Neural Class Expression Synthesis[*]

N'Dah Jean Kouagou[0000−0002−4217−897X], Stefan Heindorf[0000−0002−4525−6865], Caglar Demir[0000−0001−8970−3850], and Axel-Cyrille Ngonga Ngomo[0000−0001−7112−3516]

Paderborn University, Germany
{ndah.jean.kouagou, heindorf, caglar.demir, axel.ngonga}@upb.de

Abstract. Many applications require explainable node classification in knowledge graphs. Towards this end, a popular "white-box" approach is class expression learning: Given sets of positive and negative nodes, class expressions in description logics are learned that separate positive from negative nodes. Most existing approaches are search-based approaches generating many candidate class expressions and selecting the best one. However, they often take a long time to find suitable class expressions. In this paper, we cast class expression learning as a translation problem and propose a new family of class expression learning approaches which we dub neural class expression synthesizers. Training examples are "translated" into class expressions in a fashion akin to machine translation. Consequently, our synthesizers are not subject to the runtime limitations of search-based approaches. We study three instances of this novel family of approaches based on LSTMs, GRUs, and set transformers, respectively. An evaluation of our approach on four benchmark datasets suggests that it can effectively synthesize high-quality class expressions with respect to the input examples in approximately one second on average. Moreover, a comparison to state-of-the-art approaches suggests that we achieve better F-measures on large datasets. For reproducibility purposes, we provide our implementation as well as pretrained models in our public GitHub repository at https://github.com/dice-group/NeuralClassExpressionSynthesis

Keywords: Neural network · Concept learning · Class expression learning · Learning from examples.

1 Introduction

One of the most popular families of web-scale knowledge bases [16] is that of RDF knowledge bases equipped with an ontology in W3C's web ontology language

OWL [28]. Examples include DBpedia [3], Wikidata [40], and CaliGraph [17]. One means to implement ante-hoc explainable machine learning on these knowledge bases is class expression learning (also called concept learning) [13, 15, 24, 26, 34, 39]. Informally, class expression learning approaches learn a class expression that describes individuals provided as positive examples. Class expression learning has applications in several domains, including ontology engineering [25], biomedicine [27] and Industry 4.0 [2]. There exist three main learning settings in class expression learning: (1) positive and negative learning, (2) positive-only learning, and (3) class-inclusion learning [23]. This paper tackles setting (1).

Several methods have been proposed to solve class expression learning problems; the best are based on refinement operators [13, 20, 23, 24, 26, 34] and evolutionary algorithms [15]. A common drawback of these approaches is their lack of scalability. While the reasoning complexity of all learning approaches grows with the expressivity of the underlying description logic (DL) [19, 32], those based on refinement operators and evolutionary algorithms further suffer from the exploration of an infinite conceptual space for each learning problem [34]. Another inherent limitation of existing methods for class expression learning from examples is their inability to leverage previously solved problems—their algorithm always starts from scratch for each new learning problem.

In view of the large sizes of modern knowledge bases, e.g., DBpedia [3] and Wikidata [40], we propose a new family of approaches, dubbed **neural class expression synthesizers** (NCES), for web-scale applications of class expression learning. The fundamental hypothesis behind this family of algorithms is that one should be able to capture enough semantics from latent representations (e.g., embeddings) of examples to directly synthesize class expressions in a fashion akin to machine translation, i.e., without the need for costly exploration. This hypothesis is supported by the significant improvement in the performance of machine translation approaches brought about by neural machine translation (NMT) [7, 44]. NMT approaches translate from a source language to a target language by exploiting an intermediary representation of a text's semantics. NCES behave similarly but translate from the "language" of sets of positive/negative examples to the "language" of class expressions. We instantiate this new paradigm by implementing three NCES instances that target the description logic $\mathcal{ALC}$. We show that our NCES instances generate high-quality class expressions with respect to the given sets of examples while remaining scalable. In fact, NCES instances synthesize solutions for multiple learning problems at the same time as they accept batches of inputs. This makes NCES particularly fit for deployment in large-scale applications of class expression learning, e.g., on the web.

The rest of the paper is organized as follows: First, we present existing approaches for class expression learning and introduce the notations and prerequisites needed throughout the paper. Next, we describe the intuition behind NCES in detail and introduce three instantiations of this new family of algorithms. We then compare these instantiations with state-of-the-art approaches on four benchmark datasets. Finally, we discuss our results and draw conclusions from our experiments.

2   Related Work

Class expression learning has been of interest to many researchers in recent years. Of the proposed approaches, the most prominent include those based on evolutionary algorithms [15] and refinement operators [6, 13, 20, 24, 25, 36]. The state-of-the-art EvoLearner [15] initializes its population by random walks on the knowledge graph which are subsequently converted to description logic concepts, represented as abstract syntax trees. These concepts are further refined by means of mutation and crossover operations. EvoLearner outperformed approaches based on refinement operators such as CELOE and OCEL from the DL-Learner framework [15]. Previously, Lehmann and Hitzler [26] studied different properties that a refinement operator can have, then designed a refinement operator to learn class expressions in description logics. Their learning algorithm, CELOE [25], is implemented in DL-Learner [23] alongside OCEL and ELTL [6]. CELOE extends upon OCEL by using a different heuristic function and is currently regarded as the best class expression learning algorithm in DL-Learner. Although ELTL was designed for the lightweight description logic $\mathcal{EL}$, we include it in this study to check whether our generated learning problems can be solved in a simpler description logic. ECII [36] is a recent approach for class expression learning that does not use a refinement operator and only invokes a reasoner once for each run. This approach was designed to overcome the runtime limitations of refinement operator-based approaches. Other attempts to prune the search space of refinement operator-based approaches include DL-Focl [34]. It is a modified version of DL-Foil [13] that is quintessentially based on omission rates. DL-Focl uses techniques such as lookahead strategy and local memory to avoid reconsidering sub-optimal choices.

Even though existing approaches for class expression learning have shown promising results, most of them are search-based. As a result, these approaches often use entailment checks—which are hard to compute, see Ozaki [32]—or compute classification accuracies at each step of the search process. In contrast, NCES only require a refinement operator for generating training data but not at prediction time. Hence, NCES are particularly suitable for solving many different learning problems consisting of positive and negative examples on the same dataset. As the training process of our synthesizers only involves instance data embeddings and a vocabulary of atoms, they can be extended to more expressive description logics such as $\mathcal{ALCHIQ}(\mathcal{D})$.

3   Background

3.1   Notation

DL is short for description logic, and DNN stands for deep neural network. Unless otherwise specified, $\mathcal{K} = (TBox, ABox)$ is a knowledge base in $\mathcal{ALC}$, and $\mathcal{N}_I$ is the set of all individuals in $\mathcal{K}$. The $ABox$ consists of statements of the form $C(a)$ and $R(a, b)$, whereas the $TBox$ contains statements of the form $C \sqsubseteq D$, where C, D are concepts, R is a role, and a, b are individuals in $\mathcal{K}$. We

use the representation of OWL knowledge bases as sets of triples to compute embeddings of individuals, classes and roles. The conversion into triples is carried out using standard libraries such as RDFLib [21]. We then use the knowledge graph representation of the knowledge bases to compute embeddings, which are essential to our proposed approach (see Figure 1). The function $|.|$ returns the cardinality of a set. $\mathbb{1}$ denotes the indicator function, i.e., a function that takes two inputs and returns 1 if they are equal, and 0 otherwise. Let a matrix M and integers i, j be given. $M_{:,j}$, $M_{i,:}$, and M_{ij} represent the j-th column, the i-th row, and the entry at the i-th row and j-th column, respectively. Similar notations are used for higher-dimensional tensors.

We define the vocabulary $Vocab$ of a given knowledge base $\mathcal{K}$ to be the list of all atomic concepts and roles in $\mathcal{K}$, together with the following constructs in any fixed ordering: " " (white space), "." (dot), "$\sqcup$", "$\sqcap$", "$\exists$", "$\forall$", "$\neg$", "(", and ")", which are all referred to as atoms. $Vocab[i]$ is the atom at position i in $Vocab$. These constructs are used by NCES to synthesize class expressions in $\mathcal{ALC}$ (see Section 4 for details). Let C be a class expression, then $\bar{C}$ and $\hat{C}$ are the list (in the order they appear in C) and set of atoms in C, respectively.

3.2 Description Logics

Description logics [30] are a family of knowledge representation paradigms based on first-order logics. They have applications in several domains, including artificial intelligence, the semantic web, and biomedical informatics. In fact, the web ontology language, OWL, uses description logics to represent the terminological box of RDF ontologies. In this work, we focus on the description logic $\mathcal{ALC}$ ($\mathcal{A}$ttributive $\mathcal{L}$anguage with $\mathcal{C}$omplement) [37] because of its simplicity and expressiveness. The syntax and semantics of $\mathcal{ALC}$ are presented in Table 1.

Table 1. $\mathcal{ALC}$ syntax and semantics. $\mathcal{I}$ is an interpretation and $\Delta^{\mathcal{I}}$ its domain.

Construct	Syntax	Semantics
Atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Atomic role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Top concept	$\top$	$\Delta^{\mathcal{I}}$
Bottom concept	$\bot$	$\emptyset$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Existential role restriction	$\exists R.C$	$\{a^{\mathcal{I}} \in \Delta^{\mathcal{I}} / \exists b^{\mathcal{I}} \in C^{\mathcal{I}}, (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}\}$
Universal role restriction	$\forall R.C$	$\{a^{\mathcal{I}} \in \Delta^{\mathcal{I}} / \forall b^{\mathcal{I}}, (a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}} \Rightarrow b^{\mathcal{I}} \in C^{\mathcal{I}}\}$

3.3 Refinement Operators

Definition 1 ([26]). *Given a quasi-ordered space $(\mathcal{S}, \preceq)$, a downward (respectively upward) refinement operator on $\mathcal{S}$ is a mapping $\rho : \mathcal{S} \to 2^{\mathcal{S}}$ such that for all $C \in \mathcal{S}$, $C' \in \rho(C)$ implies $C' \preceq C$ (respectively $C \preceq C'$).*

A refinement operator can be finite, proper, redundant, minimal, complete, weakly complete or ideal. Note that some of these properties can be combined whilst others cannot [26]. For class expression learning in description logics, weakly complete, finite, and proper refinement operators are the most used.

3.4 Class Expression Learning

Definition 2. *Given a knowledge base $\mathcal{K}$, a target concept T, a set of positive examples $E^+ = \{e_1^+, e_2^+, \ldots, e_{n_1}^+\}$, and a set of negative examples $E^- = \{e_1^-, e_2^-, \ldots, e_{n_2}^-\}$, the learning problem is to find a class expression C such that for $\mathcal{K}' = \mathcal{K} \cup \{T \equiv C\}$, we have $\forall\ e^+ \in E^+\ \forall e^- \in E^-,\ \mathcal{K}' \models C(e^+)$ and $\mathcal{K}' \not\models C(e^-)$.*

Most existing approaches use hard-coded heuristics or refinement operators to search for the solution C. When an exact solution does not exist, an approximate solution in terms of accuracy or F-measure is to be returned by the approaches. In this work, we exploit the semantics embedded in latent representations of individuals (instance data) to directly synthesize C.

3.5 Knowledge Graph Embedding

A knowledge graph can be regarded as a collection of assertions in the form of subject-predicate-object triples (s, p, o). Embedding functions project knowledge graphs onto continuous vector spaces to facilitate downstream tasks such as link prediction [5], recommender systems [47], and structured machine learning [20]. Many embedding approaches for knowledge graphs exist [9, 41]. Some of them use only facts observed in the knowledge graph [4, 31]. Others leverage additional available information about entities and relations, such as textual descriptions [42, 45]. Most embedding approaches initialize each entity and relation with a random vector, matrix or tensor and learn the embeddings as an optimization problem. For example, TransE [5] represents entities and relations as vectors in the same space and aims to minimize the Euclidean distance between $s + p$ and o for each triple (s, p, o). In this work, we use ConEx [10] and TransE to evaluate NCES.

3.6 Permutation-Invariant Network Architectures for Set Inputs

We deal with set-structured input data as in 3D shape recognition [33], multiple instance learning [11], and few-shot learning [14, 38]. These tasks benefit from machine learning models that produce the same results for any arbitrary reordering of the elements in the input set. Another desirable property of these models is the ability to handle sets of arbitrary size.

In recent years, several approaches have been developed to meet the afore-mentioned requirements. The most prominent of these approaches include Deep Set [46] and Set Transformer [22]. The Deep Set architecture encodes each element in the input set independently and uses a pooling layer, e.g., averaging, to produce the final representation of the set. In contrast, the Set Transformer architecture uses a self-attention mechanism to represent the set, which allows pair-wise and even higher-order interactions between the elements of the input set. As a result, the Set Transformer architecture shows superior performance on most tasks compared to Deep Set with a comparable model size [22]. In this work, we hence use the Set Transformer architecture (more details in Section 4) and refer to [22] for a description of its building blocks: Multi-head Attention Block (MAB), Set Attention Block (SAB), Induced Set Attention Block (ISAB), and Pooling by Multi-head Attention (PMA).

4   Neural Class Expression Synthesis

In this section, we present our proposed family of approaches for class expression learning from examples. We start with a formal definition of the learning problem that we aim to solve, then present our proposed approach in detail.

4.1   Learning Problem

We adapt the classical definition of a learning problem (see Definition 2) to our setting of class expression synthesis (Definition 3).

Definition 3. *Given a knowledge base $\mathcal{K}$, a set of positive examples $E^+ = \{e_1^+, e_2^+, \ldots, e_{n_1}^+\}$, and a set of negative examples $E^- = \{e_1^-, e_2^-, \ldots, e_{n_2}^-\}$, the learning problem is to synthesize a class expression C in $\mathcal{ALC}$ using atoms (classes and roles) in $\mathcal{K}$ that (ideally) accurately classifies the provided examples.*

In theory, there can be multiple solutions to a learning problem under both Definition 2 and Definition 3; our NCES generate only one. Moreover, the solution computed by a concept learner might be an approximation, e.g., there might be some false positives and false negatives. NCES aim to obtain high values for accuracy and F-measure.

4.2   Learning Approach (NCES)

We propose the following recipe to implement the idea behind NCES. First, given a knowledge base over $\mathcal{ALC}$, convert it into a knowledge graph (see Subsection 3.1). Then, embed said knowledge graph into a continuous vector space using any state-of-the-art embedding model in the literature. In our experiments, we used two embedding models with different expressive power: ConEx which applies convolutions on complex-valued vectors, and TransE which projects entities and relations onto a Euclidean space and uses the Euclidean distance to model interactions. The computed embeddings are then used as features for a model able to take a set of embeddings as input and encode a sequence of atoms as output (see Figure 1).
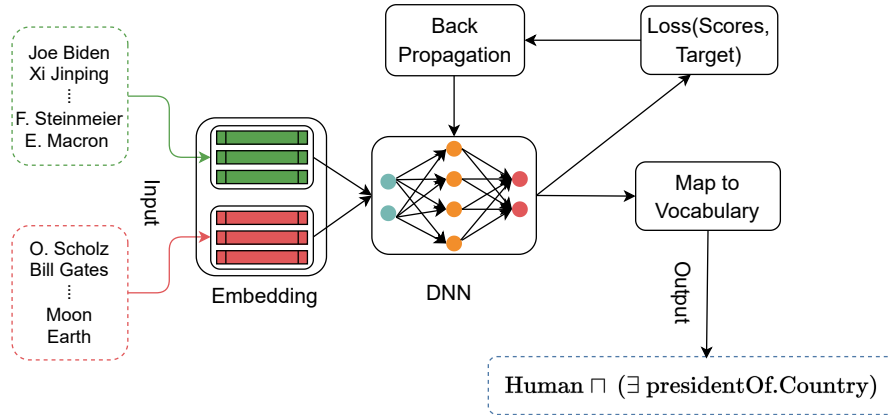
Fig. 1. NCES architecture. DNN stands for deep neural network that produces a sequence of tokens in the vocabulary (e.g., a sequence-to-sequence or a set-to-sequence model). The input consists of positive examples (upper left, dotted green box) and negative examples (bottom left, dotted red box).

Neural Network Architectures We conduct our experiments using the following network architectures: the Long Short-Term Memory (LSTM) [18], the Gated Recurrent Unit (GRU) [8], and the Set Transformer [22]. The latter is known to be permutation equivariant while the two others are not. Nonetheless, LSTM and GRU can handle set inputs as long as an ordering is defined since they deal well with sequential data [49, 50]. In this work, we use the default ordering (the order in which we received the data) of the elements in each set during the data generation process (see Section 5.1).

Recurrent Networks (LSTM and GRU) We use two recurrent layers followed by three linear layers with the `relu` activation function and a batch normalization layer. A recurrent neural network produces a sequence of n hidden states h_i ($i=1,\ldots,n$) for each input sequence of length n. In this work, we are concerned with a sequence of n_1 positive examples and a sequence of n_2 negative examples which are processed separately with the same network

$$h_1^{pos},\ldots,h_{n_1}^{pos} = RNN(x_{pos}); \ h_1^{neg},\ldots,h_{n_2}^{neg} = RNN(x_{neg}) \qquad (1)$$

where x_{pos} and x_{neg} are the sequences of embeddings of positive and negative examples, respectively. *RNN* is a two-layer LSTM or GRU network. The hidden state vectors of the two sets of examples are summed separately, then concatenated and fed to a sequence of 3 linear layers:

$$h_{pos} := \sum_{t=1}^{T_1} h_t^{pos}; \ h_{neg} := \sum_{t=1}^{T_2} h_t^{neg}; \ h := \text{Concat}(h_{pos}, h_{neg}); \qquad (2)$$

$$O = W_3(\text{bn}(W_2 f(W_1 h + b_1) + b_2)) + b_3. \qquad (3)$$

Here, f is the `relu` activation function, bn is a batch normalization layer, and $W_1, b_1, W_2, b_2, W_3, b_3$ are trainable weights.

Set Transformer This architecture comprises an encoder *Enc* and a decoder *Dec*, each with 4 attention heads. The encoder is a stack of two ISAB layers with $m = 32$ inducing points, and the decoder is composed of a single PMA layer with one seed vector $(k = 1)$, and a linear layer. As in the previous paragraph, the sets of positive and negative examples for a given class expression are first encoded separately using the encoder. The outputs are then concatenated row-wise and fed to the decoder:

$$O_{pos} = Enc(x_{pos}); \; O_{neg} = Enc(x_{neg}); \tag{4}$$
$$O = Dec(\text{Concat}(O_{pos}, O_{neg})). \tag{5}$$

Although the encoder captures interactions intra-positive and intra-negative examples separately, the decoder further captures interactions across the two sets of examples from the concatenated features through self-attention. This demonstrates the representational power of the Set Transformer model for our set-structured inputs for class expression synthesis.

The output O from 3 and 5 is reshaped into a $(1 + |Vocab|) \times L$ matrix, where L is the length of the longest class expression that NCES instances can generate. These scores allow us to compute the loss (see Equation 6) and update model weights through gradient descent during training.

Loss We train our NCES instances using the loss function $\mathcal{L}$ defined by

$$\mathcal{L}(x, y) = -\frac{1}{NL} \sum_{i=1}^{N} \sum_{j=1}^{L} \log \left(\frac{\exp(x_{i,y_{ij},j})}{\sum_{c=1}^{C} \exp(x_{i,c,j})} \right), \tag{6}$$

where N is the size of the minibatch, C is the number of classes, $x \in \mathbb{R}^{N \times C \times L}$ is the minibatch of predicted class scores for each position in the target sequence of atoms, and $y \in \mathbb{N}^{N \times L}$ is the minibatch of actual class indices. Minimizing $\mathcal{L}$ constrains the model to assign a high score to the entry corresponding to the correct token $(\exp(x_{i,y_{ij},j}) \approx 1)$ while keeping the remaining scores relatively low $(\sum_{c=1, c \neq y_{ij}}^{C} \exp(x_{i,c,j}) \approx 0)$. In this work, $C = 1 + |Vocab|$, where the additional $+1$ accounts for the special token "PAD" that we used to pad all class expressions to the same length. Contrarily to some works that omit this special token when computing the loss, we use it as an ordinary token during training. This way, we can generate class expressions more efficiently at test time with a single forward pass in the model, then strip off the generated tokens after the special token. To avoid exploding gradients and accelerate convergence during training, we adopt the gradient clipping technique [48].

Learning Metrics Apart from the loss function, we introduce two accuracy measures to quantify how well neural networks learn during training: soft accuracy

and hard accuracy. The former only accounts for the correct selection of the atoms in the target expression, while the latter additionally measures the correct ordering of the selected atoms. Formally, let T and P be the target and predicted class expressions, respectively. Recall the notation $\bar{C}$ and $\hat{C}$ introduced in Section 3.1 for any class expression C. The soft (Acc_s) accuracy and hard accuracy (Acc_h) are defined as follows:

$$Acc_s(T,P) = \frac{|\hat{T} \cap \hat{P}|}{|\hat{T} \cup \hat{P}|}; \ \ Acc_h(T,P) = \frac{\sum_{i=1}^{\min(l_1,l_2)} \mathbb{1}(\bar{T}[i], \bar{P}[i])}{\max(l_1, l_2)}, \tag{7}$$

where l_1 and l_2 are the lengths of $\bar{T}$ and $\bar{P}$, respectively.

Class Expression Synthesis We synthesize class expressions by mapping the output scores O (see Equations 3 and 5) to the vocabulary. More specifically, we select the highest-scoring atoms in the vocabulary for each position along the sequence dimension:

$$\mathrm{id}_j = \underset{c \in \{1,\ldots,C\}}{\arg\max} \ O_{c,j} \text{ for } j = 1, \ldots, L, \tag{8}$$

$$synthesized\_atom_j = Vocab[\mathrm{id}_j]. \tag{9}$$

Model Ensembling Ensemble learning has proven to be one of the most robust approaches for tasks involving complex noisy data [12, 35]. In this work, we combine class expression synthesizers' predictions post training by averaging the predicted scores. Specifically, given the output scores $O_i \in \mathbb{R}^{C \times L}$ ($i = 1, 2, 3$) as defined in 3 and 5 for the three models LSTM, GRU, and Set Transformer, we consider four different ensemble models: three pairwise ensemble models, and one global ensemble model (LSTM, GRU, and Set Transformer are combined). Formally, the ensemble scores are computed as:

$$O = \frac{\sum_{i \in \mathcal{I}} O_i}{|\mathcal{I}|} \text{ with } \mathcal{I} \subseteq \{1, 2, 3\} \text{ and } |\mathcal{I}| \geq 2. \tag{10}$$

Then, the synthesized expression is constructed following Equations 8 and 9 using the average scores O.

5   Evaluation

5.1   Experimental Setup

Datasets We evaluated our proposed approach on the Carcinogenesis [43], Mutagenesis [43], Semantic Bible[1], and the Vicodi [29] knowledge bases. Carcinogenesis and Mutagenesis are knowledge bases about chemical compounds and how they relate to each other. The Semantic Bible knowledge base describes each named

---

[1] `https://www.semanticbible.com/ntn/ntn-overview.html`

Table 2. Detailed information about the datasets used for evaluation. |LPs| is the number of learning problems in the test set.

Dataset	|Ind.|	|Classes|	|Prop.|	|TBox|	|ABox|	|Train|	|LPs|	|Vocab|
Carcinogenesis	22,372	142	4	144	74,223	10,982	111	157
Mutagenesis	14,145	86	5	82	47,722	5,333	54	102
Semantic Bible	724	48	29	56	3,106	3,896	40	88
Vicodi	33,238	194	10	204	116,181	18,243	175	215

object or thing in the New Testament, categorized according to its class, including God, groups of people, and locations. The Vicodi knowledge base was developed as part of a funded project and describes European history. The statistics of each of the knowledge bases are given in Table 2.

Training and Test Data Construction We generated class expressions of different forms from the input knowledge base using the recent refinement operator by Kouagou et al. [20] that was developed to efficiently generate numerous class expressions to serve as training data for concept length prediction in $\mathcal{ALC}$. The data that we generate is passed to the filtering process, which discards any class expression C such that an equivalent but shorter class expression D was not discarded. Note that each class expression comes with its set of instances, which are computed using the fast closed-world reasoner based on set operations described in [15]. These instances are considered positive examples for the corresponding class expression; negative examples are the rest of the individuals in the knowledge base. Next, the resulting data is randomly split into training and test sets; we used the discrete uniform distribution for this purpose. To ensure that our approach is scalable to large knowledge bases, we introduce a hyper-parameter $n = n_1 + n_2$ that represents the total number of positive and negative examples we sample for each class expression to be learned by NCES. Note that n is fixed for each knowledge base, and it depends on the total number of individuals.

Evaluation Metrics We measure the quality of a predicted class expression in terms of accuracy and F-measure with respect to the positive/negative examples. Note that we cannot expect to exactly predict the target class expression in the test data since there can be multiple equivalent class expressions.

Hyper-parameter Optimization for NCES We employed random search on the hyper-parameter space since it often yields good results while being computationally more efficient than grid search [1]; the selected values—those with the best results—are reported in Table 3. In the table, it can be seen that most knowledge bases share the same optimal values of hyper-parameters: the minibatch size N, the number of training epochs *epochs*, the optimizer *opt.*,

Table 3. Hyper-parameter settings per dataset. Recall that m is the number of inducing points in the Set Transformer model, and n is the number of examples.

Dataset	*epochs*	*opt.*	*lr*	*d*	*N*	*L*	*n*	*m*	*gc*
Carcinogenesis	300	Adam	0.001	40	256	48	1,000	32	5
Mutagenesis	300	Adam	0.001	40	256	48	1,000	32	5
Semantic Bible	300	Adam	0.001	40	256	48	362	32	5
Vicodi	300	Adam	0.001	40	256	48	1,000	32	5

Table 4. Model size and training time. The training time is in minutes.

	Carcinogenesis		Mutagenesis		Semantic Bible		Vicodi	
	|Params.|	Time	|Params.|	Time	|Params.|	Time	|Params.|	Time
NCES_{LSTM}	1,247,136	31.50	906,576	16.94	819,888	6.65	1,606,272	50.82
NCES_{GRU}	1,192,352	21.61	851,792	12.28	765,104	5.39	1,551,488	34.15
NCES_{ST}	1,283,104	40.82	942,544	21.36	855,856	7.98	1,642,240	66.19

the learning rate lr, the maximum output sequence length L, the number of embedding dimensions d, the number of inducing points m, and the gradient clipping value gc. Although we may increase n for very large knowledge bases, $n = \min(\frac{|\mathcal{N}_I|}{2}, 1000)$ appears to work well with our evaluation datasets. This suggests that one can effortlessly find fitting hyper-parameters for new datasets.

Hardware and Training Time We trained our chosen NCES instances on a server with 1TB of RAM and an NVIDIA RTX A5000 GPU with 24 GB of RAM. Note that during training, approximately 8GB of the 1TB RAM is currently used by NCES. As search-based approaches do not require a GPU for class expression learning, we used a 16-core Intel Xeon E5-2695 with 2.30GHz and 16GB RAM to run all approaches (including NCES post training) for class expression learning on the test set. The number of parameters and training time of each NCES instance are reported in Table 4. From the table, we can observe that NCES instances are lightweight and can be trained within a few hours on medium-size knowledge bases. Note that training is only required once per knowledge base.

5.2   Results

Syntactic Accuracy Our neural class expression synthesizers were trained for 300 epochs on each knowledge base. In Figure 2, we only show the hard accuracy curves during training due to space constraints. The rest of the training curves can be found on our GitHub repository. The curves in Figure 2 suggest that NCES instances train fast with an exponential growth in accuracy within the first 10 epochs. All models achieve over 95% syntactic accuracy on large knowledge bases (Carcinogenesis, Mutagenesis, and Vicodi). On the smallest knowledge base, Semantic Bible, we observe that NCES_{ST} drops in performance

as it only achieves 88% accuracy during training. On the other side, $NCES_{GRU}$ and $NCES_{LSTM}$ tend to overfit the training data. This suggests that NCES instances are well suited for large datasets. We validate this hypothesis through the quality of the synthesized solutions on the test set (see Table 5).
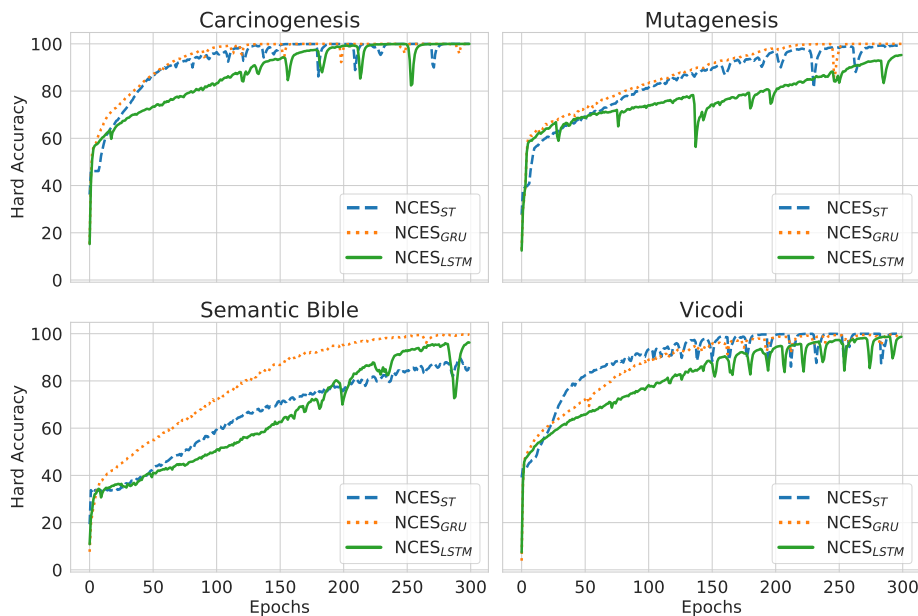

Fig. 2. Training accuracy curves.

Comparison to State-of-the-art Approaches We compare our approach against EvoLearner, CELOE, ECII, ELTL. The maximum execution time for CELOE and EvoLearner was set to 300 seconds per learning problem while ECII and ELTL were executed with their default settings, as they do not have the maximum execution time parameter in their original implementation. From Table 5, we can observe that our approach (with ensemble prediction) significantly outperforms all other approaches in runtime on all datasets, and in F-measure on Carcinogenesis and Vicodi. Table 6 shows that NCES performs slightly better with ConEx embeddings than TransE embeddings except on the Carcinogenesis dataset. The standard deviation of NCES's prediction time is 0 because it performs batch predictions, i.e., it predicts solutions for all learning problems at the same time. The prediction time is averaged across learning problems and is therefore the same for each learning problem. We used the Wilcoxon Rank Sum test with a significance level of 5% and the null hypothesis that the compared quantities per dataset are from the same distribution. The best search-based approaches

Table 5. Evaluation results per approach and dataset. NCES uses ConEx embeddings. The star (*) indicates statistically significant differences between the best search-based and the best synthesis-based approaches. ↑ indicates that the higher is better, and ↓ indicates that the lower is better. Underlined values are the second best.

	$\mathbf{F_1}$ (%) ↑			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
CELOE	37.92±44.25	82.95±33.48	**93.18±17.52***	35.66±42.06
ELTL	13.35±25.84	28.81±34.44	42.77±38.46	16.70±33.31
ECII	15.74±27.82	27.14±31.64	33.97±37.71	43.66±36.13
EvoLearner	91.48±14.30	**93.27±12.95**	91.88±10.14	92.74±10.28
$NCES_{LSTM}$	82.21±29.29	81.47±27.77	72.32±34.37	72.35±35.34
$NCES_{GRU}$	89.51±25.24	78.24±30.99	52.37±39.86	86.60±26.84
$NCES_{ST}$	90.32±25.12	80.55±34.33	72.95±38.68	77.75±37.16
$NCES_{ST+GRU}$	96.77±12.72	89.50±26.09	84.32±26.82	92.91±20.14
$NCES_{ST+LSTM}$	96.72±12.96	89.19±26.16	80.78±28.30	90.91±21.57
$NCES_{GRU+LSTM}$	94.51±15.39	81.49±30.71	75.91±32.48	88.28±24.17
$NCES_{ST+GRU+LSTM}$	**97.06±13.06***	91.39±22.91	87.11±24.05	**95.51±12.14***

	Accuracy (%) ↑			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
CELOE	66.88±24.87	94.39±11.68	98.17± 4.95	85.38±17.71
ELTL	24.96±32.86	38.01±39.26	47.31±38.22	33.35±41.88
ECII	25.95±35.98	31.99±38.36	30.35±37.91	76.73±35.22
EvoLearner	99.72± 1.44	**99.35±2.03**	**98.89±3.45***	**99.27±5.13**
$NCES_{LSTM}$	97.92±11.19	98.04± 6.81	89.83±23.59	94.03±20.77
$NCES_{GRU}$	99.32± 4.16	98.02± 4.03	80.67±30.80	98.10±11.43
$NCES_{ST}$	97.16±14.33	90.71±25.49	84.19±32.72	89.90±27.89
$NCES_{ST+GRU}$	99.03± 9.28	97.27±11.75	92.38±21.39	97.11±12.57
$NCES_{ST+LSTM}$	99.26± 5.69	96.93±12.17	90.27±21.60	95.78±16.46
$NCES_{GRU+LSTM}$	**99.90±0.31**	98.49± 5.35	87.16±27.44	96.86±14.99
$NCES_{ST+GRU+LSTM}$	99.04± 9.28	98.39± 6.45	94.70±17.83	97.48±11.71

	Runtime (sec.) ↓			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
CELOE	239.58±132.59	92.46±125.69	135.30±139.95	289.95±103.63
ELTL	23.81±1.47	15.19±12.50	4.12±0.11	299.14±202.21
ECII	22.93±2.63	18.11±4.93	6.45±1.42	37.94±28.25
EvoLearner	54.73±25.86	48.00±31.38	17.16±9.20	213.78±81.03
$NCES_{LSTM}$	0.16±0.00	0.19±0.00	0.08±0.00	0.13±0.00
$NCES_{GRU}$	0.15±0.00	0.18±0.00	0.08±0.00	0.06±0.00
$NCES_{ST}$	**0.08±0.00***	**0.11±0.00***	**0.07±0.00***	**0.04±0.00***
$NCES_{ST+GRU}$	0.16±0.00	0.25±0.00	0.11±0.00	0.09±0.00
$NCES_{ST+LSTM}$	0.23±0.00	0.23±0.00	0.11±0.00	0.11±0.00
$NCES_{GRU+LSTM}$	0.24±0.00	0.32±0.00	0.13±0.00	0.17±0.00
$NCES_{ST+GRU+LSTM}$	0.27±0.00	0.31±0.00	0.15±0.00	0.15±0.00

Table 6. Evaluation results using TransE embeddings.

	$\mathbf{F_1}$ (%) ↑			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
$NCES_{LSTM}$	84.52±27.01	76.45±34.22	59.63±34.98	79.06±30.07
$NCES_{GRU}$	87.00±27.43	77.77±36.36	65.71±33.51	79.81±33.31
$NCES_{ST}$	86.66±30.13	79.12±34.23	68.14±37.17	78.33±36.35
$NCES_{ST+GRU}$	97.47± 9.63	91.05±21.11	75.36±32.88	89.27±25.09
$NCES_{ST+LSTM}$	92.85±21.88	90.01±19.50	76.26±34.07	89.37±24.46
$NCES_{GRU+LSTM}$	91.83±21.03	81.22±33.05	76.17±31.53	87.68±24.54
$NCES_{ST+GRU+LSTM}$	97.56±11.55	91.24±21.27	85.82±24.85	90.05±23.67

	Accuracy (%) ↑			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
$NCES_{LSTM}$	97.40±14.19	96.59±11.27	84.10±24.91	97.17±11.98
$NCES_{GRU}$	97.68±12.82	95.20±15.08	82.85±29.44	95.86±16.27
$NCES_{ST}$	93.59±23.06	93.11±21.15	82.47±33.34	91.70±25.01
$NCES_{ST+GRU}$	99.35± 5.52	99.07± 2.61	85.62±27.76	96.45±15.17
$NCES_{ST+LSTM}$	98.11±11.18	99.00± 2.69	88.95±23.69	95.05±18.01
$NCES_{GRU+LSTM}$	98.91± 9.46	96.08±13.64	89.25±24.54	97.37±12.60
$NCES_{ST+GRU+LSTM}$	98.99± 9.45	98.90± 4.70	93.53±18.24	95.01±18.53

	Runtime (sec.) ↓			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
$NCES_{LSTM}$	0.09±0.00	0.14±0.00	0.06±0.00	0.12±0.00
$NCES_{GRU}$	0.05±0.00	0.15±0.00	0.06±0.00	0.13±0.00
$NCES_{ST}$	0.04±0.00	0.09±0.00	0.05±0.00	0.05±0.00
$NCES_{ST+GRU}$	0.08±0.00	0.18±0.00	0.08±0.00	0.11±0.00
$NCES_{ST+LSTM}$	0.09±0.00	0.16±0.00	0.08±0.00	0.11±0.00
$NCES_{GRU+LSTM}$	0.15±0.00	0.22±0.00	0.10±0.00	0.15±0.00
$NCES_{ST+GRU+LSTM}$	0.14±0.00	0.22±0.00	0.11±0.00	0.14±0.00

(CELOE and EvoLearner) only outperform NCES instances (including ensemble models) on the smallest datasets (Semantic Bible and Mutagenesis). The reason for this is that deep learning models are data-hungry and often fail to generalize well on small datasets. Our approach is hence well suited for large knowledge bases where search-based approaches are prohibitively slow.

5.3   Discussion

The hypothesis behind this work was that high-quality class expressions can be synthesized directly out of training data, i.e., without the need for an extensive search. Our results clearly undergird our hypothesis. While NCES is outperformed by CELOE and EvoLearner on small datasets, it achieves the best performance on Carcinogenesis with over 5% absolute improvement in F-measure. This large difference is due to the fact that most search-based approaches fail to find any

suitable solution for some learning problems. For example, the first learning problem on the Vicodi knowledge base is (`Disaster` $\sqcup$ `Military-Organisation`) $\sqcap$ (¬`Engineer`). The solutions computed by each of the approaches are as follows: CELOE: `Flavour` $\sqcap$ (¬`Battle`) $\sqcap$ (¬`Person`) [F_1: 1.95%], ELTL: `Flavour` $\sqcap$ ($\exists$ `related.`($\exists$ `related.Role`)) [F_1: 0%], ECII: `Organisation` $\sqcup$ ¬`VicodiOI` [F_1: 13.16%], EvoLearner: `Military-Organisation` $\sqcap$ `Military-Organisation` [F_1: 73.17%], and NCES$_{ST+GRU}$[2]: (`Disaster` $\sqcup$ `Military-Organisation`) $\sqcap$ (¬`Engineer`) [F_1: 100%]. Here, our ensemble model NCES$_{ST+GRU}$ synthesized the exact solution, which does not appear in the training data of NCES, while the best search-based approach, EvoLearner, could only compute an approximate solution with an F-measure of 73.17%. On the other hand, CELOE, ECII, and ELTL failed to find any suitable solutions within the set timeout.

The scalability of the synthesis step of our approaches makes them particularly suitable for situations where many class expressions are to be computed for the same knowledge base. For example, taking into account the average training and inference time of the Set Transformer architecture, one can conjecture that the minimum number of learning problems from which the cost of deep learning becomes worthwhile is: 11 for NCES vs. CELOE, 25 for NCES vs. EvoLearner, 24 for NCES vs. ELTL, and 96 for NCES vs. ECII. These values are calculated by solving for n in $n \times T_{\texttt{algo\_learn}} > T_{\texttt{train}} + T_{\texttt{inference}}$, where $T_{\texttt{algo\_learn}}$, $T_{\texttt{train}}$, and $T_{\texttt{inference}}$ are the average learning time of a search-based approach, the training time, and the inference time of NCES, respectively.

6   Conclusion and Future Work

We propose a novel family of approaches for class expression learning, which we dub neural class expression synthesizers (NCES). NCES use neural networks to directly synthesize class expressions from input examples without requiring an expensive search over all possible class expressions. Given a set timeout per prediction, we showed that our approach outperforms all state-of-the-art search-based approaches on large knowledge bases. Taking training time into account, our approach is suitable for application scenarios where many concepts are to be learned for the same knowledge base. In future work, we will investigate means to transfer the knowledge acquired on one knowledge base to other knowledge bases. Furthermore, we plan to extend our approach to more expressive description logics such as $\mathcal{ALCHIQ}(\mathcal{D})$.

Bibliography

[1] Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)

[2] Bin, S., Westphal, P., Lehmann, J., Ngonga, A.: Implementing scalable structured machine learning for big data in the SAKE project. In: IEEE BigData, pp. 1400–1407, IEEE Computer Society (2017)

---

[2] Here, NCES uses ConEx embeddings

[3] Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: Dbpedia - A crystallization point for the web of data. J. Web Semant. **7**(3), 154–165 (2009)

[4] Bordes, A., Glorot, X., Weston, J., Bengio, Y.: A semantic matching energy function for learning with multi-relational data - application to word-sense disambiguation. Mach. Learn. **94**(2), 233–259 (2014)

[5] Bordes, A., Usunier, N., García-Durán, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: NIPS, pp. 2787–2795 (2013)

[6] Bühmann, L., Lehmann, J., Westphal, P.: Dl-learner - A framework for inductive learning on the semantic web. J. Web Semant. **39**, 15–24 (2016)

[7] Cho, K., van Merrienboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches. In: SSST@EMNLP, pp. 103–111, ACL (2014)

[8] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: EMNLP, pp. 1724–1734, ACL (2014)

[9] Dai, Y., Wang, S., Xiong, N.N., Guo, W.: A survey on knowledge graph embedding: Approaches, applications and benchmarks. Electronics **9**(5), 750 (2020)

[10] Demir, C., Ngomo, A.N.: Convolutional complex knowledge graph embeddings. In: ESWC, LNCS, vol. 12731, pp. 409–424, Springer (2021)

[11] Dietterich, T.G., Lathrop, R.H., Lozano-Pérez, T.: Solving the multiple instance problem with axis-parallel rectangles. Artif. Intell. **89**(1-2), 31–71 (1997)

[12] Dong, X., Yu, Z., Cao, W., Shi, Y., Ma, Q.: A survey on ensemble learning. Frontiers of Computer Science **14**(2), 241–258 (2020)

[13] Fanizzi, N., d'Amato, C., Esposito, F.: DL-FOIL concept learning in description logics. In: ILP, LNCS, vol. 5194, pp. 107–121, Springer (2008)

[14] Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: ICML, Proceedings of Machine Learning Research, vol. 70, pp. 1126–1135, PMLR (2017)

[15] Heindorf, S., Blübaum, L., Düsterhus, N., Werner, T., Golani, V.N., Demir, C., Ngomo, A.N.: Evolearner: Learning description logics with evolutionary algorithms. In: WWW, pp. 818–828, ACM (2022)

[16] Heist, N., Hertling, S., Ringler, D., Paulheim, H.: Knowledge graphs on the web - an overview. In: Knowledge Graphs for eXplainable Artificial Intelligence, Studies on the Semantic Web, vol. 47, pp. 3–22, IOS Press (2020)

[17] Heist, N., Paulheim, H.: Uncovering the semantics of wikipedia categories. In: ISWC, LNCS, vol. 11778, pp. 219–236, Springer (2019)

[18] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)

[19] Konev, B., Ozaki, A., Wolter, F.: A model for learning description logic ontologies based on exact learning. In: AAAI, pp. 1008–1015, AAAI Press (2016)

[20] Kouagou, N.J., Heindorf, S., Demir, C., Ngomo, A.N.: Learning concept lengths accelerates concept learning in ALC. In: ESWC, LNCS, vol. 13261, pp. 236–252, Springer (2022)

[21] Krech, D.: Rdflib: A python library for working with rdf. Online https://github. com/RDFLib/rdflib (2006)

[22] Lee, J., Lee, Y., Kim, J., Kosiorek, A.R., Choi, S., Teh, Y.W.: Set transformer: A framework for attention-based permutation-invariant neural networks. In: ICML, Proceedings of Machine Learning Research, vol. 97, pp. 3744–3753, PMLR (2019)

[23] Lehmann, J.: Dl-learner: Learning concepts in description logics. J. Mach. Learn. Res. **10**, 2639–2642 (2009)

[24] Lehmann, J.: Learning OWL Class Expressions, Studies on the Semantic Web, vol. 6. IOS Press (2010)

[25] Lehmann, J., Auer, S., Bühmann, L., Tramp, S.: Class expression learning for ontology engineering. J. Web Semant. **9**(1), 71–81 (2011)

[26] Lehmann, J., Hitzler, P.: Concept learning in description logics using refinement operators. Mach. Learn. **78**(1-2), 203–250 (2010)

[27] Lehmann, J., Völker, J.: Perspectives on Ontology Learning, Studies on the Semantic Web, vol. 18. IOS Press (2014)

[28] McGuinness, D.L., Van Harmelen, F., et al.: Owl web ontology language overview. W3C recommendation **10**(10), 2004 (2004)

[29] Nagypál, G.: History ontology building: The technical view. Humanities, Computers and Cultural Heritage p. 207 (2005)

[30] Nardi, D., Brachman, R.J.: An introduction to description logics. In: Description Logic Handbook, pp. 1–40, Cambridge University Press (2003)

[31] Nickel, M., Tresp, V., Kriegel, H.: Factorizing YAGO: scalable machine learning for linked data. In: WWW, pp. 271–280, ACM (2012)

[32] Ozaki, A.: Learning description logic ontologies: Five approaches. where do they stand? Künstliche Intell. **34**(3), 317–327 (2020)

[33] Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: CVPR, pp. 652–660 (2017)

[34] Rizzo, G., Fanizzi, N., d'Amato, C.: Class expression induction as concept space exploration: From dl-foil to dl-focl. Future Gener. Comput. Syst. **108**, 256–272 (2020)

[35] Sagi, O., Rokach, L.: Ensemble learning: A survey. WIREs Data Mining Knowl. Discov. **8**(4) (2018)

[36] Sarker, M.K., Hitzler, P.: Efficient concept induction for description logics. In: AAAI, pp. 3036–3043, AAAI Press (2019)

[37] Schmidt-Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artif. Intell. **48**(1), 1–26 (1991)

[38] Snell, J., Swersky, K., Zemel, R.S.: Prototypical networks for few-shot learning. In: NIPS, pp. 4077–4087 (2017)

[39] Tran, T., Ha, Q., Hoang, T., Nguyen, L.A., Nguyen, H.S.: Bisimulation-based concept learning in description logics. Fundam. Informaticae **133**(2-3), 287–303 (2014)

[40] Vrandecic, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Commun. ACM **57**(10), 78–85 (2014)

[41] Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge graph embedding: A survey of approaches and applications. IEEE Trans. Knowl. Data Eng. **29**(12), 2724–2743 (2017)

[42] Wang, Z., Li, J.: Text-enhanced representation learning for knowledge graph. In: IJCAI, pp. 1293–1299, IJCAI/AAAI Press (2016)

[43] Westphal, P., Bühmann, L., Bin, S., Jabeen, H., Lehmann, J.: Sml-bench - A benchmarking framework for structured machine learning. Semantic Web **10**(2), 231–245 (2019)

[44] Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., Dean, J.: Google's neural machine translation system: Bridging the gap between human and machine translation. CoRR **abs/1609.08144** (2016)

[45] Xie, R., Liu, Z., Jia, J., Luan, H., Sun, M.: Representation learning of knowledge graphs with entity descriptions. In: AAAI, pp. 2659–2665, AAAI Press (2016)

[46] Zaheer, M., Kottur, S., Ravanbakhsh, S., Póczos, B., Salakhutdinov, R., Smola, A.J.: Deep sets. In: NIPS, pp. 3391–3401 (2017)

[47] Zhang, F., Yuan, N.J., Lian, D., Xie, X., Ma, W.: Collaborative knowledge base embedding for recommender systems. In: KDD, pp. 353–362, ACM (2016)

[48] Zhang, J., He, T., Sra, S., Jadbabaie, A.: Why gradient clipping accelerates training: A theoretical justification for adaptivity. arXiv preprint arXiv:1905.11881 (2019)

[49] Zhao, H., Sun, S., Jin, B.: Sequential fault diagnosis based on LSTM neural network. IEEE Access **6**, 12929–12939 (2018)

[50] Zulqarnain, M., Ghazali, R., Ghouse, M.G., et al.: Efficient processing of gru based on word embedding for text classification. JOIV **3** (2019)