Reason-able Embeddings: Learning Concept Embeddings with a Transferable Neural Reasoner

Dariusz Max Adamski [a] and Jędrzej Potoniec [a,*]
[a] *Institute of Computing Science, Poznan University of Technology, Poznan, Poland*
E-mails: maxadamski98@gmail.com, jedrzej.potoniec@cs.put.poznan.pl

Abstract. We present a novel approach for learning embeddings of $\mathcal{ALC}$ knowledge base concepts. The embeddings reflect the semantics of the concepts in such a way that it is possible to compute an embedding of a complex concept from the embeddings of its parts by using appropriate neural constructors. Embeddings for different knowledge bases are vectors in a shared vector space, shaped in such a way that approximate subsumption checking for arbitrarily complex concepts can be done by the same neural network, called a reasoner head, for all the knowledge bases. To underline this unique property of enabling reasoning directly on embeddings, we call them reason-able embeddings. We report the results of experimental evaluation showing that the difference in reasoning performance between training a separate reasoner head for each ontology and using a shared reasoner head, is negligible.

Keywords: neural-symbolic integration, deep deductive reasoning, embeddings, transfer learning, deep learning

1. Introduction

In this paper we design, implement and evaluate a novel method of learning concept embeddings in knowledge bases for the $\mathcal{ALC}$ description logic, using a transferable neural reasoner. Our method of learning embeddings ensures that the resulting embeddings are entirely data-driven, which unlike manually-designed concept vectorization schemes, captures as many useful properties of the given training data as possible. We call the resulting embeddings *reason-able embeddings*, as they are explicitly constructed in such a way as to maintain the formal semantics of the represented concepts and thus enable approximate reasoning with them.

The neural reasoner consists of two modules – a reasoner head, that is a neural network classifier, trained to classify whether subsumption axioms hold for a given knowledge base, and capable of constructing an embedding for arbitrarily complex $\mathcal{ALC}$ concepts, and an embedding layer responsible for mapping concept names to the vector space. The reasoner head is transferable, because the embedding layers learn how to embed concepts in a space that minimizes the classifier loss, and is shared between all knowledge bases. It must be pointed out that the reasoner head does not explicitly use terminological axioms for entailment classification. Instead, the terminological axioms indirectly shape the learned concept embeddings, so even though it may look like the reasoner head is simply memorizing answers, some form of deductive reasoning is actually performed.

In our work we hypothesize, and experimentally show support for the idea that **concepts from different knowledge bases can be represented in a shared embedding space, with a topology that lends itself for approximate**

*Corresponding author. E-mail: jedrzej.potoniec@cs.put.poznan.pl.

reasoning by entailment classifiers based on neural networks. This research is a curiosity-driven intellectual endeavor: to the best of our knowledge, this is the first attempt to force a kind of shared semantics on seemingly separate sets of embeddings, yet without defining the semantics explicitly. While we imagine that there may be practical applications of the proposed idea, such as hot-swapping ontologies in a neural-symbolic system, we do not make any claims about its immediate usefulness.

We purposefully limit ourselves to the taxonomical part of $\mathcal{ALC}$ knowledge bases, and assume the assertional part to be empty. We believe that accounting for a fundamental difference between an individual, and a set of individuals, is a separate difficulty in and of itself. At the same time, solving this difficulty is not necessary for the method to be useful, as many ontologies employ a modeling style such that there are no individuals or they are used very sparingly [1].

The main contributions of the work are as follows:

- A novel architecture consisting of two parts: a reasoner head, employing a recursive neural network, and a collection of embedding layers. Together, they enable the construction of reason-able embeddings.
- A training procedure employing multiple knowledge bases at once to shape the topology of the embedding space.
- Extensive experimental evaluation aimed at establishing the properties of the proposed approach. In the experiments, we show that the proposed approach indeed forces embeddings to represent the formal semantics in a shared way, that the resulting embeddings are of high quality, and that using this kind of shared, latent semantics does not incur substantial penalty on them.

On top of that, we present three open-source technical contributions implemented in Python that may be useful for the community:

- An implementation of the presented method
- A random axiom generator for the $\mathcal{ALC}$ description logic
- An extended interface for the FaCT++ reasoner

Our work is structured as follows. In section 2, we briefly introduce deep learning (subsection 2.1) and description logics (subsection 2.2), along with the related notation that we use throughout this text. In section 3, we give an overview of the current state of the art, in particular of knowledge base embeddings (subsection 3.1), and of deep deductive reasoners (subsection 3.2). Section 4 presents the details of the architecture for learning reason-able embeddings: in subsection 4.1 we present the details of the reasoner head, and in subsection 4.2 of the embedding layer. We then introduce a relaxed variant of the architecture in subsection 4.3, and describe the training procedure in subsection 4.4. The axiom generator for $\mathcal{ALC}$ is described in section 5. We report the results of the experimental evaluation in section 6, where we begin by formulating the research questions guiding the experiments in subsection 6.1. We describe the setup in subsection 6.2, the employed metrics in subsection 6.3, and the used data sets in subsection 6.4. Subsequently, we summarize the results of four experiments in sections 6.5–6.8. Lastly, in section 7 we summarize the results of our work and point out potential improvements, extensions, and applications of our neural reasoner and concept embeddings.

2. Preliminaries

2.1. Deep learning

While machine learning is ubiquitous in research these days, we give a very brief introduction to ensure there is no confusion in terms. In this, we follow [2], and the reader is welcome to the book for a more extended introduction.

Classification is a task where a learning algorithm is supposed to produce a function, called a *classifier*, mapping an object to a label from a discrete, finite set of labels. A special case, used in this paper, is *binary classification*, where only two mutually-exclusive labels are considered: 0 (negative) and 1 (positive). Objects are usually represented by vectors of numerical *features*, albeit different representations are possible. By convention, features are usually denoted by x, and predicted labels are denoted by $\hat{y}$.

Binary classification is frequently solved by constructing a function predicting the probability of an object to belong to the positive class, and then thresholding on the probability to obtain the label. Throughout this work, we employ this approach, writing $P(x)$ to denote the probability that the object represented by x should be labeled with the positive label.

A classifier usually contains a number of *parameters*, which are automatically adjusted during the *training*, a process aiming to optimize the value of some *metric* on a *training set*. The training set consists of examples, each example being an object assigned a true label that the classifier is supposed to predict. By convention, the true labels are usually denoted as y. The details of the optimization process can vary wildly, but in the context of this work, as the metric we use *binary cross-entropy* and employ a variant of *gradient descent* optimization algorithm called AdamW [3] to minimize it.

For gradient descent algorithms, it is established to use a *mini-batch training*, where the training set is randomly split into small, disjoint subsets called *mini-batches*, and each mini-batch is used to update the parameters of the model separately. A complete pass over all the mini-batches from a training set is called an *epoch*. In some special cases, only a subset of parameters is optimized at a given time, while the remaining are called *frozen* and kept constant.

Due to the stochastic nature of training, too aggressively optimizing the metric can lead to *overfitting*, a phenomenon in which the classifier labels the training examples very well, but fares poorly on new, unseen data. To evaluate the performance of a classifier on unseen data, a separate set, called *test set*, is maintained. It is assumed that both sets are *independent and identically distributed* (the i.i.d. assumption), i.e., the examples in them are drawn independently of each other from the same probability distribution. Usually, for the evaluation, different metrics are used than during training. The details of the metrics used in this paper are given in subsection 6.3.

It is frequently the case that a learning algorithm has some configuration options that cannot be automatically adjusted using the training set, e.g., because that would immediately lead to overfitting. These options are called *hyperparameters*, and are adjusted by observing the value of a metric of choice on yet another set, the *validation set*. It follows the same i.i.d. assumption as the two other sets.

A *feed-forward neural network* is a sequence of *linear transformations* and *activation functions*, where each transformation is followed by an activation function. By convention, each pair: a linear transformation and an activation function is called a *layer*. The linear transformation is of form $Ax + b$, where A is a matrix of the type $n \times m$, b is a vector of the type n, and x is a vector of the type m. A and b are parameters, called *weights*, while x is an input. By convention, m is called the *number of neurons* of the layer. Throughout the paper, we denote a single layer with L^f_m, where m is the number of neurons and f is the activation function: $L^f_m(x) = f(Ax + b)$. In a special case where there is no activation function in a layer, we replace f with $\cdot$. In this convention, a feed-forward neural network of two layers, the first consisting of 16 neurons and employing the activation function σ and the second of 1 neuron and employing no activation function, can be represented as: $L^{\cdot}_1 \left(L^{\sigma}_{16}(x)\right)$. Traditionally the last layer is called the *output layer*, and the remaining layers – *hidden layers*.

Over the years, many activation functions were considered. Throughout the paper we use two of them: the sigmoid σ, given in Equation 1, and ELU in Equation 2 [4].

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

$$ELU(x) = \begin{cases} e^x - 1 & x < 0 \\ x & x \geqslant 0 \end{cases} \tag{2}$$

Finally, by an *embedding* we understand a real-valued vector representing some concept from a non-numeric set, such that the size of the vector is much smaller than the cardinality of the set, and that the dimensions are not directly interpretable, i.e., the vector comes from a latent space.

2.2. Description logics

Description logics (DLs) are a family of logics, that are widely used as a formal way to represent knowledge in the form of knowledge bases (KBs). An advantageous property of DLs is that reasoning in most of them is decidable, since they are fragments of first-order predicate logic [5]. The formalism of DLs has been used as a basis for the Web Ontology Language (OWL) standard, which as part of the Semantic Web is responsible for describing the semantics, or meaning, of data on the Internet [6]. DLs and the foundational technologies of the Semantic Web are considered mature, and are used both in research and the industry [7]. An illustrative example of the potential applications of DLs is SNOMED CT – a knowledge base that is considered to be the most comprehensive multilingual clinical healthcare terminology in the world [8].

As mentioned, DLs are a family of logics, which means that there are many DLs of varying expressiveness. Since in our work we focus on the $\mathcal{ALC}$ DL (abbreviated from attributive language with complements), we also introduce DLs from the perspective of $\mathcal{ALC}$. There exist DLs that are simpler than $\mathcal{ALC}$, for example $\mathcal{EL}$, and there are also vastly more expressive DLs like $\mathcal{SHOIN}^{(\mathcal{D})}$ and $\mathcal{SROIQ}^{(\mathcal{D})}$, which are the basis of OWL DL and OWL 2 DL respectively. We do not deal with $\mathcal{EL}$ or $\mathcal{SROIQ}^{(\mathcal{D})}$, but we do discuss a procedure for transforming $\mathcal{SHOIN}^{(\mathcal{D})}$ knowledge bases to $\mathcal{ALC}$ knowledge bases in section 6.4.2, by removing parts that $\mathcal{ALC}$ does not support.

For readers that are unfamiliar with DLs we begin with an intuitive overview. In later sections we describe the notation we use, and define the syntax and semantics of the $\mathcal{ALC}$ DL. The introduction to DLs is based on [9].

2.2.1. Overview

At a high level, a KB divides knowledge into *terminology* (also called TBox) and *assertions* (also called ABox). One may think that the terminology describes general knowledge (e.g. "Dogs are mammals"), and assertions state facts (e.g. "Fido is a dog" or "Fido likes Alice"). Those elements of KB terminology and assertions are called *axioms*. Axioms are formulated using a *vocabulary*, which consists of *individuals* (e.g. "Fido" or "Alice"), *concept names* (e.g. "dog" or "mammal"), and *role names* (e.g. "likes"). A *concept* denotes a set of individuals, and besides concept names in the vocabulary, there are *complex concepts* that can be constructed from other concepts using *concept constructors*. The available constructors are different depending on the chosen DL. In particular, in the $\mathcal{ALC}$ DL there are concept complement, intersection, union, universal restriction, and existential restriction constructors.

The TBox contains *subsumption axioms*, that allow one to state that one concept is subsumed by another concept. There are two kinds of assertion axioms in ABox: a *concept assertion axiom* allows one to state that an individual is an instance of a concept, and a *role assertion axiom* allows one to relate two individuals by a binary role.

Semantics of DLs are defined in a model-theoretic way, by providing an *interpretation* that represents the KB vocabulary in terms of a set called the *domain*. In particular, an interpretation maps individuals to elements of the domain, concepts to sets of individuals, and roles to binary relations between individuals. There are also special concepts, called the *bottom concept* and *top concept*, which correspond to the empty set and the domain, respectively.

The semantics of DLs provide an *entailment* relation, so that given a set of axioms, logical consequences may be inferred. In other words, KB entails a given axiom if that axiom follows from KB's axioms.

2.2.2. Notation

Formally, a KB in the $\mathcal{ALC}$ DL is a pair $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is the set of terminological axioms and $\mathcal{A}$ is the set of assertion axioms. The vocabulary of a KB is defined as a triple (N_C, N_R, N_I), where N_C is the set of concept names, N_R is the set of role names, and N_I is the set of individual names. Given an arbitrary ordering of the set of concept names N_C and an arbitrary ordering of the set of role names N_R, we define A_i as the i-th concept name, and R_i as the i-th role name, so that $A_i \in N_C$ and $R_i \in N_R$. In text, we write individuals and concept names in `PascalCase`, and role names in `camelCase`. In this work we do not deal with assertion axioms and henceforth assume that $\mathcal{A} = \emptyset$.

2.2.3. Semantics

An interpretation is defined as a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a set called the domain, and $\cdot^{\mathcal{I}}$ is an interpretation function. The interpretation function $\cdot^{\mathcal{I}}$ maps each concept C to a subset of $\Delta^{\mathcal{I}}$, each role R to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each individual to an element of $\Delta^{\mathcal{I}}$. The syntax and semantics in $\mathcal{ALC}$ is summarized in Table 1.

The set of terminological axioms $\mathcal{T}$ contains subsumption axioms $C \sqsubseteq D$, that are read C is subsumed by D, where C and D are concepts. If, for an interpretation $\mathcal{I}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ then we say that the axiom $C \sqsubseteq D$ holds in the

Table 1
Syntax and semantics of $\mathcal{ALC}$

Description	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
complement	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
existential restriction	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists_{y \in \Delta^{\mathcal{I}}}((x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}})\}$
universal restriction	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall_{y \in \Delta^{\mathcal{I}}}((x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}})\}$

interpretation, denoted $\mathcal{I} \models C \sqsubseteq D$, and call the interpretation *a model* of the axiom. Sometimes $\mathcal{T}$ also contains *equivalence axioms* $C \equiv D$, although such axioms can be expressed in terms of two subsumption axioms $C \sqsubseteq D$ and $D \sqsubseteq C$. If an interpretation is a model for all the axioms in $\mathcal{K}$, we write $\mathcal{I} \models \mathcal{K}$.

The notion can be extended to arbitrary axioms α. We say that α is a logical consequence of $\mathcal{K}$, denoted $\mathcal{K} \models \alpha$, if for every model $\mathcal{I}$ of $\mathcal{K}$, the entailment $\mathcal{I} \models \alpha$ also holds.

2.2.4. Ontologies in the Web Ontology Language

While the Web Ontology Language (OWL) is underpinned by a DL, it uses a different nomenclature. In particular, knowledge bases are called *ontologies* in OWL. We also use the term ontology, but only when referring to a KB described in OWL. Concept names and role names are called *class names* and *object property names*, respectively. In general, concepts are referred to as *classes* or *class expressions*.

Terminological axioms are called class axioms, and a subsumption $C \sqsubseteq D$ is read "C is a subclass of D". Although we did not discuss role axioms as $\mathcal{ALC}$ does not support them, OWL ontologies do support object property axioms, which are the same thing. In addition to class, object property, and individual axioms, OWL ontologies can also contain annotation properties and annotation axioms, that allow one to provide additional information about classes. For example, one very common annotation property is `rdfs:label`, which is used to provide human-readable labels.

2.2.5. Semantic reasoners

The main benefit of using a knowledge representation paradigm with well-founded semantics is the ability to perform automated reasoning. In this work, we use use the *entailment checking* reasoning task, i.e., deciding whether $\mathcal{K} \models \alpha$ for a given axiom α.

Many *semantic reasoners* for DLs are available, providing reasoning services for KBs, among others, HermiT [10], Pellet [11], and FaCT++ [12] capable of reasoning in very expressive DLs, or more specialized ones such as ELK [13].

In this work, FaCT++ is the reasoner of choice, as it is capable enough to deal with $\mathcal{ALC}$ knowledge bases, yet it is written in C++, which means that it is easy and efficient to interface with it from Python via a Cython extension. That was not the case for the more popular reasoners implemented directly on the Java Virtual Machine, where the cost of interaction was prohibitive.

3. Related work

The neural and symbolic paradigms of artificial intelligence are vastly different. On one hand, the currently dominant neural paradigm shows how well neural networks perform on large-scale data sets, ranging from simple classification and regression, to language models so powerful, that their output is almost indistinguishable from human-written text [14], and generative image models that can synthesize photorealistic images from text prompts [15]. However, neural models often produce nonsensical results, for example paragraphs of text that contradict themselves. Because neural models are not easily interpretable, it is difficult to find the cause of such problems. On the other hand, the symbolic paradigm offers methods for explicitly describing knowledge, and reliably performing rea-

soning over that knowledge. Symbolic methods can provide step-by-step explanations of their inferences, because they use deductive reasoning. Unfortunately, symbolic methods are not well suited for learning from data, as real-life knowledge is often seemingly contradictory. Symbolic methods also have trouble with processing large-scale data sets, because of high time complexities of used algorithms. The research field of neuro-symbolic integration aims to combine the large-scale learning ability of neural models, and the ability of symbolic methods to express knowledge and perform reasoning, all while keeping interpretability, thus combining the benefits and avoiding the pitfalls of both paradigms [16]. There seems to be two major avenues where the Semantic Web can benefit from the neural paradigm: ontology and knowledge graphs embeddings and deep deductive reasoners [17].

3.1. Knowledge base embeddings

One way to bridge the neural and symbolic paradigms is to represent symbolic knowledge in terms of vectors in a high-dimensional real vector space. Such vectors can then be used as additional inputs to machine learning models based on neural networks, to improve their performance by allowing them to use an approximation of expert knowledge [18]. A good method of learning embeddings from symbolic knowledge should ideally leverage the structural information present in relations between abstract concepts, and should not try to learn embeddings for abstract concepts with the aid of word embeddings, due to the ambiguity of language, its limited abstraction, and other problems [19].

One of the first embeddings methods tailored to KBs to get traction were NTN proposed by Socher et al. [20], TransE proposed by Bordes et al. [21, 22], later unified into a single framework by Yang et al. [23], as both leverage linear and bilinear mappings. The notion of modeling relations as operations in some vector space was further extended by Wang et al. with TransH [24], Lin et al. with TransR [25] and PTransE [26], Sun et al. with RotatE [27], Zhang et al. with HakE [28], Wang et al. with InterHT [29], or Zhou et al. with Path-RotatE [30].

A separate category of embeddings was established by ConvE, employing convolutional neural networks [31]. This line of research was continued by Nguyen et al. with ConvKB [32], Jiang et al. with ConvR [33], Demir et al. with ConEx [34]. Over the time even more complex neural architectures were employed, e.g., capsule networks in CapsE by Nguyen et al. [35], recurrent skipping networks by Guo et al. [36], graph convolutional networks in GCN-Align by Wang et al. [37] or R-GCN by Schlichtkrull et al. [38], recurrent transformers by Werner et al. [39]

Yet another approach was taken in RESCAL by Nickel et al. where tensor-based techniques were used [40]. Similar approaches were taken in TOAST by Jachnik et al. [41], TATEC by García-Durán et al. [42], DistMult by Yang et al. [43], HolE by Nickel et al. [44], ComplEx by Trouillon et al. [45], or ANALOGY by Liu et al. [46].

In the context of the Semantic Web technologies, notable examples of embeddings are RDF2Vec by Ristoski et al. [47], OWL2Vec* by Chen et al. [48], and TransOWL by d'Amato et al. [49].

Embeddings are used in various downstream tasks, such as disambiguation [50], ontology learning [51], handling concept drift in streams [52], question answering [53] or fact classification [54]. Over the years multiple survey papers were published on the topic of KB embeddings [55–59].

To the best of our knowledge, none of the proposed methods aims to capture the semantics as a topology of the embedding space, nor enables computing the embeddings of complex expressions from the embeddings of their parts.

3.2. Deep deductive reasoners

Compared to the work on embeddings, much less attention has been devoted to the notion of approximating or emulating the results of deductive reasoning with machine learning, and in particular deep learning [16].

Earlier works, employing traditional machine learning approaches, were mostly devoted to approximate a deductive reasoner in order to obtain the results faster than from a deductive reasoner. For example, Rizzo et al. [60, 61] proposed terminological trees and forests to tackle this problem, while Paulheim and Stuckenschmidt concentrated on the problem of approximate reasoning with ABox [62].

One of the first works leveraging deep learning for deductive reasoning over ontologies were those of Makni and Hendler on using recurrent neural networks for RDFS reasoning [63], and of Hohenecker and Lukasiewicz on using recursive reasoning networks [64]. Eberhart et al. employed long short-term memory networks (LSTMs) for

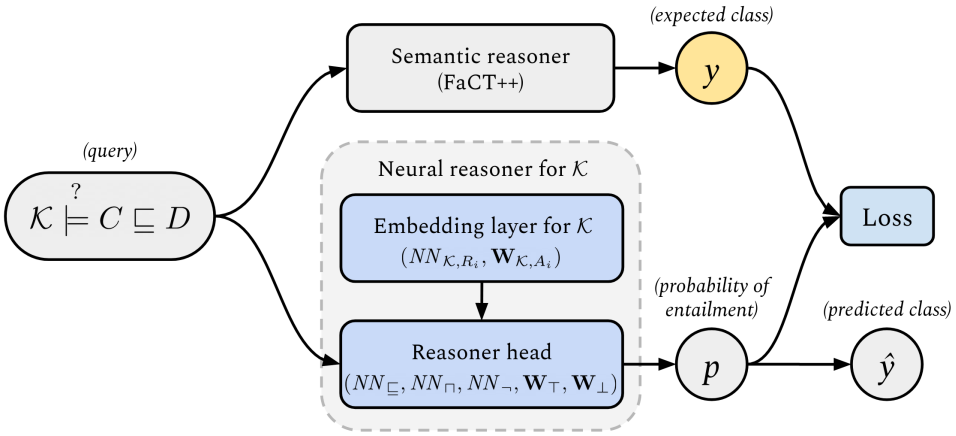

Fig. 1. A high-level overview of the neural reasoner architecture. An embedding layer for knowledge base $\mathcal{K}$ and a reasoner head are combined to create a neural reasoner for $\mathcal{K}$. The reasoner predicts whether $\mathcal{K}$ entails subsumption axioms, written $\mathcal{K} \models C \sqsubseteq D$, where C and D are $\mathcal{ALC}$ concepts. We train the reasoner by contrasting the predicted probability of entailment with the answer provided by a semantic reasoner. The reasoner head generalizes across all $\mathcal{ALC}$ knowledge bases, but for each $\mathcal{K}$ we have to train a separate embedding layer. The reasoner head may be frozen while training an embedding layer for a new knowledge base $\mathcal{K}$, to make learning faster.

reasoning in the $\mathcal{EL}^+$ description logic [65, 66]. Badreddine et al. introduced Logic Tensor Networks [67], which were then employed by Bianchi and Hitzler to perform deductive reasoning [68]. Ebrahimi et al. investigated the applicability of pointer networks to the task [69] and of memory networks [70]. Zhu et al. cast the problem of reasoning as a graph-to-graph transformation problem and applied it in the context of RDF [71].

Ebrahimi et al. in [69] introduced a few dimensions along which a reasoner can be classified: *logic* – which logical formalism is tackled; *transfer* – the capability of achieving good performance on a previously unknown KB; *generative* – the capability of generating new inferences, not only answering yes/no queries; *scale* – the size of considered KBs; *performance* – *high* if the system was able to cross the 70% threshold on the F1-score, *low* otherwise. Following these dimensions, the reasoner presented in this work tackles the $\mathcal{ALC}$ logic, is transferable, is not generative, is of low to moderate scale, and has high performance. To the best of our knowledge, this is the first transferable reasoner capable of tackling such a complex logic, albeit we must underline that the embeddings for a new KB must be trained under the supervision of the reasoner.

4. Neural reasoner

Our neural reasoner (hereafter, *reasoner*) is a classifier, which given a subsumption axiom, outputs the probability that the axiom is entailed by a given knowledge base. The reasoner consists of the generic *reasoner head*, and an interchangeable *embedding layer* specific to a given knowledge base. The generic reasoner head can classify entailment, and construct embeddings for concept complements and intersection of concepts. While an embedding layer contains embeddings for a given knowledge base – that is, it stores embedding vectors for concept names and can construct embeddings for existential restrictions for a given role name. A diagram of the architecture of our neural reasoner is shown in Figure 1.

The main purpose of the classifier is to facilitate the computation of the gradient of the loss function with respect to the weights of the embedding layer. In other words, when the reasoner is given an axiom, it builds its representation bottom-up from the KB-specific embeddings in the embedding layer. That representation is then used by a neural network in the reasoner head to classify whether the axiom is entailed by that KB. The classification output is contrasted with the target output computed by a semantic reasoner, and the value of the loss function is used to adjust weights of the embedding layer with backpropagation through structure [72].

4.1. Reasoner head

In our work we chose the reasoner to be an entailment classifier for subsumption axioms in $\mathcal{ALC}$ – that is, given an axiom $C \sqsubseteq D$, the reasoner head outputs the probability $P(\mathcal{K} \models C \sqsubseteq D)$ that the axiom is entailed by a given knowledge base $\mathcal{K}$. Assume that $h_{\mathcal{K}}$ is a function mapping $\mathcal{ALC}$ concepts in knowledge base $\mathcal{K}$ to embedding vectors in real vector space $\mathbb{R}^{N_e}$, where N_e is the embedding dimension. The details of how this function is constructed are given in subsection 4.2.

An *interaction map* for a pair of embeddings u, v, denoted $IM(u, v)$, as defined by Equation 3, is a concatenation of u, v, and their outer product[1] $u \otimes v$. Before concatenation, the outer product is reshaped to a row vector. Our preliminary experimentation indicated that including the outer product in the interaction map is of utmost importance, as this helps the classifier to learn high-order correlations [73, 74]. By extension, an interaction map for a pair of concepts C, D in knowledge base $\mathcal{K}$ is the interaction map for their embeddings: $IM_{\mathcal{K}}(C, D) = IM(h_{\mathcal{K}}(C), h_{\mathcal{K}}(D))$.

$$IM(u, v) = [u; v; \text{flatten}(u \otimes v)] \tag{3}$$

The classification output for an entailment query $P(\mathcal{K} \models C \sqsubseteq D) \in (0, 1)$ (see Equation 4) is defined as the output of the feedforward neural network $NN_{\sqsubseteq}$, which accepts the interaction map of C and D, and has a single output neuron with the sigmoid activation function σ and a single hidden layer with 16 neurons, which is followed by the ELU activation function [4].

$$P(\mathcal{K} \models C \sqsubseteq D) = NN_{\sqsubseteq}(IM_{\mathcal{K}}(C, D)) = L_1^{\sigma}(L_{16}^{ELU}(IM_{\mathcal{K}}(C, D))) \tag{4}$$

4.2. Embedding layer

Recursive neural networks have been successfully used for encoding expression trees as fixed-size vectors, that could be used as inputs to machine learning models [72]. The recursively defined DL concepts also have a tree structure [75], so it is appropriate to use a recursive neural network as the embedding layer in our reasoner architecture. The key hypothesis that defines our reasoner is that for each KB, we can train an embedding layer that embeds concepts from that KB in an embedding space with a topology that makes it easy for the reasoner head to classify entailment. An embedding topology that is beneficial to entailment classification is formed by jointly training the reasoner head and embedding layers for multiple KBs, which forces the embedding head to generalize, and the embedding layers to output embeddings in the shared embedding space.

We note in passing that we distinguish between a recursive and a recurrent neural network: in a recursive neural network, different parts of the network are applied to the input multiple times, depending on a structured input, as with recursive grammars, where the same production rule can be applied again and again; in a recurrent neural network (e.g., GRU, LSTM) the output of the network is connected back to the network, as part of its input.

In general, concept embeddings are represented by vectors in real vector space $\mathbb{R}^{N_e}$, where N_e is the embedding dimension. In Equations 5–11 we define the recursive function $h_{\mathcal{K}} : \mathcal{ALC} \to \mathbb{R}^{N_e}$ that maps $\mathcal{ALC}$ concepts for knowledge base $\mathcal{K}$ to embedding vectors in real vector space $\mathbb{R}^{N_e}$. We begin with the embedding for each concept name A_i: a vector $\mathbf{W}_{\mathcal{K},A_i}$ of dimension N_e, as given by Equation 5. The vector is optimized during training and it is KB-specific, since different KBs have a different number named concepts with different meanings.

$$h_{\mathcal{K}}(A_i) = \mathbf{W}_{\mathcal{K},A_i} \tag{5}$$

The embeddings of the bottom and top concepts are stored by the vectors $\mathbf{W}_{\bot}$ and $\mathbf{W}_{\top}$, respectively, as specified by Equation 6. Since the semantics of the top and bottom concepts are independent of any specific KB, the vectors are shared between all KBs.

[1] Each element of the outer product of two vectors $(\mathbf{u} \otimes \mathbf{v})_{ij}$ is defined as $\mathbf{u}_i \cdot \mathbf{v}_j$.

$$h_{\mathcal{K}}(\top) = \mathbf{W}_{\top} \qquad h_{\mathcal{K}}(\bot) = \mathbf{W}_{\bot} \tag{6}$$

To obtain the embedding for the complement of a given concept, one first recursively obtains the embedding of that concept, and then passes it as an input to the *complement constructor network* $NN_{\neg}$, consisting of a single layer of N_e neurons without the activation function, as defined in Equation 7.

$$h_{\mathcal{K}}(\neg C) = NN_{\neg}(h_{\mathcal{K}}(C)) \qquad \text{where } NN_{\neg} = L_{N_e}^{\cdot} \tag{7}$$

We also define in Equation 8 the embedding of a doubly negated concept as the embedding of that concept, by the double negation elimination rule, which speeds the construction of embeddings.

$$h_{\mathcal{K}}(\neg\neg C) = h_{\mathcal{K}}(C) \tag{8}$$

To obtain the embedding for an intersection of concepts, one first recursively obtains the embeddings of the two concepts, computes their interaction map, and passes it as the input to the *intersection constructor network* $NN_{\sqcap}$. It is a single layer of N_e neurons without the activation function, as specified by Equation 9, that maps the interaction map of two concept embeddings, to the embedding of their intersection. Its input is a vector of the size of the interaction map $2N_e + N_e^2$, and the output is a vector of size N_e.

$$h_{\mathcal{K}}(C \sqcap D) = NN_{\sqcap}(IM_{\mathcal{K}}(C, D)) \qquad \text{where } NN_{\sqcap} = L_{N_e}^{\cdot} \tag{9}$$

To obtain the embedding for an existential restriction, one first obtains the embedding of the given concept, and passes it to the *existential restriction constructor network*, defined as $NN_{\mathcal{K},R_i} = L_{N_e}^{\cdot}$, i.e., consisting of a single layer of N_e neurons without the activation function. To understand the intuition behind this choice, consider the following example. On the left side of Figure 2 we show an interpretation of a KB with a set of human individuals, and a set of individuals that are instances of concept `Animal`. Some individuals own an animal, which is described with role assertions $\texttt{owns}(\texttt{Alice}, \texttt{Fido})$, $\texttt{owns}(\texttt{Bob}, \texttt{Fido})$, $\texttt{owns}(\texttt{Charlie}, \texttt{Rocky})$ (shown as arrows). `Alice`, `Bob` and `Charlie` may be described as instances of $\exists\texttt{owns}.\texttt{Animal}$. Notice that if we reversed all `owns` arrows, then an `owns` arrow would always start in set `Animal`, and end in set $\exists\texttt{owns}.\texttt{Animal}$, as by definition $(\exists R.C)^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid \exists_{y \in \Delta^{\mathcal{I}}}((x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}})\}$, all elements of $(\exists R.C)^{\mathcal{I}}$ are related to some element of $C^{\mathcal{I}}$. So in general, if we visualize role assertions as arrows, for any interpretation $\mathcal{I}$, if we reverse role R_i arrows, they always begin in set $C^{\mathcal{I}}$ and end in set $(\exists R_i.C)^{\mathcal{I}}$. Now consider the right side of the figure. Our embedding layer represents concepts as points in the embedding space $\mathbb{R}^{N_e}$ and does not support individuals at all, so we "squish" the sets $\exists\texttt{owns}.\texttt{Animal}$ and `Animal` into points in $\mathbb{R}^{N_e}$, but we keep the (reversed) arrows, since their start and end points always occur inside the sets. Thus, we think that it is appropriate to model the existential restriction constructor as a function mapping concept embeddings $h_{\mathcal{K}}(C)$ to embeddings of the existential restriction $h_{\mathcal{K}}(\exists R_i.C)$. There is a separate function $NN_{\mathcal{K},R_i}$ for each role, and functions $NN_{\mathcal{K},R_i}$ need to be learned per knowledge base $\mathcal{K}$, because different KBs have different numbers of roles, and roles may have different meanings, so there is no obvious way of sharing existential restriction constructors between KBs. The formalization is given in Equation 10.

$$h_{\mathcal{K}}(\exists R_i.C) = NN_{\mathcal{K},R_i}(h_{\mathcal{K}}(C)) \tag{10}$$

In deep learning, activation functions must be used after each hidden layer so that one does not compose multiple linear transformations, which are equivalent to a single linear transformation. Note that even though our constructor networks do not have an activation function, the result of the $h_{\mathcal{K}}$ function is not simply a composition of multiple

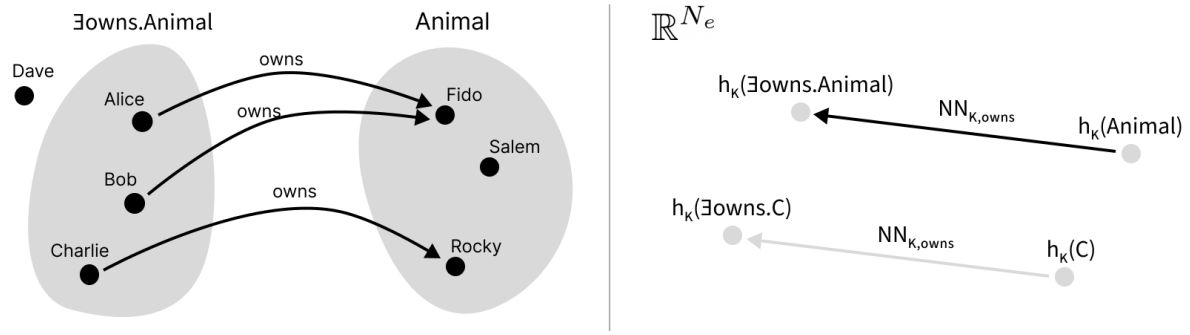

Fig. 2. An example interpretation of a knowledge base with role assertions is shown on the left side. The individuals `Fido`, `Salem`, and `Rocky` are elements of the set $\texttt{Animal}^{\mathcal{I}}$. Individuals `Alice`, `Bob`, and `Charlie` are elements of set $(\exists\texttt{owns.Animal})^{\mathcal{I}}$. An illustration of transformations from concept embeddings to existential restriction embeddings is shown on the right side. The embedding of $\exists\texttt{owns}.C$ may be obtained by applying $NN_{\mathcal{K},\texttt{owns}}$ to the embedding of concept C.

linear transformations, because at every recursive step the constructor networks (i.e. linear transformations) are given different inputs. If one would like to make the constructor networks deep, i.e., add more hidden layers, then one must add an activation function like ELU after each layer, except the last one.

To extend $h_{\mathcal{K}}$ to concept unions and universal restrictions, we use the de Morgan's laws [9], arriving at the formulas given in Equation 11.

$$\begin{aligned} h_{\mathcal{K}}(C \sqcup D) &= h_{\mathcal{K}}(\neg(\neg C \sqcap \neg D)) \\ h_{\mathcal{K}}(\forall R_i.C) &= h_{\mathcal{K}}(\neg\exists R_i.\neg C) \end{aligned} \tag{11}$$

4.3. Relaxed architecture

An important property of the embedding layer, that we introduced in the previous section, is that the weights of the neural networks $NN_{\neg}$ and $NN_{\sqcap}$ do not depend on a specific knowledge base. Since the semantics of concept complement and intersection constructors do not change depending on a KB, we think that their neural equivalents should also not change. We call a reasoner, where $NN_{\neg}$ and $NN_{\sqcap}$ network weights are stored in the reasoner head, the *restricted reasoner*. Unless stated otherwise, we use restricted reasoners.

While sharing constructors mimics the logic, it may not be an appropriate assumption in a deep learning model and thus a hindrance to learn good embeddings. To verify whether this is the case, we consider a variant of the reasoner, that allows the embedding layer to learn KB-specific concept constructor networks $NN_{\mathcal{K},\neg}$ and $NN_{\mathcal{K},\sqcap}$, and a KB-specific embedding of the bottom concept $\mathbf{W}_{\mathcal{K},\bot}$ and the top concept $\mathbf{W}_{\mathcal{K},\top}$. We call this variant the *relaxed reasoner*. All changes to the function $h_{\mathcal{K}}$ in the relaxed reasoner are marked below in red.

$$\begin{aligned} h_{\mathcal{K}}(\neg C) &= NN_{\mathcal{K},\neg}(h_{\mathcal{K}}(C)) \\ h_{\mathcal{K}}(C \sqcap D) &= NN_{\mathcal{K},\sqcap}(IM_{\mathcal{K}}(C, D)) \\ h_{\mathcal{K}}(\top) &= \mathbf{W}_{\mathcal{K},\top} \\ h_{\mathcal{K}}(\bot) &= \mathbf{W}_{\mathcal{K},\bot} \end{aligned} \tag{12}$$

In the relaxed architecture, the only weights shared between different KBs are the ones learned by the classifier $NN_{\sqsubseteq}$, which we expected would allow the embedding layers to learn more fitting embeddings, at the cost of limiting generalization and opportunities for transfer learning.

4.4. Training procedure

The training procedure for our reasoner consists of two steps. In the first step, we train both the reasoner head and embedding layers for as many diverse KBs as possible. This results in a reasoner head that learned to classify whether subsumption axioms hold in any KB, given that an appropriate embedding layer is provided. By appropriate embedding layer we mean a layer that learned to embed KB-specific concepts in a space that minimizes the classifier loss. We simply call this step *training the reasoner head*, and we consider the data used in this step as the *training set*. Note that, as a result of training the reasoner, we obtain trained embedding layers for KBs in the training set. If obtaining the trained embedding layers was the goal, then the next step is not necessary.

In the second step, we freeze the reasoner head and train the embedding layers for KBs that were not seen in the first step. This results in embedding layers that can embed concepts in a space in which the reasoner is good at classification. We think that if our reasoner can accurately classify whether subsumption axioms are entailed by a KB, then the embeddings used as inputs to the classifier $NN_{\sqsubseteq}$ faithfully capture the semantics of the KB, *even though we did not train the reasoner head in the second step*. We call this step *training the embedding layers* or *testing the reasoner head*, and we consider the data set used in this step as the *test set*.

5. Axiom generator

Our reasoner learns embeddings by learning to classify entailment queries $\mathcal{K} \models C \sqsubseteq D$. A set of query axioms could be created manually, but for learning to be successful a large number of subsumption axioms is needed. One way of obtaining a large number of axioms would be to generate all possible axioms for a given KB, up to some maximum expression tree depth. This approach would suffice for shallow expression trees and KBs with very small vocabularies, but is impractical otherwise. A more practical method of obtaining a large set of axioms is to pseudo-randomly generate them, an approach which is fast, reproducible by setting the initial state of the pseudo-random number generator to a fixed value, yet offering a virtually unlimited number of axioms, at the small cost of possibly generating duplicates.

One algorithm for pseudo-randomly generating $\mathcal{ALC}$ expressions and axioms was introduced by Eberhart et al. [76]. Unfortunately, their implementation requires the Java Virtual Machine, which did not suit our purposes. We thus decided to implement in Python our own custom generator inspired by their algorithm. It has four parameters: the set of concept names N_C, the set of role names N_R, the probability p_A of generating a concept name instead of a top or bottom concept, and the maximum depth of recursion d_{max}. Expressions are generated recursively and the current recursion depth d is tracked. At each step, the generator uses a grammar rule by replacing the symbol from the left side of the rule with one of the symbols from the right side.

Axioms are generated by starting with the grammar rule **A** (Equation 13), which returns a subsumption axiom or a disjointness axiom with equal probability. The second alternative is intentionally redundant, to ensure that disjointness axioms are generated frequently enough for the reasoner to benefit from them. Regardless of which alternative was returned, two concepts are generated according to grammar rule **C** (Equation 14). First, we select at random the maximum recursion depth for the axiom d'_{max} from the discrete uniform distribution $\mathcal{U}\{1, d_{max}\}$. Then, the maximum depth of recursion for the first concept $d_{max,1}$ is chosen from the uniform discrete distribution $\mathcal{U}\{1, d'_{max}\}$, and the maximum depth of recursion for the second concept $d_{max,2}$ is chosen from $\mathcal{U}\{1, \max\{1, d'_{max} - d_{max,1}\}\}$. This technique of limiting the expression depth results in axioms where one of the concepts is deeper than the other, or axioms where both concepts are relatively shallow, to ensure satisfactory performance of checking entailment of the generated axioms.

$$\mathbf{A} ::= \mathbf{C} \sqsubseteq \mathbf{C} \mid \mathbf{C} \sqcap \mathbf{C} \sqsubseteq \bot \tag{13}$$

The grammar rule **C** returns one of the following alternatives with equal probability: a concept according to the grammar rule **T** (Equation 15), a concept complement, a concept intersection, a concept union, an existential restriction, or a universal restriction. For existential and universal restrictions, the role name R_i is chosen uniformly

at random from N_R. If the recursion depth d is equal to or greater than d_{max}, then a concept according to the grammar rule **T** is always returned.

$$\mathbf{C} ::= \mathbf{T} \mid \neg\mathbf{C} \mid \mathbf{C} \sqcap \mathbf{C} \mid \mathbf{C} \sqcup \mathbf{C} \mid \exists R_i.\mathbf{C} \mid \forall R_i.\mathbf{C} \tag{14}$$

The grammar rule **T** returns one of the following alternatives: with probability p_A, a concept name A_i chosen uniformly at random from N_C, the top concept with probability $\frac{1-p_A}{2}$, or the bottom concept with probability $\frac{1-p_A}{2}$. To ensure proper exposure of the reasoner to the top and bottom concepts during training, we use the parameter p_A to make the probability of returning $\top$ or $\bot$ independent of the number of concept names $|N_C|$.

$$\mathbf{T} ::= A_i \mid \top \mid \bot \tag{15}$$

6. Experimental evaluation

6.1. Goals

Although the design of the proposed approach mimics the $\mathcal{ALC}$ logic and is well-grounded in research on deep learning architectures, ultimately its usability and properties must be established experimentally. To this end, we formulate four research questions and design appropriate experiments to answer them.

The main hypothesis of our research is that using a transferable reasoner head forces the embeddings to reflect the semantics of the underlying KBs in a shared way. To address it, we pose the following research question **RQ1**: *Can the reasoner induce a useful topology that it can subsequently exploit, or are the embeddings simply overfitting the KBs?* We answer **RQ1** in Experiment 1, described in subsection 6.5, where we compare the classification metrics of a trained reasoner and a random reasoner in previously unseen ontologies, and show that the trained reasoner indeed brings something to the table.

In subsection 4.3, we constructed a relaxed variant of the reasoner, hypothesising that the restricted variant may be too restrictive and impede the learning process. From this, we formulate **RQ2**: *Can concept constructors be shared between KBs, or must they be KB-specific?* To answer it, we designed Experiment 2, described in subsection 6.6. In there, we follow the same procedure as in Experiment 1, but using the relaxed reasoner. We then compare the results obtained by the restricted reasoner in Experiment 1, and by the relaxed reasoner, and show that the difference is negligible.

The overarching goal of this work is to present a method to train embeddings for $\mathcal{ALC}$ KBs including the semantics of the concepts, and thus we pose **RQ3**: *Can the presented approach learn good embeddings for concepts from a real-world ontology?* We propose Experiment 3, described in subsection 6.7, where we use the pizza ontology as the real-world ontology. We consider three different training schemes, measure the classification metrics, and visually analyze the obtained embeddings, showing that the reasoner head from Experiment 1 fares only slightly worse than a reasoner head overfitted to the ontology.

We extend **RQ1** and **RQ3** to a larger set of real-world ontologies, formulating **RQ4**: *Is transfer learning with the presented approach viable for a variety of real-world ontologies?* In Experiment 4, described in subsection 6.8, we follow the methodology established in Experiment 3 with 6 different ontologies of varying sizes and complexity and show that on average the results of using a pre-trained reasoner head are as good as training a reasoner head from scratch on the considered ontologies.

Finally, we tackle the problem of identifying what is hard for the reasoner by posing **RQ5** *Are there types of queries that are significantly harder for the reasoner than others?*. We briefly address this question in Experiment 5, however, without reaching a satisfactory answer.

6.2. Setup

We implemented the proposed approach in Python 3.9.7 and Cython [77] (a superset of Python that compiles to C or C++), and used PyTorch 1.10.1 to implement the neural reasoner [78]. The source code is available at https://github.com/maxadamski/reasonable-embeddings. We ran the experiments on a computer with an Intel Core i5-4670K CPU, 32 GB of DDR3 RAM, and no dedicated GPU, running Void Linux with kernel 5.15.

We use pseudo-randomly generated numbers to create our data sets and in training. For data generation we use the NumPy implementation of the Permuted Congruential Generator [79]. The internal PyTorch pseudo-random number generator is used during training. To ensure reproducibility of our experiments, we set the initial states of the pseudo-random number generators to a known initial value.

During training, our reasoner uses inferences made by a semantic reasoner as the expected class in classification. The Semantic Web community traditionally uses Java as the language of choice, a language which does not interface well with Python. In particular, using either HermiT [10] or Pellet [11], state-of-the-art semantic reasoners, required executing the reasoners as separate Java Virtual Machine (JVM) processes, a process for every inference. This introduced unacceptable overhead, which we alleviated by employing FaCT++ [12], a semantic reasoner implemented in C++, which makes the task of interfacing with Python much easier. We improved a Python interface for FaCT++ by wrobell[2], extending it with missing concept constructors, and addressing performance issues.

In the experiments dealing with OWL ontologies, initially we used Owlready2 [80] to parse OWL ontologies in the RDF/XML format. Eventually, to improve performance and avoid some problems with the parser, we switched to converting the ontologies to the OWL functional-style syntax [81] using the ROBOT command-line tool [82], and parsing them using a custom, high-performance parser implemented in Cython.

Both our parser and the extended version of the Python interface for FaCT++ are available at https://github.com/maxadamski/reasonable-embeddings/tree/main/src/simplefact, together with compilation instructions in the root of the repository.

6.3. Evaluation metrics

As the quality of the learned embeddings cannot be measured directly, we perform so-called extrinsic evaluation by computing classification metrics for a trained reasoner. We stipulate that if the classification metrics indicate good performance, then the embeddings learned by the embedding layer must capture the semantics of a given KB well. If that was not the case and the embeddings did not encode useful information for inference in the $\mathcal{ALC}$ DL, then the classifier should not be able to reliably classify entailment of axioms.

For simplicity, in threshold-sensitive metrics we choose a threshold of 0.5, i.e, $\hat{y} = 1$ if $P(\mathcal{K} \models \alpha) \geqslant 0.5$, and $\hat{y} = 0$ otherwise. Given that y is the expected class, TP is the number of true positives, i.e., the number of queries where $\hat{y} = y = 1$; TN is the number of true negatives, i.e., the number of queries where $\hat{y} = y = 0$; FP is the number of false positives, i.e., the number of queries where $\hat{y} = 1$ and $y = 0$; and FN is the number of false negatives, i.e., the number of queries where $\hat{y} = 0$ and $y = 1$.

Our classifier learns binary classification, so we chose appropriate metrics [83]. Firstly, we include accuracy (Equation 16) in the set of evaluation metrics. Because the data set that we generated for the experiments is slightly imbalanced we also compute precision (Equation 17), recall (Equation 18), and the F1-score (Equation 19).

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{16}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{17}$$

[2] https://bitbucket.org/wrobell/factplusplus/src/factpp/factpp/

$$\text{Recall} = \text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{18}$$

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{19}$$

We also compute the values of the area under the ROC curve (AUC-ROC) and the area under the PR curve (AUC-PR), which are threshold-invariant metrics with a minimum value of 0 and a maximum value of 1. The ROC curve shows the performance of a classification model at all classification thresholds, by plotting the true positive rate (TPR; also called recall) against the false positive rate (FPR) (Equation 20). Similarly, the PR curve shows the performance of a model at all thresholds by plotting precision against recall [84]. For both ROC and PR curves, a larger area under the curve suggests a better classifier [84, 85].

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{20}$$

In experiments leveraging multiple KBs, we compute all metrics separately for each query set, and then compute the average and standard deviation across KBs. We do this because the KBs in the test set may be unequally difficult to classify, so it is beneficial to measure the variance of the reasoner performance across different KBs.

6.4. Data sets

6.4.1. Synthetic data set

Finding a large number of $\mathcal{ALC}$ KBs with a small number of concept and role names proved difficult, so for some experiments we generated a synthetic data set, and the code to reproduce it is available in the repository. It consists of 60 randomly generated KBs, such that the number of concept names $|N_C|$ for each KB is chosen randomly from the discrete uniform distribution $\mathcal{U}\{80, 120\}$, and the number of role names $|N_R|$ is chosen randomly from the discrete uniform distribution $\mathcal{U}\{1, 5\}$. The number of role names is very small, because we observed that the number of role names in real-world KBs is typically much smaller than the number of concept names. The actual concept and role names are simply ordinal numbers and are not considered in any way during learning.

For each KB, we also randomly choose the number of terminological axioms $|\mathcal{T}|$ to generate, from the normal distribution $\lfloor \mathcal{N}(200, 10) \rfloor$. The axioms were generated according to the procedure that we described in section 5, and we set the maximum depth of axioms d_{max} to 3. The parameter p_A is chosen randomly from the uniform distribution $\mathcal{U}[0.9, 1]$ for each KB. We discard and generate anew the axioms that would make the KB inconsistent, or make more than 10% of the named concepts unsatisfiable.

For each knowledge base $\mathcal{K}$, we generate 2000 unique queries $\mathcal{K} \models \alpha$, where α is a random subsumption axiom. The same parameters of the random axiom generator are used, as during generating terminological axioms for that KB. For each query, we label it with $y = 1$ if the entailment holds according to FaCT++, and with $y = 0$ otherwise.

We assign the first 40 KBs as the training set and the remaining 20 KBs as the test set. We then create a validation set from 20% of the queries from the training set. In total, there are 64,000 queries in the training set, 16,000 queries in the validation set, and 40,000 queries in the test set. In every data set, approximately 21.5% of queries have class $y = 1$ and the remaining queries have class $y = 0$. This class imbalance does not significantly affect our experiments, so we do not use any methods for dealing with imbalanced data sets. We report detailed statistics of the data set in appendix B.

6.4.2. Data set of real-world ontologies

Preprocessing We do the following pre-processing steps to make OWL ontologies compatible with our reasoner. Pre-processing is done automatically after parsing the ontology file, so one does not need to edit the ontology manually.

- We ignore object property axioms, since role axioms are not expressible in $\mathcal{ALC}$ (including role hierarchies, inverse roles, transitive roles, and (inverse) functional roles).
- We ignore axioms with number or value restrictions, since they are not expressible in $\mathcal{ALC}$ with KBs where ABox is empty.
- We ignore individuals and ABox axioms, since our reasoner does not support ABox reasoning.
- We remove roles that do not appear in any TBox axiom kept after pre-processing. This is done because of a limitation in FaCT++, which makes it raise an exception, when constructing a concept with an unused role name.

The pizza ontology We use the pizza ontology[3] from the Manchester University OWL Tutorial [86] We chose the pizza ontology, because it has a similar number of axioms, concept and role names to the randomly generated KBs from the synthetic data set.

The ontology contains 99 classes and 8 object properties, which are equivalent to concept names and role names, respectively. There are 15 equivalence axioms and 15 class disjointness axioms. Two of the classes are unsatisfiable.

The preprocessing is not without some influence on the ontology: `InterestingPizza` is equivalent to the top concept, because the axiom defining it as a pizza with at least 3 toppings contained a number restriction, so it was removed. The equivalence axiom that defines `RealItalianPizza` as a pizza with Italy as its country of origin was removed because it contained a value restriction. Since `RealItalianPizza` was also defined as a subclass of $\exists$`hasBase.ThinAndCrispyBase`, any pizza with a thin and crispy base is inferred to be a subclass of `RealItalianPizza`. Value restrictions for `American`, `AmericanHot`, `Napoletana`, `Veneziana`, and `MozzarellaTopping` were removed, but that did not have a significant effect on reasoning.

CQ2SPARQLOWL To further test the capabilities of our architecture, we use the set of ontologies accompanying the CQ2SPARQLOWL data set[4] [87, 88]. It consists of five ontologies:

- Software Ontology[5] (SWO) with 4068 classes, 52 object properties and 7683 axioms [89]
- Stuff ontology[6] (Stuff) with 193 classes, 57 object properties and 717 axioms [90]
- African Wildlife Ontology[7] (AWO) with 31 classes, 5 object properties and 56 axioms [91]
- Dementia Ambient Care ontology[8] (Dem@Care) with 261 classes, 72 object properties and 695 axioms [92]
- Ontology of Datatypes[9] (OntoDT) with 406 classes, 16 object properties and 947 axioms [93]

The ontologies were preprocessed according to the procedure described earlier.

6.5. Experiment 1 – Topology induction on the synthetic data set

In this experiment, we consider **RQ1**: *Can the reasoner induce a useful topology that it can subsequently exploit, or are the embeddings simply overfitting the KBs?* To answer it, we designed a simple test to check whether a trained reasoner head actually learns to reason in the $\mathcal{ALC}$ description logic, and high classification metric values are not just the effect of embedding layers overfitting to KBs.

We test our architecture by first training the reasoner head and embedding layers on the training set (as usual), which results in a skillful reasoner. Then we create a no-skill reasoner head, that is not trained, but only randomly initialized. For each of the two reasoner heads, we train embedding layers on the test set, but keep the reasoner head weights frozen. After training the embedding layers on the test set for a fixed number of epochs, if the classification metrics for the reasoner with the trained head are significantly greater than for the reasoner with the random head, then the trained reasoner head learned useful relations in the $\mathcal{ALC}$ embedding space that generalize to new KBs. In short, the reasoner head is transferable.

[3] The pizza ontology in the RDF/XML format is available at http://owl.cs.manchester.ac.uk/publications/talks-and-tutorials/protg-owl-tutorial/
[4] https://github.com/CQ2SPARQLOWL/Dataset
[5] Path in the repository: `Ontologies/swo_merged.owl`
[6] Path in the repository: `Ontologies/stuff.owl`
[7] Path in the repository: `Ontologies/AfricanWildlifeOntology1.owl`
[8] Path in the repository: `Ontologies/exchangemodel.owl`
[9] Path in the repository: `Ontologies/OntoDT.owl`

Table 2

Test set metrics for the restricted reasoner. Metric values were averaged across different KBs in the test set. In addition to averages, we standard deviation values are shown.

Model	Accuracy	Precision	Recall	F1	AUC-ROC	AUC-PR
Trained head	0.9610 ± 0.0148	0.9690 ± 0.0249	0.8469 ± 0.0446	0.9036 ± 0.0346	0.9838 ± 0.0086	0.9642 ± 0.0184
Random head	0.2873 ± 0.0275	0.2318 ± 0.0305	0.9870 ± 0.0120	0.3746 ± 0.0399	0.7656 ± 0.0242	0.5588 ± 0.0522

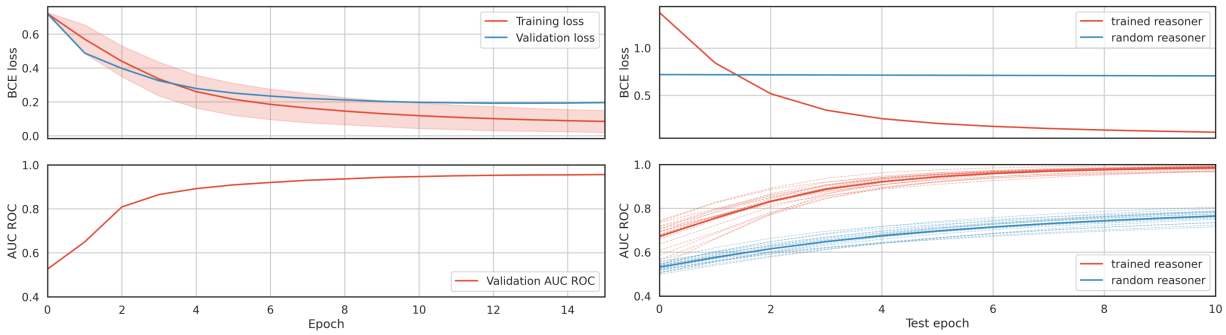

Fig. 3. Training and test progress of the restricted reasoner. The reported training loss for each epoch is the average mini-batch loss in that epoch. We also show the standard deviation of the mini-batch loss for each epoch. In test progress, the trained reasoner is shown in red, while the random reasoner is shown in blue. The reasoner was *not* trained during epoch 0 of training and testing – during epoch 0 we only compute the initial loss and metric values. Dashed curves show the AUC-ROC for each KB in the test set, while the thicker blue and red curves are the averaged AUC-ROC across KBs in the test set. Note that the random reasoner loss actually decreases, but at a very slow rate.

Conversely, if the differences between classification metrics of both reasoner heads are not significant, then the trained reasoner actually has little or no skill and the only skill in classification comes from the embedding layer, which would mean that the reasoner head is not transferable.

6.5.1. Training details

The embedding dimension is set to $N_e = 10$, which is a much smaller number than $|N_C|$. This forces the embeddings for different concept names to share embedding dimensions. As the loss for our classifier we use binary cross entropy. We create one AdamW optimizer [3] per reasoner head with learning rate set to $\eta_r = 0.0001$, and one AdamW optimizer per embedding layer with learning rate set to $\eta_e = 0.0002$. In total, for each reasoner variant, we create 1 optimizer for the reasoner head, and 60 optimizers for the embedding layers.

We train both reasoner variants with mini-batch gradient descent for 15 epochs, which was enough for the validation loss to stop decreasing. During testing, we train the embedding layers (while the reasoner head weights are frozen) for 10 epochs, as the test loss stabilized after that. We set the batch size to 32, as small batch sizes have been shown to improve generalization [94]. Unless stated otherwise, all weights are randomly initialized using the Xavier initialization [95].

6.5.2. Evaluation of reasoning ability

We evaluate the reasoning ability of the restricted variant of our reasoner by monitoring the training and validation loss and AUC-ROC values during training. The only goal of this assessment is to verify that the reasoner can effectively learn to classify entailment in the training set, and does not suffer from underfitting. Training and validation losses steadily decreasing, and validation AUC-ROC increasing with each training epoch, suggests that the reasoner architecture is sufficient for learning to classify entailment in a given data set.

As shown in Figure 3, the training loss decreases and validation AUC-ROC increases for the entirety of the training, with smaller gains after epoch 10. Furthermore, the validation loss does not start increasing in later epochs, which suggests that the restricted reasoner variant is not prone to overfitting.

Table 3
Test set metrics for the restricted reasoners with varying embedding size (N_e), and the number of neurons in the reasoner head (k). Metric values were averaged across different KBs in the test set.

k	N_e	Accuracy	Precision	Recall	F1	AUC-ROC	AUC-PR
1	1	0.8168 ± 0.0220	0.9145 ± 0.0680	0.1661 ± 0.0666	0.2762 ± 0.1008	0.7372 ± 0.0298	0.5863 ± 0.0662
16	1	0.8731 ± 0.0255	0.8000 ± 0.0922	0.5514 ± 0.0848	0.6496 ± 0.0809	0.8751 ± 0.0303	0.7541 ± 0.0802
1	10	0.9559 ± 0.0133	0.9694 ± 0.0242	0.8212 ± 0.0447	0.8888 ± 0.0341	0.9788 ± 0.0095	0.9567 ± 0.0193
16	10	0.9611 ± 0.0149	0.9691 ± 0.0249	0.8470 ± 0.0447	0.9036 ± 0.0347	0.9838 ± 0.0086	0.9643 ± 0.0184

6.5.3. Evaluation of knowledge transfer

After training the restricted reasoner, we froze the reasoner head, and trained embedding layers on the test set. We also trained separate embedding layers in conjunction with a randomly initialized, frozen reasoner head. The test loss decreased quickly for the reasoner with the trained head, while the loss for the random head decreased so slowly, that the test loss curve seems stationary on the plot. For the reasoner with the trained head, the test AUC-ROC quickly increased to almost 0.8 after epoch 2, and approached 1 after the last epoch.

Overall, training embedding layers for the reasoner with the trained head was much faster, than for the reasoner with the randomly initialized head, as the reasoner with the trained head achieved average AUC-ROC greater than 0.8 after epoch 2, while it took the reasoner with the random head 10 epochs to do the same. Furthermore, the trained head allowed the reasoner to achieve an average AUC-ROC close to 1 on the test set after 10 epochs, while the reasoner with the random head only achieved an average AUC-ROC of around 0.8.

The metrics after the last training epoch on the test data are reported in Table 2. The extremely high recall of the reasoner with the randomly initialized head is an artifact of it classifying most queries as class 1. Given the very low precision of the reasoner with the random head, its high recall should be ignored. The restricted reasoner with the trained head has lower variance of AUC-ROC and accuracy, than the reasoner with the random head. However, the restricted reasoner with the random head has lower variance for the F1-score, precision and recall. The trained reasoner outperforms the random reasoner on all measures except recall.

6.5.4. Measuring the effect of embedding size and reasoner head width on performance

To determine the effect of the embedding size and the number of neurons in the reasoner head we set the embedding size to 1 and/or the number of neurons to 1, and repeated the training procedure we described earlier. A reasoner with 1 neuron is a linear classifier and thus must rely on the embeddings to convey almost all the necessary information. Conversely, an embedding of size 1 is sufficient if all the necessary information is stored in the reasoner.

We report the results in Table 3. Following [96], we used repeated measures ANOVA on the AUC-ROC and obtained a p-value below 0.001, i.e., for at least one pair (k, N_e) there is a significant difference in the mean at the significance level $\alpha = 0.01$. We then performed pairwise comparisons using the paired t-test and obtained all p-values below 0.001, indicating the differences in AUC-ROC between every pair of rows in Table 3 are statistically significant at the family-wise error rate 0.01 with the Bonferroni correction.

The variant with $k = 1$, $N_e = 1$ barely works, with accuracy on par with always predicting the positive class (recall the datasets are not balanced, with on average 21.5% of examples in the positive class). The second-worst variant with $k = 16$, $N_e = 1$ is slightly better but is still characterized by low recall value. Both variants with $N_e = 10$ perform much better than any of the variants for $N_e = 1$. Albeit there is still a difference in performance between them, it is, on average, rather small.

Answering **RQ1**, the presented results indicate that the reasoner indeed shapes the embedding space and generalizes well to previously unseen KBs, not storing any substantial amount of knowledge.

6.6. Experiment 2 – Comparing restricted and relaxed reasoners

In this experiment we answer **RQ2**: *Can concept constructors be shared between KBs, or must they be KB-specific?* We followed the same protocol as in Experiment 1, but used the relaxed reasoner instead of the restricted reasoner.

Table 4

Test set metrics for the relaxed reasoner. Metric values were averaged across different KBs in the test set. In addition to averages, the standard deviation values are shown.

Model	Accuracy	Precision	Recall	F1	AUC-ROC	AUC-PR
Trained head	0.9585 ± 0.0138	0.9713 ± 0.0196	0.8332 ± 0.0488	0.8964 ± 0.0340	0.9836 ± 0.0086	0.9636 ± 0.0176
Random head	0.7755 ± 0.0463	0.4901 ± 0.1006	0.5494 ± 0.1363	0.5107 ± 0.0965	0.7898 ± 0.0583	0.6352 ± 0.1167

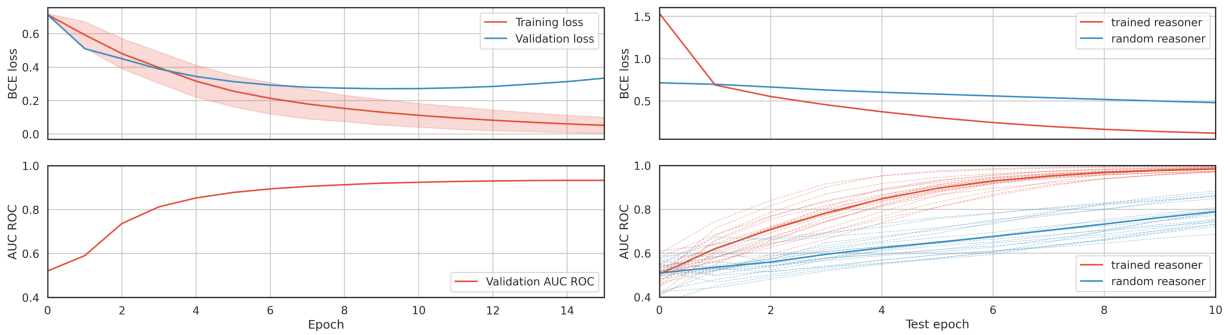

Fig. 4. Training and test progress of the relaxed reasoner. The reported training loss for each epoch is the average mini-batch loss in that epoch. We also show the standard deviation of the mini-batch loss for each epoch. In test progress, the trained reasoner is shown in red, while the random reasoner is shown in blue. The reasoner was *not* trained during epoch 0 of training and testing – during epoch 0 we only compute the initial loss and metric values. Dashed lines show the AUC-ROC for each KB in the test set, while the thicker blue and red curves are the averaged AUC-ROC across KBs in the test set.

6.6.1. Evaluation of reasoning ability

First, we trained the relaxed reasoner on the training set. The training progress for this variant is shown in Figure 4. The training loss decreases and the validation AUC-ROC increases for the entirety of the training, with smaller gains after epoch 10. The validation loss decreases until epoch 10, and then starts increasing, which indicates overfitting.

When comparing Figure 4 with Figure 3, we observe that the AUC-ROC metric for the validation set increases slower for the relaxed reasoner than for the the restricted reasoner. We attribute the faster convergence to the shared concept constructor networks $NN_\neg$, $NN_\sqcap$, which effectively have n times more training samples, where n is the number of training KBs, because they are trained on every sample from every KB, compared to the relaxed architecture, in which KB-specific concept constructor networks are trained on samples from one KB. As mentioned, the validation loss starts increasing in later epochs, suggesting that the relaxed reasoner is more prone to overfitting than the restricted reasoner.

6.6.2. Evaluation of knowledge transfer

After training the relaxed reasoner, we froze the reasoner head, and trained embedding layers on the test set. We also trained embedding layers in conjunction with a randomly initialized reasoner head. The test progress is shown on the right side of Figure 4. At the beginning, the test loss for the trained reasoner head was higher than the test loss for the randomly initialized reasoner head. The test loss decreased quickly for the reasoner with the trained head, while the loss for the random head decreased very slowly. For the reasoner with the trained head, the test AUC-ROC quickly increased to about 0.7 after epoch 2, and approached 1 after the last epoch. The average test AUC-ROC for the random head slowly increased from around 0.5 in the beginning to around 0.8 after the last epoch.

Overall, training embedding layers for the relaxed reasoner with the trained head was much faster, than for the reasoner with the randomly initialized head. Moreover, the trained head allowed the reasoner to achieve an average AUC-ROC close to 1 on the test set after 10 epochs, while the reasoner with the random head only achieved an average AUC-ROC of around 0.8.

We report the final metrics of training on the test data in Table 4. For the relaxed architecture, the trained model is strictly better than the randomly initialized one, as all metric values are higher for the former. Moreover, the trained

model has lower variance of metrics across KBs in the test set, than the random model. Similarly as for the restricted reasoner, the reasoner with trained head outperforms the reasoner with random head by a fair margin.

6.6.3. Comparative results

Comparing the metrics for the restricted reasoner in Table 2 and for the relaxed reasoner in Table 4, we observe that as expected, for both relaxed and restricted reasoners, the reasoners with trained heads achieved superior performance on the test set, which shows that both variants are indeed transferable.

Between the two random reasoners, the relaxed variant achieves better metric values, except recall, due to the very high recall of the restricted random reasoner. This was expected, since in the relaxed variant, the complement and intersection constructor networks can adjust to the randomly classifier to minimize the classification error, while in the restricted variant this is not possible.

When comparing the trained reasoners, the relaxed variant achieves slightly higher values with lower variance for all metrics (except precision, for which the restricted variant has slightly lower variance). However, the differences are not statistically significant. For each metric, we conducted a paired t-test, and checked whether the metric values for KBs from the test set, do not significantly differ between the relaxed and restricted reasoners. The resulting p-values, shown in Table 5, indicate that the null hypothesis cannot be rejected for any metric, even on a relatively high significance level $\alpha = 0.05$ and not including corrections for the family-wise error rate.

Table 5

p-values of the t statistic in a paired t-test for the test set metrics of a trained restricted reasoner and trained relaxed reasoner.

	Accuracy	Precision	Recall	F1	AUC-ROC	AUC-PR
p	0.1798	0.4431	0.0801	0.1529	0.9236	0.8399

Disregarding the performance metrics, the restricted reasoner has an advantage over the restricted reasoner – it has fewer learnable parameters. Since one of the main goals of our reasoner is transfer learning, a lower number of parameters in the embedding layer is preferable, as it speeds up training for new KBs. The average embedding layer training time per epoch was approximately 21.72 seconds for the restricted variant, and 26.84 seconds per epoch for the relaxed variant, which is 23% slower.

Answering **RQ2**, introducing restrictions does not significantly change the reasoning performance, as measured by the classification metrics, but increases training efficiency.

6.7. Experiment 3 – Visualizing concept embeddings from a real-world ontology

The previous experiments showed that our reasoner is capable of learning good concept embeddings for synthetic KBs, so the next step is to answer **RQ3**: *Can the presented approach learn good embeddings for concepts from a real-world ontology?* In this experiment we do that by training the neural reasoner to classify entailment, given randomly generated queries about the pizza ontology as the data set. We repeat this experiment three times. In the first run, we let the reasoner learn without any pre-training. In the second run, we initialize the reasoner head with weights of the restricted reasoner, that we trained in Experiment 1, then freeze it, and only allow the embedding layer to learn. In the third run, we randomly initialize the reasoner head, and also freeze it to only allow the embedding layer to learn KB-specific embedding.

6.7.1. Training set

The data set for this experiment consists of 32,000 unique random queries that we generate using the algorithm described in section 5. We set the maximum axiom depth to $d_{max} = 4$, and the probability of concept names to $p_A = 0.95$. The answer to each query is obtained using FaCT++. We do not split the data set because we want the learned embeddings to fit the data set as well as possible. The data set is slightly imbalanced, with 41.11% of queries belonging to class 1, and the rest to class 0, although the class imbalance did not significantly affect classification performance.

Table 6
Classification metrics for reasoners in Experiment 3 after training for 30 epochs.

Reasoner head	Accuracy	Precision	Recall	F1	AUC-ROC	AUC-PR	Training time
Unfrozen	0.9929	0.9897	0.9930	0.9914	0.9996	0.9995	411.16s
Frozen pre-trained	0.9527	0.9508	0.9331	0.9419	0.9859	0.9823	354.99s
Frozen random	0.7733	0.6593	0.9283	0.7710	0.8932	0.8390	328.49s

6.7.2. Training procedure

As mentioned, we repeat learning three times, which results in three reasoners:

- *Reasoner with unfrozen head* – Both the reasoner head and embedding layers were trained.
- *Reasoner with frozen pre-trained head* – Only embedding layers were trained. The reasoner head head was initialized with the weights of the restricted reasoner, that we obtained in Experiment 1.
- *Reasoner with frozen random head* – Only embedding layers were trained. The reasoner head was initialized randomly and its weights were immediately frozen.

In every run of the experiment we trained the model for 30 epochs, with learning rate set to $\eta_r = \eta_e = 0.001$ both for the optimizer of the reasoner head and the embedding layers. Similarly to the previous experiments, we use the AdamW optimizer and train the reasoners with batch size of 32. Because we initialize the frozen pre-trained reasoner head with weights of the restricted reasoner head, that we trained in Experiment 1, the embedding dimension N_e and the layers of the neural network $NN_{\sqsubseteq}$ needed to be exactly the same as in that experiment. Thus, for all reasoners in this experiment we set $N_e = 10$ and defined $NN_{\sqsubseteq}$ to be a neural network with one hidden layer with 16 neurons and the ELU activation function. The embedding dimension is adequate, since the pizza ontology has a similar number of concepts to the ontologies in the synthetic data set that we used in Experiment 1.

We expected the reasoner with the unfrozen head to achieve the best metric values and learn the best embeddings out of the three reasoners in the embedding analysis, because no weights are frozen, which means that the reasoner with the unfrozen head can fit to the data set more than the reasoners with frozen heads. The only obstacle to learning good embeddings for the reasoner with unfrozen head are the randomly generated queries, that may not contain useful entailments for the pizza ontology, although the other two reasoners learn with the same data set, so the comparison is at least fair.

Based on the results of Experiment 1, we expected the reasoner with transfer to achieve higher classification metrics than the reasoner with randomly initialized frozen head. In Experiment 1, the trained reasoner head was better at classifying queries for the unseen KBs from the test set, than the randomly initialized reasoner head, so we expected the same to be true for the pizza ontology.

6.7.3. Evaluation of reasoning ability

The classification metrics of the three reasoners are shown in Table 6. As expected, the unfrozen reasoner achieves strictly better classification performance than the reasoners with frozen heads, as the values of all metrics are higher than for other reasoners. The reasoner with the frozen pre-trained head is the second-best reasoner after the unfrozen reasoner, and is strictly better than the reasoner with the randomly initialized head.

The reasoner with the randomly initialized frozen head was the worst of the three reasoners, although it is better than a random guesser, with AUC-ROC of around 0.90. This reasoner has very high recall, but relatively low precision, which is reflected in the significantly lower F1-score and AUC-PR, when compared to the better reasoners. Even though the reasoner with the randomly initialized head was the worst of the three, it still achieved relatively high metric values, which shows that good embeddings can compensate for a bad reasoner head.

6.7.4. Evaluation of knowledge transfer

In the last section, we discussed the differences between the three reasoners that we trained in Experiment 3. Taking into account the differences between the reasoners with frozen heads, we conclude that the transfer of knowledge from the randomly generated KBs in Experiment 1 to the pizza ontology was a success. The reasoner with the pre-trained head achieved results similar to the unfrozen reasoner, which is very promising, given that the pizza ontology is certainly different from the randomly generated KBs used as the training set in Experiment 1.

It should be mentioned that the total training time of the reasoners with frozen heads is shorter than the training time of the unfrozen reasoner, which is the case because there is no time spent on updating the reasoner head weights.

6.7.5. Embedding analysis

In addition to evaluating the reasoning and transfer ability of our reasoner, which showed that it can successfully learn to classify entailment in a real-world KB, we visualize the learned embeddings. A 2D visualization that tries to preserve the distances between concepts in the high-dimensional embedding space enables visual assessment of the quality of the learned embeddings. We think that if semantically similar concepts are placed closer to each other than to dissimilar concepts, then the learned embeddings capture the semantics of a given KB.

We set the concept embedding dimension to $N_e = 10$, so to visualize the embeddings we first needed to reduce their dimensionality. We used Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP)[10], because it is a nonlinear dimensionality reduction algorithm with high performance and interpretable hyperparameters [97], that we set as follows: `n_neighbors` to 50 to emphasize the global structure of the data, and set `min_dist` by starting at a large value and decreasing it until clusters started to separate and the visualization became readable, which occurred when `min_dist` was set to 0.3.

The UMAP visualizations of the embeddings learned in this experiment, are shown in Figure 5. The visualization of embeddings learned by the unfrozen reasoner suggests, that the embeddings capture the semantics of the pizza ontology well. As expected, the unsatisfiable `CheesyVegetableTopping` is close to $\bot$, and the general `DomainThing` and `Food` concepts are close to $\top$. Spiciness and pizza bases form their own clusters. Pizzas and toppings are separated, with vegetarian pizzas forming one cluster, and non-vegetarian pizzas forming another cluster together with spicy pizzas. Cheesy toppings, vegetable toppings, spicy toppings, and pepper toppings are also close to related concepts.

The embeddings learned by the reasoner with the pre-trained head look a bit worse than those of the unfrozen reasoner. The pizzas and the toppings are separated, but inside of the topping and pizza clusters the embeddings are not as well organized.

The embeddings learned by the reasoner with the randomly initialized head do not look good when visualized. A few concept names form clusters that make sense, but most look like they are randomly scattered. General concepts are close to the top concept, which is good, but the unsatisfiable `CheesyVegetableTopping` is far away from the bottom concept.

In general, we think that the embeddings for the unfrozen reasoner are the best, the embeddings for the reasoner with the frozen pre-trained head are good, and the embeddings for the reasoner with the frozen randomly initialized head are not good at all. Again, the results of the visual assessment of the learned embeddings are consistent with the classification metrics of the reasoners.

Overall, the presented results indicate that the answer to **RQ3** is affirmative, and that the presented approach can learn embeddings of good quality.

6.8. Experiment 4 – Learning concept embeddings in different real-world ontologies

In Experiment 3 we found that a reasoner with a frozen pre-trained head achieves similar classification metrics to a reasoner trained from scratch. To see whether this result holds for a more diverse set of ontologies we posed **RQ4**: *Is transfer learning with the presented approach viable for a variety of real-world ontologies?*

6.8.1. Training set

In this experiment we use six real-world ontologies: five ontologies from the CQ2SPARQLOWL data set, and the pizza ontology. For each ontology, we generate $32{,}000$ unique random queries according to the algorithm described in section 5. We set the maximum axiom depth to $d_{max} = 4$, and the probability of concept names to $p_A = 0.95$. The answer to each query is obtained using FaCT++. Similarly to Experiment 3, we do not split the data set because we want the learned embeddings to fit the data set as well as possible. Class imbalance depends on the ontology,

[10] https://github.com/lmcinnes/umap

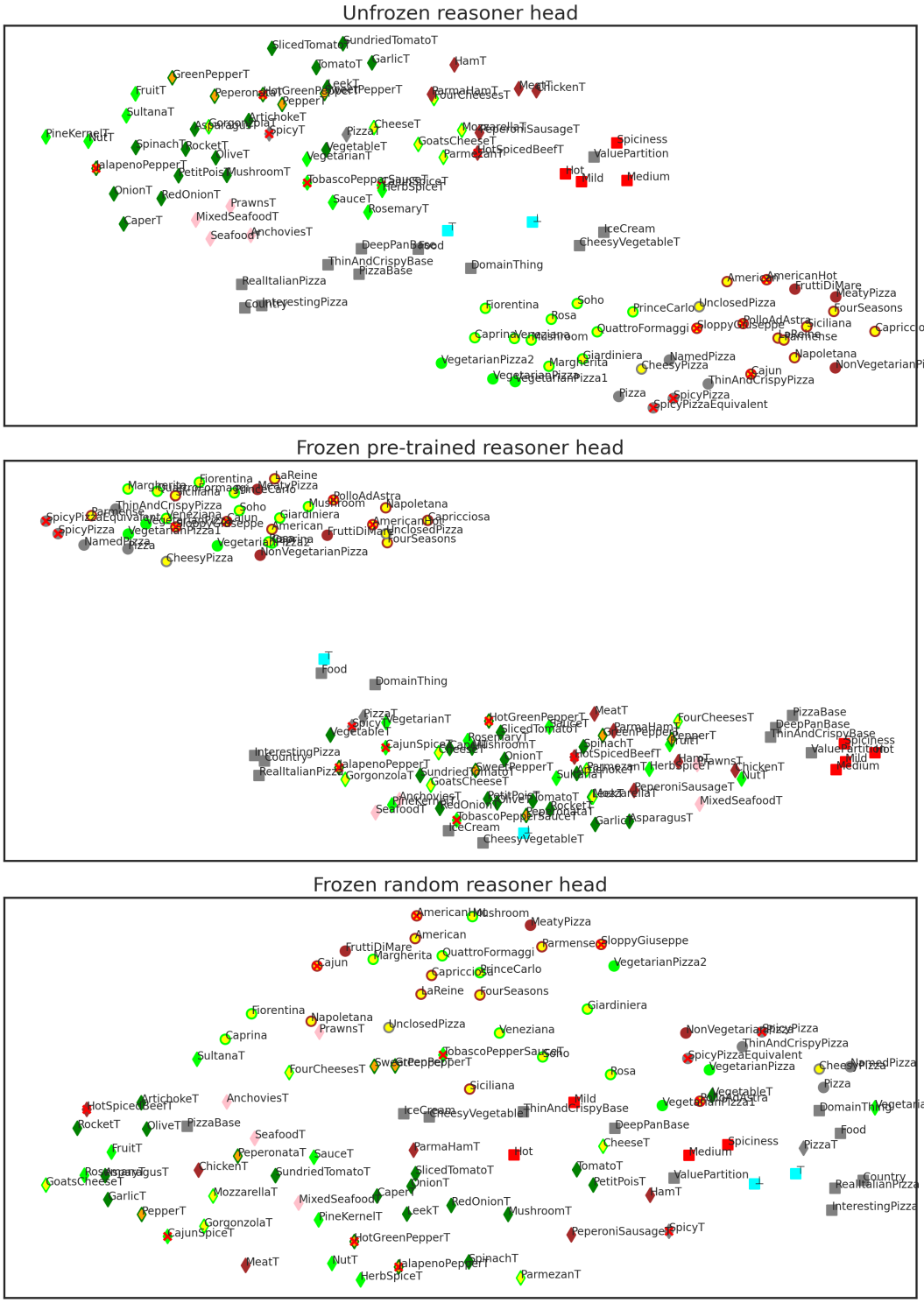

Fig. 5. UMAP visualization of the learned concept embeddings. We replaced "Topping" in concept names with "T" to improve readability. We use different shapes, sizes, and colors of markers as follows: by default concepts are square and gray; the top concept, bottom concept and concept expressions are cyan; toppings diamond-shaped, and pizzas are round; vegetarian pizzas and toppings are light green, except vegetable toppings, which are dark green; seafood toppings are pink; non-vegetarian pizzas and meat toppings are dark red; cheesy pizzas, and cheese toppings are respectively marked with a yellow disk or yellow diamond inside; pepper toppings are marked with an orange diamond inside; spicy things are marked with a red "x" inside; spiciness levels are light red.

Table 7

Classification metrics for reasoners for the AWO, Dem@Care, Stuff, SWO, OntoDT, and Pizza ontologies. Values were averaged over all ontologies.

Reasoner head	Accuracy	Precision	Recall	F1	AUC-ROC	AUC-PR	Train time
Unfrozen	0.9766 ± 0.0135	0.9425 ± 0.0489	0.9017 ± 0.0772	0.9207 ± 0.0574	0.9907 ± 0.0076	0.9637 ± 0.0343	27min 18s
Frozen pre-trained	0.9600 ± 0.0219	0.9121 ± 0.0605	0.8449 ± 0.1122	0.8755 ± 0.0838	0.9776 ± 0.0156	0.9284 ± 0.0566	24min 6s
Frozen random	0.7789 ± 0.0468	0.3672 ± 0.2526	0.4371 ± 0.1679	0.3879 ± 0.2198	0.7230 ± 0.0819	0.3460 ± 0.2497	24min 5s

ranging from only 4.24% of the Dem@Care ontology queries belonging to class 1, to 41.11% of the pizza ontology queries in class 1. We reused queries for the pizza ontology from Experiment 3.

6.8.2. Training procedure

Similarly to the Experiment 3, we repeat learning three times, which results in three reasoner heads, each with six embedding layers.

- *Reasoner with unfrozen head* – Both the reasoner head and embedding layers were trained.
- *Reasoner with frozen pre-trained head* – Only embedding layers were trained. The reasoner head head was initialized with the weights of the restricted reasoner, that we obtained in Experiment 1.
- *Reasoner with frozen random head* – Only embedding layers were trained. The reasoner head was initialized randomly.

We use the exact reasoner architecture and training setup as in Experiment 3, but in this experiment each mini-batch could contain queries for any of the six ontologies, as the embedding layers were trained in parallel.

6.8.3. Results

The metric values for the pizza ontology are slightly different from the ones in Experiment 3, as this time, the reasoner head was trained on six ontologies at once, in contrast to exclusively training on the pizza ontology. We report the classification metrics for each reasoner head, averaged over all ontologies, in Table 7, and we give the values for each ontology separately in appendix A.

In Table 8 we report the p-values obtained from a paired t-test between the unfrozen head, and the frozen pre-trained head. As some of the p-values hint on the possibility of statistical significance, we also report the adjusted p-values $\tilde{p}$, obtained using the Holm-Bonferroni method. After adjustment, on the significance level $\alpha = 0.05$, no difference is statistically significant.

The results indicate that using a pre-trained head is a viable choice for real-world ontologies, as the performance drop is negligible, and thus the answer for **RQ4** is positive.

Table 8

p-values of the t statistic in a paired t-test for the test set metrics of a reasoner with an unfrozen head and a frozen pre-trained head. Adjusted p-values $\tilde{p}$ were obtained using the Holm-Bonferroni method.

	Accuracy	Precision	Recall	F1	AUC-ROC	AUC-PR
p	0.0338	0.0432	0.0319	0.0277	0.0208	0.0335
$\tilde{p}$	0.1383	0.1383	0.1383	0.1383	0.1244	0.1383

6.9. Experiment 5 – What is difficult for the reasoner?

One can wonder what types of axioms are particularly hard for the reasoner. To answer this, we started with the reasoner and the test set from Experiment 1. Then, for each query in the test set, we computed the following features:

- $n_\bot$ – the number of occurrences of the bottom concept in the query;
- n_C – the number of different concept names in the query;

– n_R – the number of different role names in the query;
– $n_\neg$ – the number of negations $\neg$ in the query;
– $n_\sqcap$ – the number of conjunctions $\sqcap$ in the query;
– $n_\exists$ – the number of existential restrictions $\exists$ in the query;
– d – the depth of the syntax tree of the query (i.e., the recursion depth d as defined in section 5);

Next, we computed the Pearson correlation coefficient between each feature and the classification error of the reasoner (before thresholding). The absolute values of the obtained coefficients (Table 9) were all below 0.05, indicating that none of the features have a significant impact on the performance of the reasoner.

Currently, we do not have a decisive answer to **RQ5**. There do not seem to be any obvious indicators of hardness for a query. We stipulate it might be possible to detect more complex patterns by using, e.g., an adversarial attack [98], or frequent pattern mining [99, 100]. However, we deem this to be out of the scope of this work.

Table 9
The Pearson correlation coefficient between each feature and the classification error of the reasoner.

$n_\bot$	$n_\neg$	$n_\sqcap$	$n_\exists$	n_C	n_R	d
-0.0106	-0.0051	-0.0062	-0.0015	0.0282	0.0004	-0.0065

7. Conclusions

In this work, we introduced a novel method of learning data-driven concept embeddings, called reason-able embeddings, in $\mathcal{ALC}$ knowledge bases. To our best knowledge, our method of learning concept embeddings is the first one using an entailment classifier based on neural networks. Thanks to of our unique approach, after learning embeddings, one can use the resulting classifier to perform fast approximate reasoning.

We also show that using recursive neural networks for constructing embeddings of arbitrarily complex concepts obviates the need for manually designing concept vectorization schemes, and avoids the pitfalls of recurrent neural networks operating on a textual representations of concepts. Instead, concept embeddings can be learned in a data-driven way, by simply asking entailment queries for a given knowledge base.

Finally, we show that a significant part of our reasoner is transferable across knowledge bases in the $\mathcal{ALC}$ description logic, including real-world knowledge bases like the pizza ontology, or the ontologies accompanying the CQ2SPARQLOWL data set. The advantage of a transferable reasoner is that learning concept embeddings takes less time, and thus is less expensive, when using a pre-trained reasoner head, compared to training an entire reasoner from scratch. Promising results in the transferability of neural reasoners suggest that it is indeed feasible to embed concepts from multiple domains in a single shared embedding space.

We hope that our neural reasoner architecture will allow for greater use of knowledge in models based on neural networks, both by providing an effective way of learning concept embeddings, and learning an accurate entailment classifier for knowledge bases in description logics, thus, making a small step towards the integration of the neural and symbolic paradigms in artificial intelligence.

We identified many opportunities to improve, extend and apply our neural reasoner, that were out-of-scope for this work, but look like promising avenues for future research. In our work we used small neural networks, but deeper and wider concept constructor networks, and subsumption entailment classifier networks could be examined. The number of parameters in the reasoner could also be reduced, while preserving the quality of embeddings and accuracy of entailment classification. Currently, the number of parameters scales quadratically with the embedding dimension, because the reasoner uses the outer product of embeddings as an input to neural networks $NN_\sqsubseteq$ and $NN_\sqcap$. Instead of passing the entire interaction map as input, convolutional neural networks could be used, in turn greatly reducing the number of parameters. The number of parameters used by existential restriction constructor networks could possibly be reduced by representing weights as sparse matrices.

Finally, it would be interesting to see if recursive neural networks could be applied in reverse to how we use them – to generate concepts, given learned concept embeddings. That would make it possible to not only learn concept

embeddings by classifying entailment, but also to induce new concepts by sampling the embedding space, e.g., to construct a scalable algorithm for explainable artificial intelligence [101].

Acknowledgements

This paper is, in part, a summary of the master thesis of Dariusz Max Adamski, done under the supervision of Jedrzej Potoniec. This research was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215. We would like to thank Prof. Agnieszka Ławrynowicz for helpful feedback on our work.

References

[1] P.R. Fillottrani and C.M. Keet, Dimensions Affecting Representation Styles in Ontologies, in: *Knowledge Graphs and Semantic Web - First Iberoamerican Conference, KGSWC 2019, Villa Clara, Cuba, June 23-30, 2019, Proceedings*, B. Villazón-Terrazas and Y. Hidalgo-Delgado, eds, Communications in Computer and Information Science, Vol. 1029, Springer, 2019, pp. 186–200. doi:10.1007/978-3-030-21395-4_14.

[2] I.J. Goodfellow, Y. Bengio and A.C. Courville, *Deep Learning*, Adaptive computation and machine learning, MIT Press, 2016. ISBN 978-0-262-03561-3. http://www.deeplearningbook.org/.

[3] I. Loshchilov and F. Hutter, Decoupled Weight Decay Regularization, in: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019. https://openreview.net/forum?id=Bkg6RiCqY7.

[4] D. Clevert, T. Unterthiner and S. Hochreiter, Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs), in: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, eds, 2016. http://arxiv.org/abs/1511.07289.

[5] F. Baader and W. Nutt, Basic description logics, in: *The description logic handbook: theory, implementation, and applications*, Cambridge University Press, USA, 2003, pp. 43–95. ISBN 9780521781763. https://www.inf.unibz.it/~franconi/dl/course/dlhb/dlhb-02.pdf.

[6] D. McGuinness and F. van Harmelen, OWL Web Ontology Language Overview, 2004, https://www.w3.org/TR/2004/REC-owl-features-20040210/.

[7] P. Hitzler, M. Krotzsch and S. Rudolph, *Foundations of Semantic Web Technologies*, 0 edn, Chapman and Hall/CRC, 2009. ISBN 9781420090512. doi:10.1201/9781420090512. https://www.taylorfrancis.com/books/9781420090512.

[8] T. Benson, *Principles of Health Interoperability HL7 and SNOMED*, Health Informatics, Springer London, London, 2010. ISBN 9781848828025 9781848828032. doi:10.1007/978-1-84882-803-2.

[9] S. Rudolph, Foundations of Description Logics, in: *Reasoning Web. Semantic Technologies for the Web of Data - 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*, A. Polleres, C. d'Amato, M. Arenas, S. Handschuh, P. Kroner, S. Ossowski and P.F. Patel-Schneider, eds, Lecture Notes in Computer Science, Vol. 6848, Springer, 2011, pp. 76–136. doi:10.1007/978-3-642-23032-5_2.

[10] B. Motik, R.D.C. Shearer and I. Horrocks, Hypertableau Reasoning for Description Logics, *J. Artif. Intell. Res.* **36** (2009), 165–228. doi:10.1613/jair.2811.

[11] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur and Y. Katz, Pellet: A practical OWL-DL reasoner, *Journal of Web Semantics* **5**(2) (2007), 51–53. doi:10.1016/j.websem.2007.03.004. https://linkinghub.elsevier.com/retrieve/pii/S1570826807000169.

[12] D. Tsarkov and I. Horrocks, FaCT++ Description Logic Reasoner: System Description, in: *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, U. Furbach and N. Shankar, eds, Lecture Notes in Computer Science, Vol. 4130, Springer, 2006, pp. 292–297. doi:10.1007/11814771_26.

[13] Y. Kazakov, M. Krötzsch and F. Simancik, The Incredible ELK - From Polynomial Procedures to Efficient Reasoning with $\mathcal{EL}$ Ontologies, *J. Autom. Reason.* **53**(1) (2014), 1–61. doi:10.1007/s10817-013-9296-3.

[14] T.B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D.M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever and D. Amodei, Language Models are Few-Shot Learners, in: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan and H. Lin, eds, 2020. https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[15] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu and M. Chen, Hierarchical Text-Conditional Image Generation with CLIP Latents, *CoRR* **abs/2204.06125** (2022). doi:10.48550/arXiv.2204.06125.

[16] M. Ebrahimi, A. Eberhart, F. Bianchi and P. Hitzler, Towards bridging the neuro-symbolic gap: deep deductive reasoners, *Applied Intelligence* **51**(9) (2021), 6326–6348. doi:10.1007/s10489-020-02165-6.

[17] P. Hitzler, F. Bianchi, M. Ebrahimi and M.K. Sarker, Neural-symbolic integration and the Semantic Web, *Semantic Web* **11**(1) (2020), 3–11. doi:10.3233/SW-190368.

[18] G.G. Towell and J.W. Shavlik, Knowledge-based artificial neural networks, *Artificial Intelligence* **70**(1–2) (1994), 119–165. doi:10.1016/0004-3702(94)90105-8. https://linkinghub.elsevier.com/retrieve/pii/0004370294901058.

[19] J. Chen, Y. He, E. Jimenez-Ruiz, H. Dong and I. Horrocks, Contextual Semantic Embeddings for Ontology Subsumption Prediction, *arXiv:2202.09791 [cs]* (2022), arXiv: 2202.09791. http://arxiv.org/abs/2202.09791.

[20] R. Socher, D. Chen, C.D. Manning and A.Y. Ng, Reasoning With Neural Tensor Networks for Knowledge Base Completion, in: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, C.J.C. Burges, L. Bottou, Z. Ghahramani and K.Q. Weinberger, eds, 2013, pp. 926–934. https://proceedings.neurips.cc/paper/2013/hash/b337e84de8752b27eda3a12363109e80-Abstract.html.

[21] A. Bordes, N. Usunier, A. García-Durán, J. Weston and O. Yakhnenko, Translating Embeddings for Modeling Multi-relational Data, in: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, C.J.C. Burges, L. Bottou, Z. Ghahramani and K.Q. Weinberger, eds, 2013, pp. 2787–2795. https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html.

[22] A. Bordes, X. Glorot, J. Weston and Y. Bengio, A semantic matching energy function for learning with multi-relational data - Application to word-sense disambiguation, *Mach. Learn.* **94**(2) (2014), 233–259. doi:10.1007/s10994-013-5363-6.

[23] B. Yang, W. Yih, X. He, J. Gao and L. Deng, Embedding Entities and Relations for Learning and Inference in Knowledge Bases, in: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, eds, 2015. http://arxiv.org/abs/1412.6575.

[24] Z. Wang, J. Zhang, J. Feng and Z. Chen, Knowledge Graph Embedding by Translating on Hyperplanes, in: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, C.E. Brodley and P. Stone, eds, AAAI Press, 2014, pp. 1112–1119. http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8531.

[25] Y. Lin, Z. Liu, M. Sun, Y. Liu and X. Zhu, Learning Entity and Relation Embeddings for Knowledge Graph Completion, in: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*, B. Bonet and S. Koenig, eds, AAAI Press, 2015, pp. 2181–2187. http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9571.

[26] Y. Lin, Z. Liu, H. Luan, M. Sun, S. Rao and S. Liu, Modeling Relation Paths for Representation Learning of Knowledge Bases, in: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, L. Màrquez, C. Callison-Burch, J. Su, D. Pighin and Y. Marton, eds, The Association for Computational Linguistics, 2015, pp. 705–714. doi:10.18653/v1/d15-1082.

[27] Z. Sun, Z. Deng, J. Nie and J. Tang, RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space, in: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019. https://openreview.net/forum?id=HkgEQnRqYQ.

[28] Z. Zhang, J. Cai, Y. Zhang and J. Wang, Learning Hierarchy-Aware Knowledge Graph Embeddings for Link Prediction, in: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, AAAI Press, 2020, pp. 3065–3072. https://ojs.aaai.org/index.php/AAAI/article/view/5701.

[29] B. Wang, Q. Meng, Z. Wang, D. Wu, W. Che, S. Wang, Z. Chen and C. Liu, InterHT: Knowledge Graph Embeddings by Interaction between Head and Tail Entities, *CoRR* **abs/2202.04897** (2022). https://arxiv.org/abs/2202.04897.

[30] X. Zhou, Y. Yi and G. Jia, Path-RotatE: Knowledge Graph Embedding by Relational Rotation of Path in Complex Space, in: *10th IEEE/CIC International Conference on Communications in China, ICCC 2021, Xiamen, China, July 28-30, 2021*, IEEE, 2021, pp. 905–910. doi:10.1109/ICCC52777.2021.9580273.

[31] T. Dettmers, P. Minervini, P. Stenetorp and S. Riedel, Convolutional 2D Knowledge Graph Embeddings, in: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S.A. McIlraith and K.Q. Weinberger, eds, AAAI Press, 2018, pp. 1811–1818. https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17366.

[32] D.Q. Nguyen, T.D. Nguyen, D.Q. Nguyen and D.Q. Phung, A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network, in: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, M.A. Walker, H. Ji and A. Stent, eds, Association for Computational Linguistics, 2018, pp. 327–333. doi:10.18653/v1/n18-2053.

[33] X. Jiang, Q. Wang and B. Wang, Adaptive Convolution for Multi-Relational Learning, in: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran and T. Solorio, eds, Association for Computational Linguistics, 2019, pp. 978–987. doi:10.18653/v1/n19-1103.

[34] C. Demir and A.N. Ngomo, Convolutional Complex Knowledge Graph Embeddings, in: *The Semantic Web - 18th International Conference, ESWC 2021, Virtual Event, June 6-10, 2021, Proceedings*, R. Verborgh, K. Hose, H. Paulheim, P. Champin, M. Maleshkova, Ó. Corcho, P. Ristoski and M. Alam, eds, Lecture Notes in Computer Science, Vol. 12731, Springer, 2021, pp. 409–424. doi:10.1007/978-3-030-77385-4_24.

[35] D.Q. Nguyen, T. Vu, T.D. Nguyen, D.Q. Nguyen and D.Q. Phung, A Capsule Network-based Embedding Model for Knowledge Graph Completion and Search Personalization, in: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1*

(Long and Short Papers), J. Burstein, C. Doran and T. Solorio, eds, Association for Computational Linguistics, 2019, pp. 2180–2189. doi:10.18653/v1/n19-1226.

[36] L. Guo, Z. Sun and W. Hu, Learning to Exploit Long-term Relational Dependencies in Knowledge Graphs, in: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, K. Chaudhuri and R. Salakhutdinov, eds, Proceedings of Machine Learning Research, Vol. 97, PMLR, 2019, pp. 2505–2514. http://proceedings.mlr.press/v97/guo19c.html.

[37] Z. Wang, Q. Lv, X. Lan and Y. Zhang, Cross-lingual Knowledge Graph Alignment via Graph Convolutional Networks, in: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, E. Riloff, D. Chiang, J. Hockenmaier and J. Tsujii, eds, Association for Computational Linguistics, 2018, pp. 349–357. doi:10.18653/v1/d18-1032.

[38] M.S. Schlichtkrull, T.N. Kipf, P. Bloem, R. van den Berg, I. Titov and M. Welling, Modeling Relational Data with Graph Convolutional Networks, in: *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, A. Gangemi, R. Navigli, M. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai and M. Alam, eds, Lecture Notes in Computer Science, Vol. 10843, Springer, 2018, pp. 593–607. doi:10.1007/978-3-319-93417-4_38.

[39] S. Werner, A. Rettinger, L. Halilaj and J. Lüttin, RETRA: Recurrent Transformers for Learning Temporally Contextualized Knowledge Graph Embeddings, in: *The Semantic Web - 18th International Conference, ESWC 2021, Virtual Event, June 6-10, 2021, Proceedings*, R. Verborgh, K. Hose, H. Paulheim, P. Champin, M. Maleshkova, Ó. Corcho, P. Ristoski and M. Alam, eds, Lecture Notes in Computer Science, Vol. 12731, Springer, 2021, pp. 425–440. doi:10.1007/978-3-030-77385-4_25.

[40] M. Nickel, V. Tresp and H. Kriegel, A Three-Way Model for Collective Learning on Multi-Relational Data, in: *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, L. Getoor and T. Scheffer, eds, Omnipress, 2011, pp. 809–816. https://icml.cc/2011/papers/438_icmlpaper.pdf.

[41] A. Jachnik, A. Szwabe, P. Misiorek and P. Walkowiak, TOAST results for OAEI 2012, in: *Proceedings of the 7th International Workshop on Ontology Matching, Boston, MA, USA, November 11, 2012*, P. Shvaiko, J. Euzenat, A. Kementsietsidis, M. Mao, N.F. Noy and H. Stuckenschmidt, eds, CEUR Workshop Proceedings, Vol. 946, CEUR-WS.org, 2012. http://ceur-ws.org/Vol-946/oaei12_paper13.pdf.

[42] A. García-Durán, A. Bordes and N. Usunier, Effective Blending of Two and Three-way Interactions for Modeling Multi-relational Data, in: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part I*, T. Calders, F. Esposito, E. Hüllermeier and R. Meo, eds, Lecture Notes in Computer Science, Vol. 8724, Springer, 2014, pp. 434–449. doi:10.1007/978-3-662-44848-9_28.

[43] B. Yang, W. Yih, X. He, J. Gao and L. Deng, Embedding Entities and Relations for Learning and Inference in Knowledge Bases, in: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, eds, 2015. http://arxiv.org/abs/1412.6575.

[44] M. Nickel, L. Rosasco and T.A. Poggio, Holographic Embeddings of Knowledge Graphs, in: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, D. Schuurmans and M.P. Wellman, eds, AAAI Press, 2016, pp. 1955–1961. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12484.

[45] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier and G. Bouchard, Complex Embeddings for Simple Link Prediction, in: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, M. Balcan and K.Q. Weinberger, eds, JMLR Workshop and Conference Proceedings, Vol. 48, JMLR.org, 2016, pp. 2071–2080. http://proceedings.mlr.press/v48/trouillon16.html.

[46] H. Liu, Y. Wu and Y. Yang, Analogical Inference for Multi-relational Embeddings, in: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, D. Precup and Y.W. Teh, eds, Proceedings of Machine Learning Research, Vol. 70, PMLR, 2017, pp. 2168–2178. http://proceedings.mlr.press/v70/liu17d.html.

[47] P. Ristoski, J. Rosati, T.D. Noia, R.D. Leone and H. Paulheim, RDF2Vec: RDF graph embeddings and their applications, *Semantic Web* **10**(4) (2019), 721–752. doi:10.3233/SW-180317.

[48] J. Chen, P. Hu, E. Jiménez-Ruiz, O.M. Holter, D. Antonyrajah and I. Horrocks, OWL2Vec*: embedding of OWL ontologies, *Mach. Learn.* **110**(7) (2021), 1813–1845. doi:10.1007/s10994-021-05997-6.

[49] C. d'Amato, N.F. Quatraro and N. Fanizzi, Injecting Background Knowledge into Embedding Models for Predictive Tasks on Knowledge Graphs, in: *The Semantic Web - 18th International Conference, ESWC 2021, Virtual Event, June 6-10, 2021, Proceedings*, R. Verborgh, K. Hose, H. Paulheim, P. Champin, M. Maleshkova, Ó. Corcho, P. Ristoski and M. Alam, eds, Lecture Notes in Computer Science, Vol. 12731, Springer, 2021, pp. 441–457. doi:10.1007/978-3-030-77385-4_26.

[50] C. Santini, G.A. Gesese, S. Peroni, A. Gangemi, H. Sack and M. Alam, A knowledge graph embeddings based approach for author name disambiguation using literals, *Scientometrics* **127**(8) (2022), 4887–4912. doi:10.1007/s11192-022-04426-2.

[51] J. Potoniec, Learning OWL 2 Property Characteristics as an Explanation for an RNN, *Bulletin of the Polish Academy of Sciences: Technical Sciences* **68**(No. 6) (2020), 1481–1490. doi:10.24425/bpasts.2020.134625. http://journals.pan.pl/Content/117659/PDF/23_D1481-1490_01383_Bpast.No.68-6_29.12.20_OK.pdf.

[52] J. Chen, F. Lécué, J.Z. Pan, S. Deng and H. Chen, Knowledge graph embeddings for dealing with concept drift in machine learning, *J. Web Semant.* **67** (2021), 100625. doi:10.1016/j.websem.2020.100625.

[53] G. Mai, K. Janowicz, L. Cai, R. Zhu, B. Regalia, B. Yan, M. Shi and N. Lao, SE-KGE : A location-aware Knowledge Graph Embedding model for Geographic Question Answering and Spatial Semantic Lifting, *Trans. GIS* **24**(3) (2020), 623–655. doi:10.1111/tgis.12629.

[54] U. Joshi and J. Urbani, Ensemble-Based Fact Classification with Knowledge Graph Embeddings, in: *The Semantic Web - 19th International Conference, ESWC 2022, Hersonissos, Crete, Greece, May 29 - June 2, 2022, Proceedings*, P. Groth, M. Vidal, F.M. Suchanek, P.A. Szekely, P. Kapanipathi, C. Pesquita, H. Skaf-Molli and M. Tamper, eds, Lecture Notes in Computer Science, Vol. 13261, Springer, 2022, pp. 147–164. doi:10.1007/978-3-031-06981-9_9.

[55] S. Choudhary, T. Luthra, A. Mittal and R. Singh, A Survey of Knowledge Graph Embedding and Their Applications, *CoRR* **abs/2107.07842** (2021). https://arxiv.org/abs/2107.07842.

[56] Q. Wang, Z. Mao, B. Wang and L. Guo, Knowledge Graph Embedding: A Survey of Approaches and Applications, *IEEE Trans. Knowl. Data Eng.* **29**(12) (2017), 2724–2743. doi:10.1109/TKDE.2017.2754499.

[57] Y. Dai, S. Wang, N.N. Xiong and W. Guo, A Survey on Knowledge Graph Embedding: Approaches, Applications and Benchmarks, *Electronics* **9**(5) (2020), 750. doi:10.3390/electronics9050750.

[58] H. Cai, V.W. Zheng and K.C. Chang, A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications, *IEEE Trans. Knowl. Data Eng.* **30**(9) (2018), 1616–1637. doi:10.1109/TKDE.2018.2807452.

[59] M. Wang, L. Qiu and X. Wang, A Survey on Knowledge Graph Embeddings for Link Prediction, *Symmetry* **13**(3) (2021). doi:10.3390/sym13030485. https://www.mdpi.com/2073-8994/13/3/485.

[60] G. Rizzo, C. d'Amato, N. Fanizzi and F. Esposito, Tree-based models for inductive classification on the Web Of Data, *J. Web Semant.* **45** (2017), 1–22. doi:10.1016/j.websem.2017.05.001.

[61] G. Rizzo, N. Fanizzi, C. d'Amato and F. Esposito, Approximate classification with web ontologies through evidential terminological trees and forests, *Int. J. Approx. Reason.* **92** (2018), 340–362. doi:10.1016/j.ijar.2017.10.019.

[62] H. Paulheim and H. Stuckenschmidt, Fast Approximate A-Box Consistency Checking Using Machine Learning, in: *The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings*, H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S.P. Ponzetto and C. Lange, eds, Lecture Notes in Computer Science, Vol. 9678, Springer, 2016, pp. 135–150. doi:10.1007/978-3-319-34129-3_9.

[63] B. Makni and J.A. Hendler, Deep learning for noise-tolerant RDFS reasoning, *Semantic Web* **10**(5) (2019), 823–862. doi:10.3233/SW-190363.

[64] P. Hohenecker and T. Lukasiewicz, Ontology Reasoning with Deep Neural Networks, *J. Artif. Intell. Res.* **68** (2020), 503–540. doi:10.1613/jair.1.11661.

[65] A. Eberhart, M. Ebrahimi, L. Zhou, C. Shimizu and P. Hitzler, Completion Reasoning Emulation for the Description Logic EL+, in: *Proceedings of the AAAI 2020 Spring Symposium on Combining Machine Learning and Knowledge Engineering in Practice, AAAI-MAKE 2020, Palo Alto, CA, USA, March 23-25, 2020, Volume I*, A. Martin, K. Hinkelmann, H. Fill, A. Gerber, D. Lenat, R. Stolle and F. van Harmelen, eds, CEUR Workshop Proceedings, Vol. 2600, CEUR-WS.org, 2020. https://ceur-ws.org/Vol-2600/paper5.pdf.

[66] B. Makni, M. Ebrahimi, D. Gromann and A. Eberhart, Neuro-Symbolic Semantic Reasoning, in: *Neuro-Symbolic Artificial Intelligence: The State of the Art*, H. Pascal and S.M. Kamruzzaman, eds, Frontiers in Artificial Intelligence and Applications, Vol. 342, IOS Press, 2021, pp. 253–279. doi:10.3233/FAIA210358.

[67] S. Badreddine, A. d'Avila Garcez, L. Serafini and M. Spranger, Logic Tensor Networks, *Artif. Intell.* **303** (2022), 103649. doi:10.1016/j.artint.2021.103649.

[68] F. Bianchi and P. Hitzler, On the Capabilities of Logic Tensor Networks for Deductive Reasoning, in: *Proceedings of the AAAI 2019 Spring Symposium on Combining Machine Learning with Knowledge Engineering (AAAI-MAKE 2019) Stanford University, Palo Alto, California, USA, March 25-27, 2019., Stanford University, Palo Alto, California, USA, March 25-27, 2019*, A. Martin, K. Hinkelmann, A. Gerber, D. Lenat, F. van Harmelen and P. Clark, eds, CEUR Workshop Proceedings, Vol. 2350, CEUR-WS.org, 2019. http://ceur-ws.org/Vol-2350/paper22.pdf.

[69] M. Ebrahimi, A. Eberhart and P. Hitzler, On the Capabilities of Pointer Networks for Deep Deductive Reasoning, *CoRR* **abs/2106.09225** (2021). https://arxiv.org/abs/2106.09225.

[70] M. Ebrahimi, M.K. Sarker, F. Bianchi, N. Xie, A. Eberhart, D. Doran, H. Kim and P. Hitzler, Neuro-Symbolic Deductive Reasoning for Cross-Knowledge Graph Entailment, in: *Proceedings of the AAAI 2021 Spring Symposium on Combining Machine Learning and Knowledge Engineering (AAAI-MAKE 2021), Stanford University, Palo Alto, California, USA, March 22-24, 2021*, A. Martin, K. Hinkelmann, H. Fill, A. Gerber, D. Lenat, R. Stolle and F. van Harmelen, eds, CEUR Workshop Proceedings, Vol. 2846, CEUR-WS.org, 2021. http://ceur-ws.org/Vol-2846/paper8.pdf.

[71] X. Zhu, B. Liu, Z. Ding, C. Zhu and L. Yao, Approximate Ontology Reasoning for Domain-Specific Knowledge Graph based on Deep Learning, in: *2021 7th International Conference on Big Data and Information Analytics (BigDIA)*, 2021, pp. 172–179. doi:10.1109/BigDIA53151.2021.9619694.

[72] C. Goller and A. Küchler, Learning task-dependent distributed representations by backpropagation through structure, in: *Proceedings of International Conference on Neural Networks (ICNN'96), Washington, DC, USA, June 3-6, 1996*, IEEE, 1996, pp. 347–352. doi:10.1109/ICNN.1996.548916.

[73] X. He, X. Du, X. Wang, F. Tian, J. Tang and T. Chua, Outer Product-based Neural Collaborative Filtering, in: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, J. Lang, ed., ijcai.org, 2018, pp. 2227–2233. doi:10.24963/ijcai.2018/308.

[74] M. Bednarek, P. Kicki and K. Walas, On Robustness of Multi-Modal Fusion—Robotics Perspective, *Electronics* **9**(7) (2020), 1152. doi:10.3390/electronics9071152. https://www.mdpi.com/2079-9292/9/7/1152.

[75] A. Ławrynowicz, J. Potoniec, M. Robaczyk and T. Tudorache, Discovery of emerging design patterns in ontologies using tree mining, *Semantic Web* **9**(4) (2018), 517–544. doi:10.3233/SW-170280.

[76] A. Eberhart, M. Cheatham and P. Hitzler, Pseudo-Random ALC Syntax Generation, in: *The Semantic Web: ESWC 2018 Satellite Events*, A. Gangemi, A.L. Gentile, A.G. Nuzzolese, S. Rudolph, M. Maleshkova, H. Paulheim, J.Z. Pan and M. Alam, eds, Lecture Notes in Computer Science, Springer International Publishing, Cham, 2018, pp. 19–22. ISBN 9783319981925. doi:10.1007/978-3-319-98192-5_4.

[77] S. Behnel, R. Bradshaw, C. Citro, L. Dalcín, D.S. Seljebotn and K. Smith, Cython: The Best of Both Worlds, *Comput. Sci. Eng.* **13**(2) (2011), 31–39. doi:10.1109/MCSE.2010.118.

[78] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, PyTorch: An Imperative Style, High-Performance Deep Learning Library, in: *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, eds, Curran Associates, Inc., 2019, pp. 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[79] M.E. O'Neill, PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation, Technical Report, HMC-CS-2014-0905, Harvey Mudd College, Claremont, CA, 2014. https://www.cs.hmc.edu/tr/hmc-cs-2014-0905.pdf.

[80] J.-B. Lamy, Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies, *Artificial Intelligence in Medicine* **80** (2017), 11–28. doi:10.1016/j.artmed.2017.07.002. https://linkinghub.elsevier.com/retrieve/pii/S0933365717300271.

[81] P. Patel-Schneider, B. Motik and B. Parsia, OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition), 2012, https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/.

[82] R.C. Jackson, J.P. Balhoff, E. Douglass, N.L. Harris, C.J. Mungall and J.A. Overton, ROBOT: A Tool for Automating Ontology Workflows, *BMC Bioinformatics* **20**(1) (2019), 407. doi:10.1186/s12859-019-3002-3.

[83] A. Tharwat, Classification assessment methods, *Applied Computing and Informatics* **17**(1) (2021), 168–192. doi:10.1016/j.aci.2018.08.003.

[84] T. Saito and M. Rehmsmeier, The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets, *PLOS ONE* **10**(3) (2015), e0118432. doi:10.1371/journal.pone.0118432.

[85] T. Fawcett, An introduction to ROC analysis, *Pattern Recognition Letters* **27**(8) (2006), 861–874. doi:10.1016/j.patrec.2005.10.010. https://linkinghub.elsevier.com/retrieve/pii/S016786550500303X.

[86] M. Horridge, A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.3, 2011, [Online; accessed 2022-06-20] http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf.

[87] D. Wisniewski, J. Potoniec, A. Lawrynowicz and C.M. Keet, Analysis of Ontology Competency Questions and their formalizations in SPARQL-OWL, *J. Web Semant.* **59** (2019). doi:10.1016/j.websem.2019.100534.

[88] J. Potoniec, D. Wiśniewski, A. Ławrynowicz and C.M. Keet, Dataset of ontology competency questions to SPARQL-OWL queries translations, *Data in Brief* **29** (2020), 105098. doi:https://doi.org/10.1016/j.dib.2019.105098. https://www.sciencedirect.com/science/article/pii/S2352340919314544.

[89] J. Malone, A. Brown, A.L. Lister, J.C. Ison, D. Hull, H.E. Parkinson and R. Stevens, The Software Ontology (SWO): a resource for reproducibility in biomedical data analysis, curation and digital preservation, *J. Biomed. Semant.* **5** (2014), 25. doi:10.1186/2041-1480-5-25.

[90] C.M. Keet, A Core Ontology of Macroscopic Stuff, in: *Knowledge Engineering and Knowledge Management - 19th International Conference, EKAW 2014, Linköping, Sweden, November 24-28, 2014. Proceedings*, K. Janowicz, S. Schlobach, P. Lambrix and E. Hyvönen, eds, Lecture Notes in Computer Science, Vol. 8876, Springer, 2014, pp. 209–224. doi:10.1007/978-3-319-13704-9_17.

[91] C.M. Keet, The African wildlife ontology tutorial ontologies, *J. Biomed. Semant.* **11**(1) (2020), 4. doi:10.1186/s13326-020-00224-y.

[92] S. Dasiopoulou, G. Meditskos and V. Efstathiou, Semantic Knowledge Structures and Representation, Technical Report, D5.1, FP7-288199 Dem@Care: Dementia Ambient Care: Multi-Sensing Monitoring for Intelligence Remote Management and Decision Support. http://www.demcare.eu/downloads/D5.1SemanticKnowledgeStructures_andRepresentation.pdf.

[93] P. Panov, L.N. Soldatova and S. Dzeroski, Generic ontology of datatypes, *Inf. Sci.* **329** (2016), 900–920. doi:10.1016/j.ins.2015.08.006.

[94] D. Masters and C. Luschi, Revisiting Small Batch Training for Deep Neural Networks, *CoRR* **abs/1804.07612** (2018). http://arxiv.org/abs/1804.07612.

[95] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, Y.W. Teh and D.M. Titterington, eds, JMLR Proceedings, Vol. 9, JMLR.org, 2010, pp. 249–256. http://proceedings.mlr.press/v9/glorot10a.html.

[96] N. Japkowicz and M. Shah (eds), *Evaluating Learning Algorithms: A Classification Perspective*, Cambridge University Press, 2011. ISBN 9780521196000.

[97] L. McInnes and J. Healy, UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, *CoRR* **abs/1802.03426** (2018). http://arxiv.org/abs/1802.03426.

[98] I.J. Goodfellow, J. Shlens and C. Szegedy, Explaining and Harnessing Adversarial Examples, in: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, eds, 2015. http://arxiv.org/abs/1412.6572.

[99] A. Lawrynowicz and J. Potoniec, Pattern Based Feature Construction in Semantic Data Mining, *Int. J. Semantic Web Inf. Syst.* **10**(1) (2014), 27–65. doi:10.4018/ijswis.2014010102.

[100] J. Potoniec, P. Jakubowski and A. Lawrynowicz, Swift Linked Data Miner: Mining OWL 2 EL class expressions directly from online RDF datasets, *J. Web Semant.* **46-47** (2017), 31–50. doi:10.1016/j.websem.2017.08.001.

[101] M.K. Sarker and P. Hitzler, Efficient Concept Induction for Description Logics, *Proceedings of the AAAI Conference on Artificial Intelligence* **33**(01) (2019), 3036–3043. doi:10.1609/aaai.v33i01.33013036. https://ojs.aaai.org/index.php/AAAI/article/view/4161.

Table 10
The values of the classification metrics separately for each ontology and run from Experiment 4

Model	Ontology	Accuracy	Precision	Recall	F1	AUC-ROC	AUC-PR
Unfrozen	AWO	0.9639	0.8697	0.8247	0.8466	0.9803	0.9214
	Dem@Care	0.9956	0.9984	0.8969	0.9449	0.9979	0.9772
	Stuff	0.9903	0.9853	0.9725	0.9789	0.9970	0.9921
	SWO	0.9635	0.9175	0.9588	0.9377	0.9929	0.9808
	OntoDT	0.9752	0.9187	0.7934	0.8515	0.9821	0.9187
	Pizza	0.9710	0.9654	0.9640	0.9647	0.9940	0.9920
Frozen pre-trained	AWO	0.9428	0.8105	0.6877	0.7441	0.9518	0.8363
	Dem@Care	0.9956	0.9984	0.8976	0.9453	0.9977	0.9762
	Stuff	0.9658	0.9258	0.9261	0.9260	0.9831	0.9410
	SWO	0.9558	0.9179	0.9290	0.9234	0.9861	0.9640
	OntoDT	0.9672	0.8981	0.7148	0.7960	0.9704	0.8828
	Pizza	0.9328	0.9218	0.9143	0.9180	0.9768	0.9700
Frozen random	AWO	0.7564	0.1638	0.2474	0.1971	0.6615	0.1644
	Dem@Care	0.8618	0.1090	0.3144	0.1619	0.6245	0.0851
	Stuff	0.7618	0.4870	0.5988	0.5371	0.7809	0.4905
	SWO	0.7482	0.5619	0.5529	0.5574	0.7977	0.5087
	OntoDT	0.8064	0.1687	0.2967	0.2151	0.6636	0.1361
	Pizza	0.7391	0.7126	0.6122	0.6586	0.8100	0.6912

Appendix A. Detailed results of Experiment 4

In Table 10 we report the values of classification metrics separately for each of the six ontologies considered in Experiment 4, and for each of the three runs.

Appendix B. Detailed statistics of the synthetic dataset

In Tables 11–13, we report detailed statistics about the queries of the synthetic dataset, introduced in subsubsection 6.4.1. We computed them separately for subsumption queries and for disjointness queries (see Equation 13). Moreover, for subsumption queries, we also computed them for only the left-hand sides (LHS) and only for the right-hand sides (RHS), since subsumptions are not commutative.

We computed the following statistics:

count number of queries;
depth depth of the syntax tree;
distinc concepts number of different concepts in a single query;
distinct roles number of different roles in a single query;
distinct constructors number of different constructors (i.e., $\sqcap$, $\sqcup$, $\neg$, $\forall$, $\exists$) in a single query;
concepts total number of concepts in a single query;
roles total number of roles in a single query;
constructors total number of constructors in a single query;
complement number of complement constructors $\neg$ in a single query;
with complement number of queries with at least one complement constructor;
intersection number of intersection constructors $\sqcap$ in a single query;
with intersection number of queries with at least one intersection constructor;
union number of union constructors $\sqcup$ in a single query;
with union number of queries with at least one union constructor;

Table 11
Detailed statistics of the training set of the synthetic data set

	subsumption			disjointness
	LHS	RHS	total	
count	31955	31955	31955	32045
depth	$1.413 \pm 0.639\,[1, 3]$	$1.046 \pm 0.210\,[1, 2]$	$2.459 \pm 0.643\,[2, 4]$	$2.462 \pm 0.647\,[2, 4]$
distinct concepts	$1.175 \pm 0.448\,[1, 4]$	$1.019 \pm 0.138\,[1, 2]$	$2.182 \pm 0.469\,[1, 5]$	$2.180 \pm 0.470\,[1, 5]$
distinct roles	$0.166 \pm 0.394\,[0, 2]$	$0.018 \pm 0.132\,[0, 1]$	$0.183 \pm 0.408\,[0, 2]$	$0.188 \pm 0.414\,[0, 2]$
distinct constructors	$0.414 \pm 0.660\,[0, 3]$	$0.046 \pm 0.210\,[0, 1]$	$1.460 \pm 0.664\,[1, 4]$	$1.462 \pm 0.668\,[1, 4]$
concepts	$1.177 \pm 0.452\,[1, 4]$	$1.019 \pm 0.138\,[1, 2]$	$2.196 \pm 0.465\,[2, 5]$	$2.196 \pm 0.466\,[2, 5]$
roles	$0.173 \pm 0.419\,[0, 2]$	$0.018 \pm 0.132\,[0, 1]$	$0.191 \pm 0.432\,[0, 2]$	$0.197 \pm 0.440\,[0, 2]$
constructors	$0.439 \pm 0.718\,[0, 3]$	$0.046 \pm 0.210\,[0, 1]$	$1.485 \pm 0.720\,[1, 4]$	$1.489 \pm 0.726\,[1, 4]$
complement	$0.089 \pm 0.300\,[0, 2]$	$0.009 \pm 0.095\,[0, 1]$	$0.098 \pm 0.312\,[0, 2]$	$0.096 \pm 0.312\,[0, 2]$
with complement	2701	292	2993	2903
intersection	$0.087 \pm 0.304\,[0, 3]$	$0.009 \pm 0.097\,[0, 1]$	$0.096 \pm 0.316\,[0, 3]$	$0.098 \pm 0.320\,[0, 3]$
with intersection	2570	303	2873	2953
union	$0.090 \pm 0.311\,[0, 3]$	$0.010 \pm 0.099\,[0, 1]$	$0.100 \pm 0.324\,[0, 3]$	$0.097 \pm 0.321\,[0, 3]$
with union	2658	319	2977	2893
universal restriction	$0.086 \pm 0.294\,[0, 2]$	$0.009 \pm 0.092\,[0, 1]$	$0.095 \pm 0.306\,[0, 2]$	$0.099 \pm 0.312\,[0, 2]$
with universal restriction	2627	272	2899	3028
existential restriction	$0.087 \pm 0.294\,[0, 2]$	$0.009 \pm 0.095\,[0, 1]$	$0.096 \pm 0.306\,[0, 2]$	$0.098 \pm 0.312\,[0, 2]$
with existential restriction	2673	291	2964	3014
top	$0.027 \pm 0.161\,[0, 2]$	$0.023 \pm 0.152\,[0, 2]$	$0.050 \pm 0.222\,[0, 2]$	$0.051 \pm 0.225\,[0, 2]$
with top	847	748	1574	1618
bottom	$0.026 \pm 0.160\,[0, 2]$	$0.023 \pm 0.150\,[0, 2]$	$0.049 \pm 0.219\,[0, 2]$	$0.048 \pm 0.217\,[0, 2]$
with bottom	831	737	1549	1510
queries per concept	$8.913 \pm 3.123\,[1, 22]$	$7.720 \pm 2.836\,[1, 22]$	$16.538 \pm 4.397\,[3, 35]$	$16.574 \pm 4.514\,[4, 39]$
queries per role	$44.847 \pm 27.843\,[15, 143]$	$4.896 \pm 3.881\,[1, 19]$	$49.619 \pm 31.202\,[19, 157]$	$51.169 \pm 33.033\,[18, 174]$

universal restriction number of universal restrictions $\forall$ in a single query;
with universal restriction number of queries with at least one universal restriction;
existential restriction number of existential restrictions $\forall$ in a single query;
with existential restriction number of queries with at least one existential restriction;
top number of occurrences of the top concept in a single query;
with top number of queries with a top concept;
bottom number of occurrences of the bottom concept in a single query;
with bottom number of queries with a bottom concept;
queries per concept number of queries containing a concept;
queries per role number of roles containing a concept;

We report statistics requiring aggregation in the following format: mean$\pm$standard deviation[minimum, maximum].

Table 12
Detailed statistics of the validation set of the synthetic data set

	subsumption			disjointness
	LHS	RHS	total	
count	7989	7989	7989	8011
depth	$1.421 \pm 0.649\,[1, 3]$	$1.049 \pm 0.216\,[1, 2]$	$2.470 \pm 0.653\,[2, 4]$	$2.459 \pm 0.645\,[2, 4]$
distinct concepts	$1.179 \pm 0.462\,[1, 4]$	$1.021 \pm 0.142\,[1, 2]$	$2.188 \pm 0.488\,[1, 5]$	$2.182 \pm 0.469\,[1, 5]$
distinct roles	$0.168 \pm 0.403\,[0, 2]$	$0.020 \pm 0.140\,[0, 1]$	$0.188 \pm 0.418\,[0, 2]$	$0.184 \pm 0.407\,[0, 2]$
distinct constructors	$0.420 \pm 0.664\,[0, 3]$	$0.049 \pm 0.216\,[0, 1]$	$1.469 \pm 0.668\,[1, 4]$	$1.461 \pm 0.669\,[1, 4]$
concepts	$1.182 \pm 0.465\,[1, 4]$	$1.021 \pm 0.143\,[1, 2]$	$2.203 \pm 0.479\,[2, 5]$	$2.197 \pm 0.465\,[2, 5]$
roles	$0.175 \pm 0.424\,[0, 2]$	$0.020 \pm 0.140\,[0, 1]$	$0.195 \pm 0.438\,[0, 2]$	$0.193 \pm 0.434\,[0, 2]$
constructors	$0.450 \pm 0.733\,[0, 3]$	$0.049 \pm 0.216\,[0, 1]$	$1.499 \pm 0.735\,[1, 4]$	$1.486 \pm 0.726\,[1, 4]$
complement	$0.093 \pm 0.307\,[0, 2]$	$0.008 \pm 0.090\,[0, 1]$	$0.101 \pm 0.318\,[0, 2]$	$0.095 \pm 0.306\,[0, 2]$
with complement	701	65	766	732
intersection	$0.089 \pm 0.316\,[0, 3]$	$0.012 \pm 0.108\,[0, 1]$	$0.101 \pm 0.331\,[0, 3]$	$0.095 \pm 0.318\,[0, 3]$
with intersection	640	94	734	704
union	$0.093 \pm 0.318\,[0, 3]$	$0.009 \pm 0.095\,[0, 1]$	$0.102 \pm 0.330\,[0, 3]$	$0.102 \pm 0.324\,[0, 3]$
with union	683	73	756	773
universal restriction	$0.090 \pm 0.301\,[0, 2]$	$0.011 \pm 0.102\,[0, 1]$	$0.100 \pm 0.315\,[0, 2]$	$0.098 \pm 0.310\,[0, 2]$
with universal restriction	680	84	764	754
existential restriction	$0.085 \pm 0.294\,[0, 2]$	$0.009 \pm 0.096\,[0, 1]$	$0.095 \pm 0.307\,[0, 2]$	$0.095 \pm 0.308\,[0, 2]$
with existential restriction	648	75	723	729
top	$0.025 \pm 0.156\,[0, 2]$	$0.023 \pm 0.150\,[0, 1]$	$0.048 \pm 0.217\,[0, 2]$	$0.047 \pm 0.214\,[0, 2]$
with top	195	184	373	370
bottom	$0.027 \pm 0.164\,[0, 2]$	$0.024 \pm 0.154\,[0, 1]$	$0.052 \pm 0.228\,[0, 2]$	$0.053 \pm 0.227\,[0, 2]$
with bottom	217	195	402	415
queries per concept	$2.496 \pm 1.359\,[1, 10]$	$2.263 \pm 1.235\,[1, 8]$	$4.200 \pm 2.032\,[1, 12]$	$4.200 \pm 2.023\,[1, 13]$
queries per role	$11.398 \pm 7.449\,[1, 36]$	$2.149 \pm 1.449\,[1, 8]$	$12.746 \pm 8.571\,[1, 44]$	$12.517 \pm 9.058\,[2, 52]$

Table 13
Detailed statistics of the test set of the synthetic data set

	subsumption			disjointness
count	19874	19874	19874	20126
depth	$1.419 \pm 0.647\ [1, 3]$	$1.049 \pm 0.215\ [1, 2]$	$2.468 \pm 0.651\ [2, 4]$	$2.464 \pm 0.644\ [2, 4]$
distinct concepts	$1.174 \pm 0.444\ [1, 4]$	$1.018 \pm 0.133\ [1, 2]$	$2.179 \pm 0.469\ [1, 5]$	$2.180 \pm 0.470\ [1, 5]$
distinct roles	$0.173 \pm 0.402\ [0, 2]$	$0.021 \pm 0.144\ [0, 1]$	$0.195 \pm 0.419\ [0, 2]$	$0.189 \pm 0.413\ [0, 2]$
distinct constructors	$0.420 \pm 0.670\ [0, 3]$	$0.049 \pm 0.215\ [0, 1]$	$1.469 \pm 0.674\ [1, 4]$	$1.464 \pm 0.664\ [1, 4]$
concepts	$1.176 \pm 0.448\ [1, 4]$	$1.018 \pm 0.134\ [1, 2]$	$2.194 \pm 0.461\ [2, 5]$	$2.195 \pm 0.464\ [2, 5]$
roles	$0.181 \pm 0.427\ [0, 2]$	$0.021 \pm 0.144\ [0, 1]$	$0.202 \pm 0.442\ [0, 2]$	$0.197 \pm 0.436\ [0, 2]$
constructors	$0.448 \pm 0.733\ [0, 3]$	$0.049 \pm 0.215\ [0, 1]$	$1.496 \pm 0.734\ [1, 4]$	$1.490 \pm 0.721\ [1, 4]$
complement	$0.090 \pm 0.304\ [0, 2]$	$0.009 \pm 0.095\ [0, 1]$	$0.100 \pm 0.316\ [0, 2]$	$0.098 \pm 0.311\ [0, 2]$
with complement	1695	180	1875	1875
intersection	$0.088 \pm 0.307\ [0, 3]$	$0.009 \pm 0.095\ [0, 1]$	$0.098 \pm 0.319\ [0, 3]$	$0.095 \pm 0.317\ [0, 3]$
with intersection	1632	181	1813	1781
union	$0.088 \pm 0.307\ [0, 3]$	$0.009 \pm 0.096\ [0, 1]$	$0.097 \pm 0.319\ [0, 3]$	$0.100 \pm 0.324\ [0, 3]$
with union	1606	183	1789	1883
universal restriction	$0.088 \pm 0.299\ [0, 2]$	$0.010 \pm 0.099\ [0, 1]$	$0.098 \pm 0.313\ [0, 2]$	$0.099 \pm 0.311\ [0, 2]$
with universal restriction	1667	198	1865	1909
existential restriction	$0.093 \pm 0.306\ [0, 2]$	$0.011 \pm 0.106\ [0, 1]$	$0.104 \pm 0.321\ [0, 2]$	$0.098 \pm 0.311\ [0, 2]$
with existential restriction	1750	225	1975	1890
top	$0.031 \pm 0.174\ [0, 2]$	$0.027 \pm 0.162\ [0, 1]$	$0.058 \pm 0.239\ [0, 2]$	$0.057 \pm 0.236\ [0, 2]$
with top	612	537	1126	1121
bottom	$0.032 \pm 0.178\ [0, 2]$	$0.025 \pm 0.157\ [0, 1]$	$0.057 \pm 0.237\ [0, 2]$	$0.057 \pm 0.237\ [0, 2]$
with bottom	635	500	1120	1123
queries per concept	$10.789 \pm 3.653\ [1, 25]$	$9.374 \pm 3.275\ [2, 23]$	$20.048 \pm 5.336\ [6, 41]$	$20.329 \pm 5.163\ [8, 41]$
queries per role	$55.565 \pm 33.345\ [28, 174]$	$6.934 \pm 5.958\ [1, 30]$	$62.387 \pm 38.611\ [32, 204]$	$61.452 \pm 37.867\ [26, 194]$