy results per class

Listing 7: Original graph (LUBM1 description of GraduateStudent114)

```

_:Publication8 ub:publicationAuthor _:GraduateStudent114 .
_:Publication7 ub:publicationAuthor _:GraduateStudent114 .
_:Publication2 ub:publicationAuthor _:GraduateStudent114 .
_:Publication11 ub:publicationAuthor _:GraduateStudent114 .
_:Publication6 ub:publicationAuthor _:GraduateStudent114 .
_:GraduateStudent114 a ub:GraduateStudent,
    ub:TeachingAssistant ;
ub:advisor _:FullProfessor4 ;
ub:emailAddress "GraduateStudent114@Department0.University0.edu" ;
ub:memberOf <http://www.Department0.University0.edu> ;
ub:name "GraduateStudent114" ;
ub:takesCourse _:GraduateCourse16,
    _:GraduateCourse66,
    _:GraduateCourse8 ;
ub:teachingAssistantOf _:Course47 ;
ub:telephone "xxx-xxx-xxxx" ;
ub:undergraduateDegreeFrom <http://www.University989.edu> .

```

Listing 8: Distorted graph (LUBM1 description of GraduateStudent114)

```

_:Publication8 ub:publicationAuthor _:GraduateStudent114 .
_:Publication7 ub:publicationAuthor _:GraduateStudent114 .
_:Publication2 ub:publicationAuthor _:GraduateStudent114 .
_:Publication11 ub:publicationAuthor _:GraduateStudent114 .
_:Publication6 ub:publicationAuthor _:GraduateStudent114 .
_:GraduateStudent114 a ub:University ,
    ub:TeachingAssistant ;
ub:advisor _:FullProfessor4 ;
ub:emailAddress "GraduateStudent114@Department0.University0.edu" ;
ub:memberOf <http://www.Department0.University0.edu> ;
ub:name "GraduateStudent114" ;
ub:takesCourse _:GraduateCourse16,
    _:GraduateCourse66,
    _:GraduateCourse8 ;
ub:teachingAssistantOf _:Course47 ;
ub:telephone "xxx-xxx-xxxx" ;
ub:undergraduateDegreeFrom <http://www.University989.edu> .

```

Listing 9: Deep reasoner wrong inference from noisy input of type UGS

```

_:Publication8 a ub:Publication .
_:Publication7 a ub:Publication .
_:Publication2 a ub:Publication .
_:Publication11 a ub:Publication .
_:Course47 a ub:Organization,
    ub:University .
_:FullProfessor4 a ub:Employee,
    ub:Faculty,
    ub:Professor .
_:Publication6 a ub:Publication .
_:GraduateStudent114 a ub:Person ;
ub:degreeFrom <http://www.University989.edu> .

```

Listing 10: Jena inference from noisy input of type UGS

```

_:Publication4 a ub:Publication .
_:AssistantProfessor3 a ub:Employee,
    ub:Faculty,
    ub:Professor .
_:Publication8 a ub:Publication .
_:Publication7 a ub:Publication .
_:Publication14 a ub:Publication .
_:Publication13 a ub:Publication .
_:GraduateStudent0 a ub:Organization,
    ub:Person ;
ub:degreeFrom <http://www.University358.edu> .
<http://www.University358.edu> a ub:Organization,
    ub:University .

```


Listing 11: GCC distorted graph

```

_:FullProfessor8 ub:teacherOf _:Course12 .
_:GraduateStudent15 ub:teachingAssistantOf _:Course12 .
_:UndergraduateStudent114 ub:takesCourse _:Course12 .
_:UndergraduateStudent132 ub:takesCourse _:Course12 .
_:UndergraduateStudent143 ub:takesCourse _:Course12 .
_:UndergraduateStudent18 ub:takesCourse _:Course12 .
_:UndergraduateStudent199 ub:takesCourse _:Course12 .
_:UndergraduateStudent200 ub:takesCourse _:Course12 .
_:UndergraduateStudent205 ub:takesCourse _:Course12 .
_:UndergraduateStudent21 ub:takesCourse _:Course12 .
_:UndergraduateStudent215 ub:takesCourse _:Course12 .
_:UndergraduateStudent225 ub:takesCourse _:Course12 .
_:UndergraduateStudent317 ub:takesCourse _:Course12 .
_:UndergraduateStudent330 ub:takesCourse _:Course12 .
_:UndergraduateStudent347 ub:takesCourse _:Course12 .
_:UndergraduateStudent348 ub:takesCourse _:Course12 .
_:UndergraduateStudent353 ub:takesCourse _:Course12 .
_:UndergraduateStudent383 ub:takesCourse _:Course12 .
_:UndergraduateStudent406 ub:takesCourse _:Course12 .
_:UndergraduateStudent419 ub:takesCourse _:Course12 .
_:UndergraduateStudent425 ub:takesCourse _:Course12 .
_:UndergraduateStudent465 ub:takesCourse _:Course12 .
_:UndergraduateStudent466 ub:takesCourse _:Course12 .
_:UndergraduateStudent467 ub:takesCourse _:Course12 .
_:UndergraduateStudent475 ub:takesCourse _:Course12 .
_:UndergraduateStudent478 ub:takesCourse _:Course12 .
_:UndergraduateStudent492 ub:takesCourse _:Course12 .
_:UndergraduateStudent520 ub:takesCourse _:Course12 .
_:UndergraduateStudent523 ub:takesCourse _:Course12 .
_:UndergraduateStudent64 ub:takesCourse _:Course12 .
_:Course12 a ub:GraduateCourse ;
    ub:name "Course12" .

```

Listing 12: Deep reasoner inference from noisy input of type GCC

```

_:Course12 a ub:Organization .
_:UndergraduateStudent114 a ub:Person .
_:UndergraduateStudent132 a ub:Person .
_:UndergraduateStudent143 a ub:Person .
_:UndergraduateStudent18 a ub:Person .
_:UndergraduateStudent199 a ub:Person .

```

Listing 13: Jena inference from noisy input of type GCC

```
_:Course12 a ub:Work .  
_:FullProfessor8 a ub:Employee,  
    ub:Faculty .  
_:GraduateStudent15 a ub:TeachingAssistant .
```

References

- [1] 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>.
- [2] A. d’Avila Garcez, T. Besold, L. de Raedt, P. Földiák, P. Hitzler, T. Icard, K.-U. KÄijhnerberger, L. Lamb, R. Mikulainen and D. Silver, Neural-Symbolic Learning and Reasoning: Contributions and Challenges, in: *Knowledge Representation Reasoning: Integrating Symbolic Neural Approaches*, 2015, pp. 18–21. <https://www.aaai.org/ocs/index.php/SSS/SSS15/paper/view/10281>.
- [3] 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.
- [4] 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. <https://doi.org/10.4018/ijswis.2014040104>.
- [5] 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>.
- [6] M. Zhu, Z. Gao and Z. Quan, Noisy Type Assertion Detection in Semantic Datasets, 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. 373–388. doi:10.1007/978-3-319-11964-9_24. https://doi.org/10.1007/978-3-319-11964-9_24.
- [7] 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. https://doi.org/10.1007/978-3-319-11964-9_23.
- [8] 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. https://doi.org/10.1007/978-3-319-07443-6_34.
- [9] 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>.
- [10] 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.
- [11] T.R. Gruber, Toward principles for the design of ontologies used for knowledge sharing?, *Int. J. Hum.-Comput. Stud.* **43**(5–6) (1995), 907–928. doi:10.1006/ijhc.1995.1081. <http://www.sciencedirect.com/science/article/pii/S1071581985710816>.
- [12] M. Uschold and M. King, Towards a methodology for building ontologies, 1995, Workshop Basic Ontological Issues in Knowledge Sharing.
- [13] A. Maedche and S. Staab, Mining Ontologies from Text, in: *Knowledge Eng. Knowledge Manage. Methods, Models, Tools*, Vol. 1937, R. Dieng and O. Corby, eds, Springer Berlin Heidelberg, Berlin, Germany, 2000, pp. 189–202. doi:10.1007/3-540-39967-4_14. https://doi.org/10.1007/3-540-39967-4_14.
- [14] K.S. Metaxiotis, J.E. Psarras and D. Askounis, Building ontologies for production scheduling systems: towards a unified methodology, *Inform. Manage. Computer Security* **9**(1) (2001), 44–51. <https://doi.org/10.1108/09685220110366803>.
- [15] M. Fernández-López and A. Gómez-Pérez, Overview and analysis of methodologies for building ontologies, *Knowledge Eng. Review* **17**(2) (2002), 129–156. doi:10.1017/S0269888902000462. <https://doi.org/10.1017/S0269888902000462>.
- [16] M.A. Casteleiro, M.J.F. Prieto, G. Demetriou, N. Maroto, W.J. Read, D. Maseda-Fernandez, J.J.D. Diz, G. Nenadic, J.A. Keane and R. Stevens, Ontology Learning with Deep Learning: a Case Study on Patient Safety Using PubMed, in: *Proc. 9th Int. Conf. Semantic Web Applications and Tools Life Sciences*, Vol. 1795, A. Paschke, A. Burger, A. Splendiani, M.S. Marshall and P. Romano, eds, CEUR-WS.org, Amsterdam, The Netherlands, 2016, pp. 1–10. <http://ceur-ws.org/Vol-1795/paper12.pdf>.
- [17] 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>.
- [18] T.K. Landauer and S.T. Dumais, A solution to Plato’s problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge., *Psychological Review* **104**(2) (1997), 211.
- [19] D.M. Blei, A.Y. Ng and M.I. Jordan, Latent Dirichlet Allocation, *J. Mach. Learning Research* **3** (2003), 993–1022. <http://www.jmlr.org/papers/v3/blei03a.html>.
- [20] S. Albukhitani, T. Helmy and A. Al-Nazer, Arabic ontology learning using deep learning, in: *Proc. Int. Conf. Web Intelligence*, A.P. Sheth, A. Ngonga, Y. Wang, E. Chang, D. Slezak, B. Franczyk, R. Alt, X. Tao and R. Unland, eds, ACM, New York, NY, USA, 2017, pp. 1138–1142. doi:10.1145/3106426.3109052. <http://doi.acm.org/10.1145/3106426.3109052>.
- [21] P. Hohenecker and T. Lukasiewicz, Deep Learning for Ontology Reasoning, Vol. abs/1705.10342, 2017. <http://arxiv.org/abs/1705.10342>.
- [22] 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>.

- [23] J. Lehmann, S. Bader and P. Hitzler, Extracting reduced logic programs from artificial neural networks, *Appl. Intell.* **32**(3) (2010), 249–266. doi:10.1007/s10489-008-0142-y. <https://doi.org/10.1007/s10489-008-0142-y>.
- [24] 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.
- [25] I. Niles and A. Pease, Towards a Standard Upper Ontology, in: *Proc. Int. Conf. Formal Ontology in Inform. Syst. - Volume 2001*, ACM, New York, NY, USA, 2001, pp. 2–9. doi:10.1145/505168.505170. <http://doi.acm.org/10.1145/505168.505170>.
- [26] 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. <http://doi.acm.org/10.1145/1013367.1013381>.
- [27] 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.
- [28] A. Zaveri, D. Kontokostas, M.A. Sherif, L. Böhmann, 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. <http://doi.acm.org/10.1145/2506182.2506195>.
- [29] 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/>.
- [30] 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. <https://doi.org/10.1016/j.websem.2005.06.005>.
- [31] S. Projects, SWAT Projects - the Lehigh University Benchmark (LUBM), 2002. <http://swat.cse.lehigh.edu/projects/lubm/>.
- [32] 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. <https://doi.org/10.4018/jswis.2009081901>.
- [33] M.L. Richard Cyganiak David Wood, RDF 1.1 Concepts and Abstract Syntax, W3C, W3C, 2014. <https://www.w3.org/TR/rdf11-concepts/>.
- [34] J.A. Bondy, *Graph Theory With Applications*, Elsevier Sci. Ltd., Oxford, UK, 1976.
- [35] 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. Kashyap, S. Decker and R. Eckstein, eds, CEUR-WS.org, Aachen, Germany, 2003, pp. 279–293. <http://dl.acm.org/citation.cfm?id=2889905.2889924>.
- [36] 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>.
- [37] 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. <https://doi.org/10.1016/j.websem.2014.06.004>.
- [38] 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. https://doi.org/10.1007/978-3-540-30475-3_5.
- [39] E.W. Weisstein, Hypergraph. From MathWorld—A Wolfram Web Resource, 2018. <http://mathworld.wolfram.com/Hypergraph.html>.
- [40] J. Hayes, A graph model for RDF, Technical Report, Darmstadt University of Technology, 2004.
- [41] 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. <https://doi.org/10.1109/ICMLA.2013.31>.
- [42] 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. <https://doi.org/10.1007/s11390-009-9265-9>.
- [43] 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.
- [44] A. Basu and R.W. Blanning, *Metagraphs and Their Applications (Integrated Series in Information Systems)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [45] DBpedia, About: James Hendler, 2016. http://dbpedia.org/page/James_Hendler.
- [46] DBpedia, About: Yoshua Bengio, 2016. http://dbpedia.org/page/Yoshua_Bengio.
- [47] 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.
- [48] 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. <http://doi.acm.org/10.1145/2939672.2939751>.
- [49] M. Schuster and K.K. Paliwal, Bidirectional recurrent neural networks, *IEEE Trans. Signal Process* **45**(11) (1997), 2673–2681. doi:10.1109/78.650093.
- [50] A. Graves and J. Schmidhuber, Framewise phoneme classification with bidirectional LSTM and other neural network architectures, *Neural Networks* **18**(5–6)

- (2005), 602–610. doi:10.1016/j.neunet.2005.06.042. <https://doi.org/10.1016/j.neunet.2005.06.042>.
- [51] F. Chollet et al., Keras, GitHub, 2015. <https://github.com/keras-team/keras>.
- [52] 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>.
- [53] 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>.
- [54] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro and E. Shelhamer, cuDNN: Efficient Primitives for Deep Learning, Vol. abs/1410.0759, 2014. <http://arxiv.org/abs/1410.0759>.
- [55] B. Bishop, A. Kiryakov, D. Ognyanoff, I. Peikov, Z. Tashev and R. Velkov, OWLIM: A family of scalable semantic repositories, *Semantic Web* 2(1) (2011), 33–42. doi:10.3233/SW-2011-0026. <https://doi.org/10.3233/SW-2011-0026>.
- [56] 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.
- [57] I.J. Goodfellow, J. Shlens and C. Szegedy, Explaining and Harnessing Adversarial Examples, Vol. abs/1412.6572, 2014. <http://arxiv.org/abs/1412.6572>.
- [58] 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>.
- [59] 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. <http://dx.doi.org/10.1093/nar/gku1011>.
- [60] 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. <https://doi.org/10.1109/TKDE.2009.191>.
- [61] 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, HybridSem-Stats*, 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>.