Fast Neuro-Symbolic Approaches for Class Expression Learning

N'Dah Jean Kouagou

May 16th, 2025
Version: Final

Heinz Nixdorf Institute, Department of Computer Science
Data Science Group (DICE)

Doctoral Dissertation

Fast Neuro-Symbolic Approaches for Class Expression Learning

A dissertation presented by

N'Dah Jean Kouagou

to the
Faculty of Computer Science, Electrical Engineering and Mathematics
of
Paderborn University
in partial fulfillment of the requirements for the degree of
Dr. rer. nat.

1. Reviewer Prof. Dr. Axel-Cyrille Ngonga Ngomo
Heinz Nixdorf Institute, Department of Computer Science
Paderborn University

2. Reviewer Prof. Dr. Paul Groth
Informatics Institute
University of Amsterdam

Supervisor Prof. Dr. Axel-Cyrille Ngonga Ngomo

May 16th, 2025

N'Dah Jean Kouagou
Fast Neuro-Symbolic Approaches for Class Expression Learning
Doctoral Dissertation, May 16th, 2025
Reviewers: Prof. Dr. Axel-Cyrille Ngonga Ngomo and Prof. Dr. Paul Groth
Supervisor: Prof. Dr. Axel-Cyrille Ngonga Ngomo

Paderborn University
Data Science Group (DICE)
Heinz Nixdorf Institute, Department of Computer Science
Faculty of Computer Science, Electrical Engineering and Mathematics
Warburger Str. 100
33098 and Paderborn

Abstract

With the swift progress of current AI systems' capabilities, there is a pressing need for explainable AI methods to understand the behavior of these systems. To this end, class expression learning in description logics holds enormous potential. A class expression provides a clear description of why a given example is classified as positive, and is hence a white-box model. Most existing approaches for class expression learning are search-based methods which generate many candidate class expressions and select the one with the highest classification score. While these approaches often perform well on small datasets, their search space is infinite and its exploration becomes arduous on large datasets even for a single learning problem. In this thesis, we develop several neuro-symbolic approaches to tackle class expression learning at scale. Our first approach CLIP uses neural networks to learn concept lengths, and utilizes trained concept length predictors to solve learning problems efficiently. Neural class expression synthesizers (NCES) tackle class expression learning in $\mathcal{ALC}$ in a fashion akin to machine translation, and support sophisticated computing hardware such as GPUs. NCES2 extends NCES to the description logic $\mathcal{ALCHIQ}^{(\mathcal{D})}$, integrates an embedding model for end-to-end training, and employs a data augmentation technique to enhance generalization to unseen learning problems. Finally, ROCES uses iterative sampling to improve the robustness of neural class expression synthesizers to changes in the number of input examples. Experimental results on benchmark datasets suggest that our proposed approaches are significantly faster than the state of the art (up to 10,000$\times$ with NCES, NCES2 and ROCES) while being highly competitive in predictive performance.

Kurzfassung

Angesichts der rasanten Entwicklung der Fähigkeiten moderner KI-Systeme besteht ein dringender Bedarf an erklärbaren KI-Methoden, um das Verhalten dieser Systeme zu verstehen. Lernen von Klassenausdrücken in Beschreibungslogiken birgt enormes Potenzial dafür. Ein Klassenausdruck liefert eine klare Beschreibung, warum ein bestimmtes Beispiel als positiv oder negativ eingestuft wird und ist somit ein White-Box-Modell. Bei den meisten existierende Ansätzen zum Lernen von Klassenausdrücken handelt es sich um suchbasierte Methoden, die viele verschiedene Klassenausdrücke generieren und denjenigen mit der höchsten Klassifizierungspunktzahl auswählen. Während diese Ansätze bei kleinen Datensätzen oft gut funktionieren, ist ihr Suchraum unendlich groß und dessen Erkundung wird gerade bei großen Datensätzen zeitaufwändig. In dieser Arbeit entwickeln wir mehrere neurosymbolische Ansätze, um das Lernen von Klassenausdrücken in großem Maßstab zu bewältigen. Unser erster Ansatz CLIP verwendet neuronale Netze, um Konzeptlängen zu lernen, und nutzt trainierte Konzeptlängenprädiktoren, um Lernprobleme effizient zu lösen. Neuronale Klassenausdrucksynthetisierer (NCES) gehen das Lernen von Klassenausdrücken in $\mathcal{ALC}$ ähnlich wie maschinelle Übersetzung an und unterstützen anspruchsvolle Computerhardware wie GPUs. NCES2 erweitert NCES auf die Beschreibungslogik $\mathcal{ALCHIQ}^{(\mathcal{D})}$, integriert ein Einbettungsmodell für das End-to-End-Training und setzt eine Datenerweiterungstechnik ein, um die Generalisierung auf ungesehene Lernprobleme zu verbessern. Schließlich verwendet ROCES iteratives Sampling, um die Robustheit von neuronalen Klassenausdrucksynthetisierern gegenüber Änderungen in der Anzahl der Eingabebeispiele zu verbessern. Experimentelle Ergebnisse auf Benchmark-Datensätzen deuten darauf hin, dass die von uns vorgeschlagenen Ansätze deutlich schneller sind als der Stand der Technik (bis zu 10.000$\times$ mit NCES, NCES2 und ROCES) und gleichzeitig in der Vorhersageleistung vergleichbar gut abschneiden.

Publications and Declaration of Authorship

Below, we list publications and notable mentions of this thesis' author. All conference, journal, workshop and poster publications are peer reviewed. Publications that are part of this thesis are marked with a ♦ symbol.

Conference Articles

1. ♦ **N'Dah Jean Kouagou**, Stefan Heindorf, Caglar Demir, Axel-Cyrille Ngonga Ngomo: *Learning Concept Lengths Accelerates Concept Learning in $\mathcal{ALC}$*, ESWC, 2022. [107]

   Declaration of Authorship. The initial research idea was proposed by Axel-Cyrille Ngonga Ngomo and further developed by N'Dah Jean Kouagou. N'Dah Jean Kouagou implemented the approach, conducted experiments and analyzed their results. N'Dah Jean Kouagou also wrote the manuscript and revised it with Stefan Heindorf, Caglar Demir, and Axel-Cyrille Ngonga Ngomo. A brief summary of this publication can be found in Section 1.2.1. The complete publication is presented in Chapter 4.

2. ♦ **N'Dah Jean Kouagou**, Stefan Heindorf, Caglar Demir, Axel-Cyrille Ngonga Ngomo: *Neural Class Expression Synthesis*, ESWC, 2023. [113]

   Declaration of Authorship. The original research contributions were developed by N'Dah Jean Kouagou and discussed with Axel-Cyrille Ngonga Ngomo and Stefan Heindorf. N'Dah Jean Kouagou implemented the approach, conducted experiments and analyzed their results. N'Dah Jean Kouagou also wrote the manuscript and revised it with Stefan Heindorf, Caglar Demir, and

Axel-Cyrille Ngonga Ngomo. A brief summary of this publication can be found in Section 1.2.2. The complete publication is presented in Chapter 5.

3. ♦ **N'Dah Jean Kouagou**, Stefan Heindorf, Caglar Demir, Axel-Cyrille Ngonga Ngomo: *Neural Class Expression Synthesis in $\mathcal{ALCHIQ}(\mathcal{D})$*, ECML PKDD, 2023. [109]

   Declaration of Authorship. The original research contributions were developed by N'Dah Jean Kouagou and discussed with Axel-Cyrille Ngonga Ngomo and Stefan Heindorf. N'Dah Jean Kouagou implemented the approach, conducted experiments and analyzed their results. N'Dah Jean Kouagou also wrote the manuscript and revised it with Stefan Heindorf, Caglar Demir, and Axel-Cyrille Ngonga Ngomo. A brief summary of this publication can be found in Section 1.2.3. The complete publication is presented in Chapter 6.

4. ♦ **N'Dah Jean Kouagou**, Stefan Heindorf, Caglar Demir, Axel-Cyrille Ngonga Ngomo: *ROCES: Robust Class Expression Synthesis in Description Logics via Iterative Sampling*, IJCAI, 2024. [111]

   Declaration of Authorship. The original research contributions were developed by N'Dah Jean Kouagou and discussed with Axel-Cyrille Ngonga Ngomo and Stefan Heindorf. N'Dah Jean Kouagou implemented the approach, conducted experiments and analyzed their results. N'Dah Jean Kouagou also wrote the manuscript and revised it with Stefan Heindorf, Caglar Demir, and Axel-Cyrille Ngonga Ngomo. A brief summary of this publication can be found in Section 1.2.4. The complete publication is presented in Chapter 7.

Journal Articles

1. **N'Dah Jean Kouagou**, P.G. Dlamini, S.M. Simelane: *On the multi-domain compact finite difference relaxation method for high dimensional chaos: The nine-dimensional Lorenz system*, Alexandria Engineering Journal, 2020. [106]

2. **N'Dah Jean Kouagou**, Arif Yilmaz, Michel Dumontier, Axel-Cyrille Ngonga Ngomo: *Discovering the unknown: Improving rule mining via embedding-based link prediction*, arXiv, 2024. [112]
3. Caglar Demir, Alkid Baci, **N'Dah Jean Kouagou**, Leonie Nora Sieger, Stefan Heindorf, Simon Bin, Lukas Blübaum, Alexander Bigerl, Axel-Cyrille Ngonga Ngomo: *Ontolearn—A Framework for Large-scale OWL Class Expression Learning in Python*, Journal of Machine Learning Research, 2025.
4. Arnab Sharma, **N'Dah Jean Kouagou**, Axel-Cyrille Ngonga Ngomo: *Resilience in Knowledge Graph Embeddings*, Transactions on Graph Data and Knowledge (under review), 2025.

Book Chapters

1. Axel-Cyrille Ngonga Ngomo, Caglar Demir, **N'Dah Jean Kouagou**, Stefan Heindorf, Nikolaos Karalis, Alexander Bigerl: *Class Expression Learning with Multiple Representations*, Compendium of Neurosymbolic Artificial Intelligence, 2023. [155]

Workshop Articles, Posters & Others

1. ♦ **N'Dah Jean Kouagou**, Stefan Heindorf, Caglar Demir, Axel-Cyrille Ngonga Ngomo: *Neural Class Expression Synthesis*, NeSy, 2023. [108]
2. **N'Dah Jean Kouagou**, Caglar Demir, Hamada M. Zahera, Adrian Wilke, Stefan Heindorf, Jiayi Li, Axel-Cyrille Ngonga Ngomo: *Universal Knowledge Graph Embeddings*, WWW'24 Companion Proceedings. [110]
3. Caglar Demir, **N'Dah Jean Kouagou**, Arnab Sharma, Axel-Cyrille Ngonga Ngomo: *Inference over Unseen Entities, Relations and Literals on Knowledge Graphs*, ECAI–CompAI Workshop, 2024. [51]
4. Ding Ning, Varvara Vetrova, Karin Bryan, Yun Sing Koh, Andreas Voskou, **N'Dah Jean Kouagou**, Arnab Sharma: *Diving Deep: Forecasting Sea Surface Temperatures and Anomalies*, ECML PKDD–Discovery Track, 2024. [160]
5. Louis Mozart KAMDEM TEYOU, Luke Friedrichs, **N'Dah Jean Kouagou**, Caglar Demir, Yasir Mahmood, Stefan Heindorf, Axel-Cyrille Ngonga Ngomo: *Neural Reasoning for Robust Concept Learning*, IJCAI (under review), 2025.
6. Quannian Zhang, Michael Röder, Nikit Srivastava, **N'Dah Jean Kouagou**, Axel-Cyrille Ngonga Ngomo: *Neural Reasoning for Robust Concept Learning*, IJCAI (under review), 2025.

Awards and Notable Mentions

1. Kaggle Featured Competition & NIPS 2023 Winner Award.[1]
   N'Dah Jean Kouagou: *1st Place Winner at the Open Problems–Single-Cell Perturbations Challenge*, NIPS 2023.
2. Kaggle Best Solution Write-up Award—Judges Award.[2]
   N'Dah Jean Kouagou: *Best Solution Write-up for the Open Problems–Single-Cell Perturbations challenge*, NIPS 2023.
3. ECML PKDD 2024 Challenge Winner[3]
   N'Dah Jean Kouagou and Arnab Sharma: *2nd Place Winner at the Diving Deep–Forecasting Sea Surface Temperatures and Anomalies Challenge*, ECML PKDD 2024.

[1] https://www.kaggle.com/competitions/open-problems-single-cell-perturbations
[2] https://www.kaggle.com/competitions/open-problems-single-cell-perturbations/discussion/459258
[3] https://divingdeepecml.github.io/

Acknowledgments

I would like to express my sincere gratitude to my supervisor Prof. Dr. Axel-Cyrille Ngonga Ngomo, whose guidance and unwavering support were essential to the realization of the scientific contributions presented herein. It has been a privilege to witness and share his genuine passion for research. I am grateful for the opportunity to have been his student.

My sincere gratitude goes to Prof. Dr. Paul Groth and the rest of my thesis committee—Prof. Dr. Anni-Yasmin Turhan, Prof Dr. Stefan Sauer, and Dr. Caglar Demir—for their insightful comments and encouragement.

I would also like to thank my wife, my parents, and my friends. Without their unconditional love and support, this thesis would not have been possible.

My deepest gratitude to my collaborators Dr. Stefan Heindorf, Dr. Caglar Demir, Dr. Hamada Zahera, Dr. Michael Röder, Dr. Arnab Sharma, Dr. Yasir Mahmood, Dr. Pamela Heidi Douglas, and again Prof. Dr. Axel-Cyrille Ngonga Ngomo for our scientific discussions. I also extend my gratitude to my colleagues at the DICE Group, who created a supportive and enjoyable work environment.

Parts of this work have been supported by 1.) the European Union's Horizon 2020 research and innovation programme within the project KnowGraphs under the Marie Skłodowska-Curie grant No 860801, 2.) the European Union's Horizon Europe research and innovation programme within the project ENEXA under the grant No 101070305, 3.) the Ministry of Culture and Science of North Rhine-Westphalia (MKW NRW) within the project SAIL under the grant No NW21-059D, and 4.) the German Research Foundation (DFG): TRR 318/1 2021–438445824.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

There is a pressing need for explainable AI methods to understand the behavior of AI systems [161, 164, 249]. For example, the EU AI Act[1] mandates transparency for AI systems [136]. Moreover, Article 22 of the GDPR[2] grants data subjects the right to understand how AI systems use their data to reach conclusions. In this thesis, we contribute to the need for explainable AI methods by developing scalable neuro-symbolic approaches for class expression learning in description logics. The thesis comprises 8 chapters. In Chapter 1, we further motivate our research, identify relevant gaps in existing literature, formulate our research questions, and highlight our contributions. In Chapters 2 and 3, we present the prerequisites needed throughout the thesis, and describe related works, respectively. In Chapters 4—7, we present our research contributions. In Chapter 8, we summarize the thesis and outline future work.

1.1 Motivation

Artificial intelligence (AI) has revolutionized our approach to problem-solving and the way we interact with the rest of the world. Whether in our everyday life, e.g., product purchase with AI-powered recommendation systems [140, 245], social media [7, 96], smart homes [59, 76], or complex application domains such as biomedical sciences [2, 13], or autonomous driving [73, 238], AI has become an integral part of our existence. Let us delve into the last two application domains. In medical diagnosis, AI systems analyze large amounts of data to assist healthcare professionals in identifying diseases at early stages [42, 218]. For example, deep learning models such as convolutional neural networks (CNNs) have shown remarkable success in interpreting medical images like X-rays, magnetic resonance imaging (MRI) or computerized tomography (CT) scans [93, 179, 186, 192, 203]. These models can detect subtle patterns that may be invisible to humans, which yields an enormous potential for earlier intervention and improved patient outcomes. AI has also been successfully applied in protein structure prediction, e.g., AlphaFold2 [99].

[1] https://artificialintelligenceact.eu
[2] https://gdpr-info.eu/art-22-gdpr. Accessed on January 28th, 2025.

In fact, two authors of AlphaFold2 have been awarded the 2024 Nobel Prize in Chemistry[3] for their valuable work in protein structure prediction. With accurate protein structure prediction, scientists can now improve their understanding of chemical reactions in proteins, and generate real-time 3D images of proteins with specific properties, e.g., the ability to decompose plastic [60, 175].

Autonomous driving [70, 73, 125, 238] is another critical application of AI. Here, complex algorithms concert to interpret real-time data from various sources (e.g., cameras, sensors), helping vehicles to navigate safely in dynamic environments. Most of these algorithms leverage advanced machine learning techniques—particularly deep learning—to enhance perception, decision-making, and control systems [73, 151]. The aim is that, with these techniques, autonomous vehicles identify obstacles, predict the behavior of other road users, and make informed decisions on navigation paths, e.g., accelerate or break to avoid a collision. Moreover, the integration of technologies such as LiDAR (light detection and ranging) and radar can further support AI algorithms and enhance the overall situational awareness of a vehicle [137, 240].

Despite its immense potential across several applications, AI faces significant barriers to effective adoption, particularly in critical domains such as healthcare. This is because, in contrast to symbolic AI methods which use human comprehensible semantics and logic-based reasoning [95], the best-performing AI approaches currently rely primarily on deep neural networks [9, 21]. Deep neural networks have complex prediction mechanisms which are difficult to interpret, leading to their classification as "black boxes". However, as pointed out in [212], it is hard to trust AI when we cannot understand how it reaches its conclusions. Transparency in AI is therefore essential for its effective adoption in various application domains. The search for these transparent AI methods has lead to the ever-growing interest to reconcile symbolic AI with deep learning [68, 84, 95], resulting in a plethora of methods for explainable AI (XAI) [3, 11, 12, 74, 130, 145, 211]. Some of these methods are post-hoc, i.e., they aim to explain existing black-box models [39, 98, 114, 115, 204], while others are self-explained , i.e., ante-hoc [181].

This thesis contributes to advancing both the fields of learning from examples [27, 43, 62, 119, 177, 185] and explainable AI [3, 11, 12, 74, 130, 145, 211], with a focus on scalability. More specifically, we develop scalable neuro-symbolic approaches for class expression learning (CEL) in description logics (DLs). Our approaches combine the strengths of deep neural networks, which are parallelizable and can handle

[3] `https://www.nobelprize.org/prizes/chemistry/2024/press-release`. Accessed on October 14th, 2024.

different data formats (e.g., image, text, audio), and symbolic AI methods, which have clear semantics and can produce interpretable results, e.g., a class expression in description logics. A class expression is human-readable and interpretable[4] and is hence a white-box model. For instance, a class expression to describe penguins is $(\texttt{Bird} \sqcap \texttt{CanSwim}) \sqcup (\texttt{Animal} \sqcap \texttt{HasWings} \sqcap \neg\texttt{CanFly})$, which stands for *birds that can swim or animals that have wings and cannot fly*.

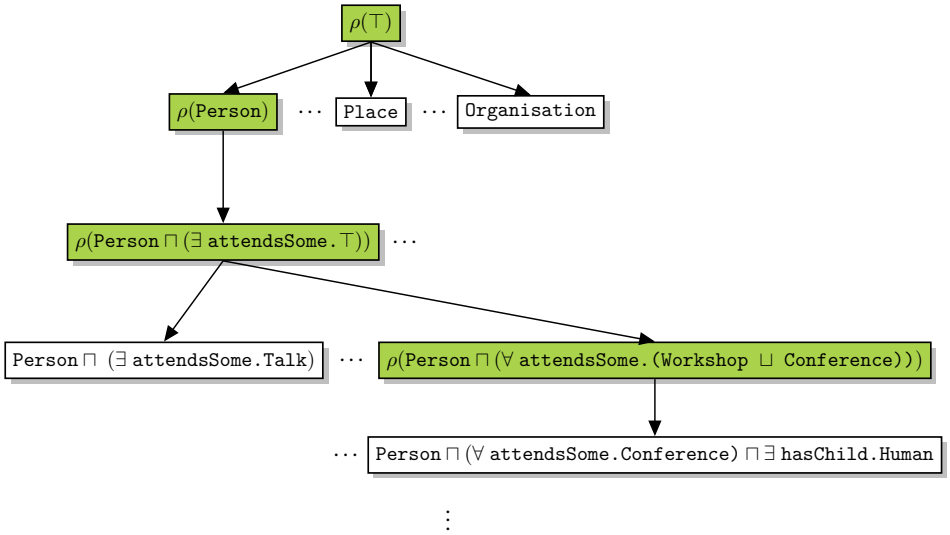


Figure 1.1: Search tree of refinement operator-based approaches for class expression learning. ρ is a downward refinement operator as defined in the next chapter. Green rectangles correspond to the selected concepts (which are considered the most promising according to some heuristic function) for further exploration. Horizontal and vertical suspension points indicate that the tree expands further horizontally and vertically, respectively.

Early approaches for CEL are search-based [82, 120, 121, 122, 177, 185]. Some of these approaches employ a refinement operator [15, 123] to construct an infinite conceptual space, and a heuristic function to traverse it. Figure 1.1 illustrates a search tree for refinement operator-based approaches. Given sets of positive and negative examples, the search process starts with the most general concept[5] $\top$ which is refined to yield a sequence of more specific concepts, each covering some part of positive examples and ruling out a number of negative examples. The most promising refinement according to a heuristic function is then selected for further exploration (see the concept "Person" in Figure 1.1). The search process continues until, e.g., a solution is found (a refinement which perfectly covers all positive

[4] Here, "interpretability" refers to the fact that the string representation of a class expression carries an intuitive meaning which is accessible to human readers.

[5] Some refinement operator-based approaches start with the bottom concept $\bot$ and use an upward refinement operator to iteratively construct a solution.

examples while ruling out all negative examples) or until the maximum number of iterations is reached[6]. While not based on refinement operators, EvoLearner [82] and ECII [185] employ a similar search strategy, i.e., they construct a search tree akin to the one in Figure 1.1. As such, given certain requirements on the quality of the solution sought (e.g., a solution must achieve at least 99% F_1 score), the search tree can grow indefinitely large. This makes search-based approaches time-inefficient and they often yield poor results on especially large datasets where they fail to find suitable solutions within a reasonable amount of time [109, 113].

There have been efforts to improve the scalability of search-based approaches for CEL, including DRILL [48] and NERO [47]. DRILL learns the search space traversal via deep Q-learning [144]. To this end, DRILL can either be pretrained in an unsupervised manner, e.g. by learning from example learning problems, or actively learn space traversal during a class expression learning task. In both cases, DRILL requires a refinement operator to construct its search space (akin to the one in Figure 1.1), a trainable heuristic function and a scoring function (an extension of CELOE's scoring function) to guide the search. Although DRILL outperforms some existing search-based approaches in terms of runtime, its reliance on a refinement operator and a scoring function during search can still constitute a drawback on especially large datasets.

NERO [47] learns permutation-invariant continuous vector representations for sets of positive/negative examples tailored towards predicting the F_1 scores of predefined class expressions with respect to input examples. Given sets of examples for a learning problem, NERO selects the class expression with the highest predicted score as solution. A clear limitation of NERO is that it cannot find solutions outside the predefined set, making it a sub-optimal approach especially for complex learning problems. The synthesis-based approaches presented in this thesis (see Chapters 5–7) overcome many of the aforementioned limitations. For instance, by synthesizing class expressions using atomic concept and role names as well as description logic-specific symbols (e.g., $\exists, \leq$), our approaches are not limited to any predefined set of class expressions, i.e., they are fully flexible w.r.t. the construction of a solution. The development of our approaches was guided by a number of research questions, which we present below.

[6] Note that there are other termination criteria which can be used, e.g., *a set timeout*, or *a minimum solution score*.

1.2 Research Questions and Contributions

To tackle class expression learning, several approaches have been proposed, most of which are refinement operator-based [120, 121, 122, 177]. A refinement operator-based approach builds a search tree whose nodes are concepts and whose edges represent hierarchies between those concepts. This tree is then traversed using heuristic functions which measure the likelihood of a given path leading to a solution (see Chapter 3 for details). As illustrated by Figure 1.1, this search tree can grow indefinitely on large datasets. This makes current refinement operator-based approaches computationally inefficient. Below, we formulate our first research question which aims to prune the search space of refinement operator-based approaches.

1.2.1 Concept Length Prediction

In our quest for ways to improve the scalability of refinement operator-based approaches, we envisage possible links between the length of a concept (as defined in Section 2.2) and its set of instances. For this, we ask ourselves the following question.

Research Question 1

Can we learn concept lengths in the description logic $\mathcal{ALC}$?

To answer this research question, we develop an approach to automatically generate numerous non-redundant class expressions with various lengths, each with a non-empty instance set referred to as the set of positive examples. We then use the closed-world assumption (CWA) [121, 174] to construct a set of negative examples for each generated class expression. Finally, we use knowledge graph embedding techniques (see Chapter 2) to compute continuous vector representations for examples and use them as input features for neural network architectures capable of predicting concept lengths. Experimental results on several benchmark datasets suggest that all of our chosen neural networks outperform a random model which gives preference to the most represented concept lengths on each dataset. Moreover, recurrent neural networks perform best at the task of concept length prediction, surpassing many other architectures, including multilayer perceptrons (MLPs) and convolutional neural networks (CNNs).

With the success of neural networks in concept length prediction, we were interested in knowing whether this could help accelerate class expression learning. This lead

to the development of our complete approach CLIP (concept learner with integrated length prediction) [107] which we detailedly present in Chapter 4. In short, we incorporate trained concept length predictors into refinement operator-based concept learners (CELOE [122] in our experiments), and compare the resulting algorithm to baselines, namely CELOE, ELTL, and OCEL [120]. Given sets of positive and negative examples for a learning problem, our approach CLIP first predicts the length of the target concept and uses it as the radius of the search space. More specifically, refinements whose lengths are above the predicted value are discarded during the search process. Experimental results suggest that CLIP is significantly faster (p-value < 0.05 with the Wilcoxon Signed Rank test) than base algorithms while being more accurate in most cases.

1.2.2 Neural Class Expression Synthesis

While CLIP showed significant improvements over search-based approaches for CEL, its reliance on a refinement operator can still be seen as a hindrance on real world datasets which are increasingly growing in size. For instance, the October 2023 crawl[7] of the web revealed that around 98 billion triple-formatted statements can be extracted. This number was only 82 billion in October 2021[8], indicating an increase of nearly 20 billion datapoints within 2 years. The pruning technique in CLIP might therefore not be sufficient on such datasets, especially when real-time solutions to learning problems are needed. To further mitigate the scalability issues of search-based approaches, we consider tackling class expression learning from a completely different angle, summarized in the following research question.

Research Question 2

Can we develop an approach for class expression learning in the description logic $\mathcal{ALC}$ that can compute solutions without a search process?

To answer this research question, we develop a novel family of approaches [113] (also see Chapter 5) dubbed synthesis-based approaches which compute a solution to a learning problem in a fashion akin to machine translation [35, 228]. In this way, sets of positive and negative examples are translated to class expressions in description logics without the need for a costly exploration. To achieve this goal, we implement neural network architectures able to encode sets of examples, e.g.,

[7] October 2023: `https://webdatacommons.org/structureddata/#results-2023-1` Accessed on October 24th 2024

[8] October 2021: `http://webdatacommons.org/structureddata/#results-2021-1` Accessed on October 24th 2024

Set-Transformer, and output a sequence of tokens which form a class expression. These network architectures utilize a vocabulary of tokens, which is fixed for each input knowledge base, and learn output token positions based on input examples by minimizing the cross-entropy loss, see Chapter 5 for details. Experimental results on benchmark datasets suggest that our neural class expression synthesizers (NCES) are highly accurate and significantly faster than search-based approaches for CEL (p-value < 0.05 with the Wilcoxon Signed Rank test) as they can compute solutions to learning problems within a fraction of a second. Moreover, NCES can solve multiple learning problems in parallel, and support sophisticated computing hardware such as GPUs.

1.2.3 Neural Class Expression Synthesis in $\mathcal{ALCHIQ}^{(\mathcal{D})}$

Our proposed family of synthesis-based approaches require pretrained embeddings for input knowledge bases, and can solve learning problems in $\mathcal{ALC}$. Although they perform well with the pretrained embeddings available for benchmark datasets, most real world knowledge bases do not have precomputed embeddings, and can be represented in more expressive description logics. We therefore consider another set of research questions to further improve our synthesis-based approaches.

Research Question 3

Is neural class expression synthesis possible in more expressive description logics, e.g., $\mathcal{ALCHIQ}^{(\mathcal{D})}$?

To extend neural class expression synthesizers to $\mathcal{ALCHIQ}^{(\mathcal{D})}$, we add new tokens to the vocabulary of NCES, e.g., "$\geq$" and "$\leq$" for cardinality and value restrictions, and "$^-$" for inverse roles. We also adopt an approach based on information gain to precompute data value thresholds [82] which we add to NCES's vocabulary to fully support $\mathcal{ALCHIQ}^{(\mathcal{D})}$.

To alleviate dependency on pretrained embeddings, we also investigate the integration of an embedding model into synthesis-based approaches (see the following research question) for end-to-end training and inference.

Research Question 4

Can we train an embedding model end-to-end with a neural synthesizer?

Here, we combine an embedding model and a neural synthesizer supporting the description logic $\mathcal{ALCHIQ}^{(\mathcal{D})}$ (as described above) into a unified learner which we

call NCES2 [109], see Chapter 6 for more details. During training, the embedding model receives a batch of triples representing factual knowledge, e.g., relationships between individuals in the input knowledge base, and the neural synthesizer receives a batch of class expressions with their corresponding examples. The embedding model provides current vector representations of examples to the neural synthesizer, and the loss from both components are backpropagated simultaneously. NCES2 also introduces a new data generation technique which improves the generalization capability of synthesis-based approaches to unseen learning problems. Experimentally, NCES2 shows a better predictive performance compared to the initial family of neural synthesizers (NCES) especially on complex learning problems, while being equally fast at inference time.

1.2.4 Robust Class Expression Synthesis via Iterative Sampling

During our investigation of the two research questions in Section 1.2.3, we realized that neural synthesizers are sensitive to small changes in input examples. In particular, their performance drops drastically at inference time when dealing with learning problems whose example sets are smaller than those they are trained on. However, in real world applications of class expression learning such as ontology engineering [122], a knowledge engineer may be interested in describing a few or many examples at a time. Consequently, a class expression learner should be able to handle learning problems with arbitrary numbers of examples, which is not the case with our previous neural synthesizers. We therefore consider the following research question which aims to improve the robustness[9] of neural class expression synthesizers with respect to changes in the number of input examples.

Research Question 5

How can we ensure that neural class expression synthesizers are robust to changes in the numbers of input examples?

To answer the above research question, we develop our approach ROCES [111], presented in Chapter 7. In ROCES, we propose a generalization of the classical learning problem which encourages class expression learners to solve learning problems by using cardinality-minimal sets of examples. We also propose a learning algorithm for synthesis-based approaches which combines interative sampling and

[9] Here, the term "robustness" refers to the ability of neural class expression synthesizers to compute correct solutions to learning problems with complete or few input examples.

gradient-based optimization to solve our generalized learning problem. Our experimental results suggest that ROCES significantly outperforms search- and previous synthesis-based approaches on learning problems with limited examples, while being highly competitive on full learning problems.

1.3 Thesis Outline

In this section, we present the structure of the thesis which comprises 8 chapters. In the current chapter, we motivate our research in class expression learning, identify limitations of existing approaches, and provide an overview of the scientific contributions made herein. The rest of the thesis is organized as follows:

Chapter 2 establishes the foundational knowledge required for a comprehensive understanding of this thesis.

Chapter 3 presents existing works in class expression learning and related tasks. These include the state of the art for class expression learning, e.g., CELOE [122], EvoLearner [82], and inductive logic programming approaches.

Chapter 4 presents our approach CLIP [107] which integrates concept length predictors to prune the search space of refinement operator-based approaches for class expression learning. The source code of CLIP is available on GitHub at `https://github.com/dice-group/LearnALCLengths`. CLIP is also implemented in the structured machine learning library Ontolearn (`https://github.com/dice-group/Ontolearn`).

Chapter 5 introduces NCES [113], our family of synthesis-based approaches (also called neural class expression synthesizers) for $\mathcal{ALC}$, which can compute a solution without a search process. NCES is implemented in the structured machine learning library Ontolearn (`https://github.com/dice-group/Ontolearn`), and experimental results reported herein can be reproduced at `https://github.com/dice-group/NeuralClassExpressionSynthesis`.

Chapter 6 describes NCES2 [109], an instance of synthesis-based approaches that supports the description logic $\mathcal{ALCHIQ}^{(\mathcal{D})}$ and integrates an embedding model for end-to-end training and inference. The source code of NCES2 is available on GitHub at `https://github.com/dice-group/NCES2`. NCES2 is also implemented in Ontolearn.

Chapter 7 presents ROCES [111], our approach to improve the robustness of class expression learners in general, and that of neural class expression synthesizers in particular. The source code of ROCES is available on GitHub at `https://github.com/dice-group/ROCES`. ROCES is also integrated into Ontolearn.

Chapter 8 concludes the thesis and outlines future work.

2 Background

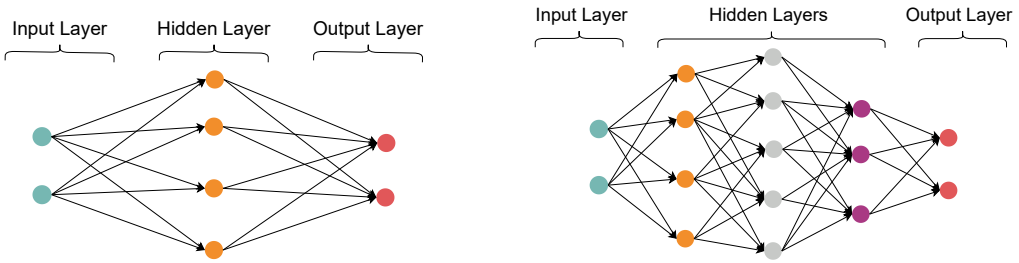
This thesis leverages findings from different well-established fields of research, including machine learning, deep learning, description logics, and knowledge graphs. In this chapter, we present the prerequisites needed throughout the thesis so it is self-contained.

2.1 Machine Learning

2.1.1 Overview

Machine learning (ML) is a branch of artificial intelligence that focuses on the development of statistical algorithms able to learn from data, generalize to unseen data, and perform tasks autonomously [10, 14]. For example, artificial neural networks (ANNs) or simply neural networks (NNs), which are inspired by the (human) brain structure, have revolutionized the fields of natural language processing, computer vision, speech recognition, and many others [8, 103, 236], surpassing the good old-fashioned AI (GOFAI) systems which are mainly based on symbolic representations [134, 182]. Apart from single-layered networks like Hopfield networks [87, 88], the most prominent neural networks are composed of an input layer, one (or many) hidden[1] layer(s), and an output layer. Each layer consists of one or many artificial neurons which carry messages (typically real numbers) and eventually transfer them to other neurons. A neural network with many hidden layers is called a deep neural network, and the learning process is then referred to as "deep learning" [72, 117]. In Figure 2.1a, we show the picture of a neural network, and in Figure 2.1b, the picture of a deep neural network. In both figures, nodes represent artificial neurons, and arrows indicate connections between nodes in terms of message reception and transfer. In a forward pass, a node aggregates information from (some of) its predecessors and pass it onto (some of) its successors. During training, information about the incurred loss travels backward from the output to the input layer (via chain rule in gradient computation) to help update the state of

[1] Hidden layers are intermediate layers, i.e., they locate between the input and output layers.



(a) Simple (non-deep) neural network **(b)** Deep neural network

Figure 2.1: Structure of a neural network

each node. Popular optimization algorithms used to train neural networks include Adam [104] and its variants, ADOPT [199], SGD [178], and RMSProp [202].

Unlike simple neural networks, deep learning models are well-known for their ability to learn from unlabelled data (i.e., unsupervised learning) where they automatically extract important features and characteristics from the input data. Large language models (LLMs), e.g., GPT [1], LLama [205], Gemma [201], and DeepSeek [75, 131] are examples of deep neural networks that are trained in an auto-regressive manner (i.e., predict the next token given a sequence of preceding tokens) using unlabelled text data.

2.1.2 Neural Network Architectures

Multi-layer Perceptron

A multi-layer perceptron (MLP) is a neural network architecture consisting of multiple layers transmitting information in a single direction [19, 167]. Such a neural network can be defined by its weight matrices and bias vectors:

$$f_\theta(x) = \sigma_n(W_n(\dots\sigma_2(W_2(\sigma_1(W_1x + b_1)) + b_2) + b_n), \tag{2.1}$$

where x is an input vector, $\theta = \{(W_i, b_i)\}_{i=1}^n$ weight matrices and bias vectors, and $\{\sigma_i\}_{i=1}^n$ non-linear activation functions. Note that activation functions may not be applied to the output of certain layers. For instance, in regression tasks the outputs of the last layer are usually not activated. One limitation of MLP is its inability to natively handle multi-dimensional inputs such as images. To process such inputs, a flattening operation is necessary, which often leads to high-dimensional input vectors and costly computations. To mitigate this issue, other types of neural network architectures have been proposed, including convolutional neural networks,

recurrent neural networks, and transformers. These complex architectures often employ MLP as the head (last layer) producing the final outputs.

Convolutional Neural Network

A basic convolutional neural network (CNN) [6, 227, 234] consists of a convolution kernel, and a linear layer. Formally, given an input vector x, a simple convolutional neural network with kernel ω computes its output as:

$$f_\theta(x) = W\,\mathrm{vec}(\omega * x) + b, \tag{2.2}$$

where $*$ denotes a convolution operation, $\theta = \{\omega, W, b\}$ trainable parameters, and $\mathrm{vec}(.)$ a flattening function that transforms a high-dimensional array into a one-dimensional vector.[2] The shape of the kernel ω depends on the input. As an example, ω is a vector in $\mathbb{R}^k$ for one-dimensional inputs $x \in \mathbb{R}^d$ such as audio signals. For RGB images (which are three-dimensional), the kernel ω is three-dimensional. Convolutional neural networks are well known for their capability to process high-dimensional data efficiently. In Figure 2.2, we show how a convolution operation is performed on 2D (two-dimensional) inputs such as grey images. Basically, the kernel matrix is multiplied element-wise with a portion of the input and the coefficients of the resulting sub-matrix are added up. This process is repeated on different portions of the input to produce the final output. It is important noting that activation functions may be applied to intermediate outputs (e.g., between a convolution and a linear layer) or to the final output.

The main hyperparameters of a convolutional neural network are the *kernel size* (also known as *filter size*), the *stride*, and the *padding*. The kernel size determines the receptive field of the neural network; roughly, this refers to the size of a portion of the input that is captured by the network at a time. For instance, a larger kernel size captures more information in the input, but it increases the size of the network. The stride determines the step size by which the kernel moves while operating on the input. Finally, padding is a technique used to augment the input and reduce information loss at its edges. This involves adding zeros around the edges of the input. In Figure 2.2, the kernel size is 2×2, the stride is $(1, 1)$, and the padding is $(0, 0)$. These hyperparameters also control the size of the output. Along each

[2] One-dimensional in this case refers to vectors in $\mathbb{R}^d$, as opposed to real numbers.

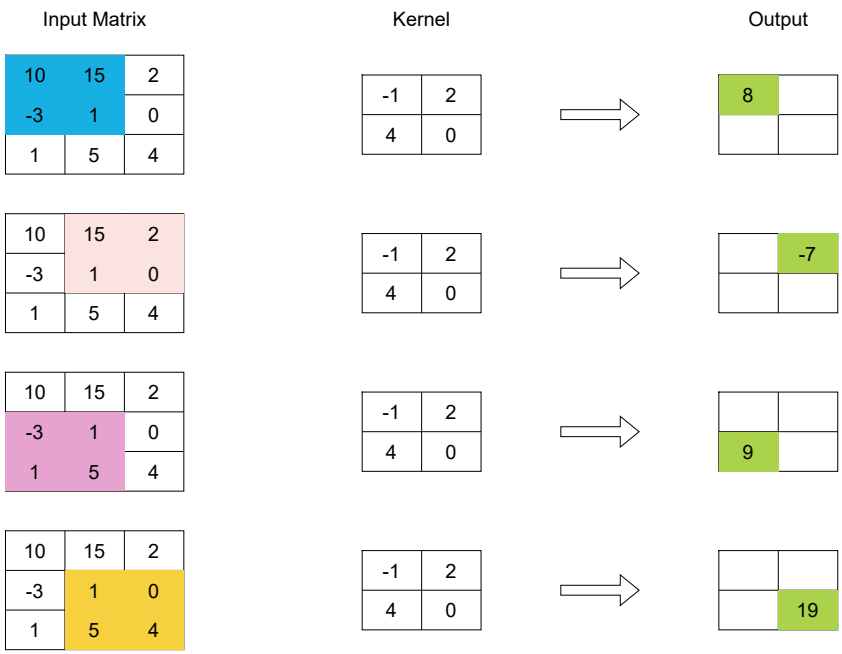


Figure 2.2: Convolution operation on 2D inputs

dimension D, the relationship between the size of the input and the size of the output is given by:

$$\text{Output}^{(D)} = \frac{\text{Input}^{(D)} + 2 \times \text{Padding}^{(D)} - \text{Kernel\_Size}^{(D)}}{\text{Stride}^{(D)}} + 1. \tag{2.3}$$

In complex convolutional neural networks, e.g., Unet [179], multiple convolution layers are composed to capture intricate patterns in the input.

Recurrent Neural Network

Recurrent neural networks (RNNs) [85, 142, 188] are a family of neural architectures able to process sequential data, e.g., times series, text, videos (sequence of frames). The fundamental component of an RNN is a recurrent unit, which embodies a hidden state (also called a memory state) capable of remembering some part of the past information in an input sequence at a given time step. Recurrent units share common weight matrices W_{hh} and W_{xh}, and eventually a bias vector b. The basic equation to describe a recurrent unit is

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b), \tag{2.4}$$

where σ is a non-linear activation function such as *tanh*, h_t the current hidden state (i.e., at time step t), h_{t-1} the previous hidden state, and x_t the input at time step t. In Figure 2.3, we illustrate a three-layer recurrent neural network processing a sequence of words. Rectangles annotated with $h_i^{(j)}$ are recurrent units, and each $h_i^{(j)}$ is a hidden state. The superscript $.^{(1)}$ indicates that a hidden state is from the first RNN layer, whereas $.^{(2)}$ and $.^{(3)}$ correspond to the second and third layers, respectively. The first layer receives an initial hidden state h_0 in addition to the input. From the left to the right, the RNN processes the input tokens "Love", "the", "life", "you", and aims to predict the next token, which in this case could be "live". As input tokens are received, at each layer, the RNN updates its hidden states following Equation 2.4 above.

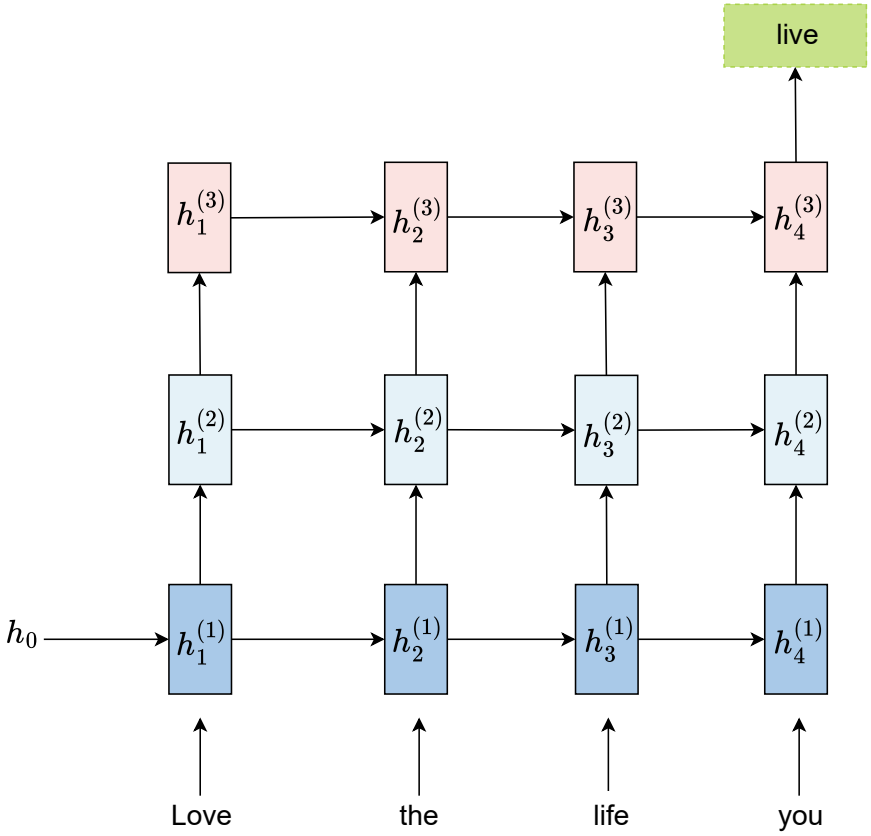


Figure 2.3: Two-layer RNN. The network processes multiple input tokens ("Love", "the", "life", "you") and sequentially updates its hidden states $h_j^{(j)}$. In this figure, the network aims to predict the next token, e.g., "live" that is most likely to come after the consumed tokens.

There are several RNN variants. Examples include BiRNN, LSTM, and GRU. BiRNN is short for bidirectional RNN, a recurrent neural network that processes an input sequence in two directions: from the start of the sequence to its end, and from the end of the sequence to its starting point. This is often achieved by two RNNs, the outputs of which are combined to produce the final output. LSTM, long short-term memory, is an RNN architecture that operates in a read-write-forget principle by introducing three gates: an *input gate* i_t, a *forget gate* f_t, and an *output gate* o_t. The forget gate decides which part of the past information may be forgotten depending on the target output. LSTM equations are as follows:

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}), \tag{2.5}$$
$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}), \tag{2.6}$$
$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}), \tag{2.7}$$
$$\tilde{c}_t = \tanh(W_{ic}x_t + b_{ic} + W_{hc}h_{t-1} + b_{hc}), \tag{2.8}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t, \tag{2.9}$$
$$h_t = o_t \circ \tanh(c_t). \tag{2.10}$$

In the equations 2.5–2.10, h_t, c_t, and x_t are the hidden state, the cell state, and the input at time step t, respectively. h_{t-1} is the hidden state at time step $t-1$. i_t, f_t, and o_t are the input, forget, and output gates, respectively. $\tilde{c}_t$ is candidate for the cell state at time step t. The W's and b's are trainable weight matrices and weight vectors. σ is the `sigmoid` activation function, and $\circ$ is the element-wise multiplication (also called Hadamard product). By introducing the three gates, LSTM outperforms simple RNNs on tasks involving long input sequences. GRU, gated recurrent unit, is a simplification of LSTM where the output gate is removed. GRU equations are as follows:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}), \tag{2.11}$$
$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}), \tag{2.12}$$
$$n_t = tanh(W_{in}x_t + b_{in} + r_t \circ (W_{hn}h_{t-1} + b_{hn})), \tag{2.13}$$
$$h_t = (1 - z_t) \circ n_t + z_t \circ h_{t-1}, \tag{2.14}$$

where r_t, z_t, and n_t are the reset gate, the update gate, and a candidate activation vector, respectively. In terms of predictive performance, GRUs are competitive with LSTMs, but the former are more parameter-efficient.

While RNNs and their variants perform relatively well on learning tasks involving sequential input data, e.g., statistical machine translation (SMT) [36], they often suffer exploding or vanishing gradients as input sequences get longer. Moreover, the

back-propagation through time (BPTT) algorithm [223] via which these networks are trained is computationally expensive. These limitations have led to the development of the transformer architecture as an alternative to RNNs. In the next lines, we present the transformer architecture and elucidate its building blocks, namely, the *attention* and *multi-head attention* functions.

Transformer

A transformer [79, 209, 210, 225] is a deep neural network architecture based on the multi-head attention mechanism which captures dependencies between the elements of an input sequence and their weighted contribution to the desired output, without recurrence or convolution. A basic transformer model is composed of an encoder and a decoder, both based on a multi-head attention mechanism. A multi-head attention function applies multiple attention functions in parallel, the outputs of which are combined for the final prediction. Additionally, this architecture employs a positional encoding technique to preserve the ordering information in an input sequence. Together, these components enhance the transformer's expressive power and computational efficiency.

Attention. An attention function $\mathrm{Att}(\cdot)$ takes three inputs: queries, keys, and values, and computes its output as a weighted sum of values [210]. Formally, let $Q \in \mathbb{R}^{n_q \times d_k}$, $K \in \mathbb{R}^{n_v \times d_k}$, and $V \in \mathbb{R}^{n_v \times d_v}$ be the query, key, and value matrices, respectively. The dot-product attention (also called multiplicative attention) function is defined as

$$\mathrm{Att}(Q, K, V) = Softmax(QK^T)V. \tag{2.15}$$

There are two main attention functions: dot-product attention, and additive attention [16]. Although the two have comparable model size and expressive power, the dot-product attention is computationally more efficient [25]. However, as the embedding dimension d_k increases, additive attention tends to outperform the dot-product attention [25]. Vaswani et al. [210] therefore proposed to rescale the dot-product in the attention computation by $\frac{1}{\sqrt{d_k}}$:

$$\mathrm{Att}(Q, K, V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2.16}$$

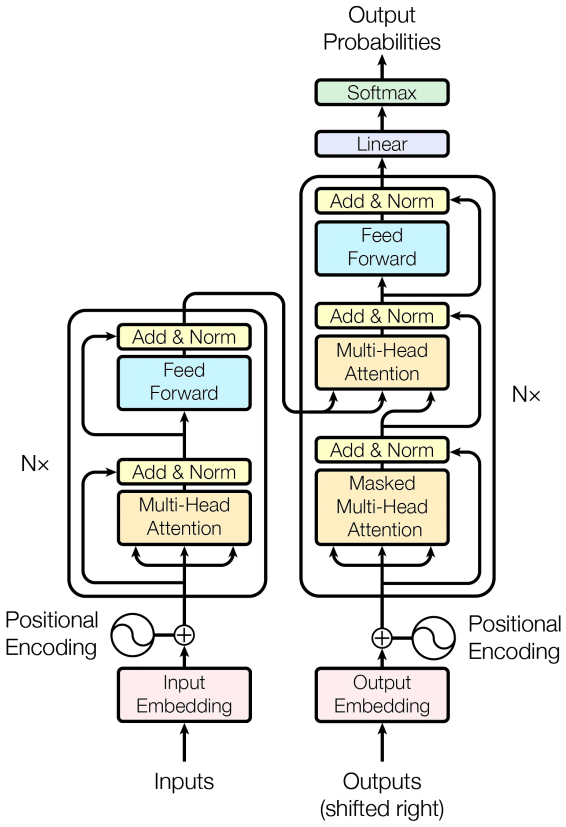


Figure 2.4: Transformer architecture. Source: Vaswani et al. [210].

The scaling factor prevents the dot-products from reaching high values which would lead to vanishing gradients during training.

Multi-head Attention. The multi-head attention function [118, 210] with h heads is defined as

$$\text{Multihead}(Q, K, V) = \text{Concat}([\text{head}_1, \text{head}_2, \ldots, \text{head}_\text{h}])W^O, \tag{2.17}$$

$$\text{head}_\text{i} = \text{Att}(\text{QW}_\text{i}^\text{Q}, \text{KW}_\text{i}^\text{K}, \text{VW}_\text{i}^\text{V}), \tag{2.18}$$

where $W_i^Q \in \mathbb{R}^{d_k \times d_{model}}$, $W_i^K \in \mathbb{R}^{d_k \times d_{model}}$, and $W_i^V \in \mathbb{R}^{d_v \times d_{model}}$ are head-specific projection layers for queries, keys, and values, respectively. $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ is the final projection layer, d_k is the size of hidden layers, and Concat a horizontal concatenation function. For computational efficiency, the dimension of each attention head is often chosen such that $hd_v = hd_k = d_{model}$, thus $d_v = d_k = d_{model}/h$.

Initially developed as an alternative to RNNs for machine translation [36, 133], transformers have since been adopted in various fields of AI, including large-scale natural language processing [209, 213, 225], computer vision (with vision

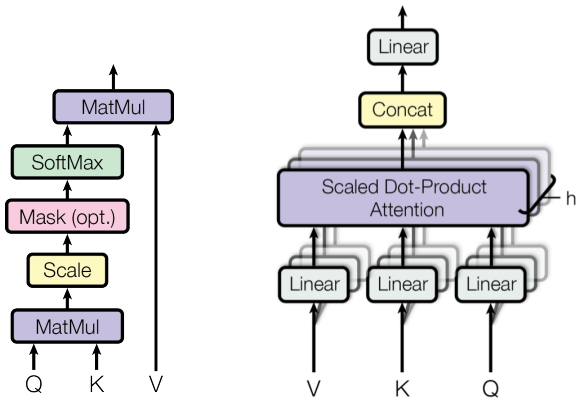


Figure 2.5: (Left): Scaled dot-product attention function. (Right): Multi-head attention function. Source: Vaswani et al. [210], and Weng [222].

transformers) [80, 102], reinforcement learning [32, 128], multi-modal applications [168, 231], and robotics [26, 217]. In each of these subfields, transformers achieve state-of-the-art performance. The success of transformers, especially in the field of natural language processing (NLP), has led to the creation of pre-trained systems, such as generative pre-trained transformers (GPTs) among which feature the well-known OpenAI's ChatGPT and GPT-4[1], Google's Gemini [200], or Meta's Llama [205], only to mention a few.

2.1.3 Permutation-Invariant Neural Network Architectures for Set Inputs

In this thesis, we mainly deal with set-structured input data (i.e., sets of positive and negative examples for a given learning problem) as in 3D shape recognition [169], multiple instance learning [54], and few-shot learning [63, 194]. 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 sizes.

DeepSets

In recent years, several approaches have been developed to meet the aforementioned requirements. The most prominent of these approaches include DeepSets [242] and Set-Transformer [118]. The DeepSets model encodes each element in an input set independently and uses a pooling layer, e.g., averaging, to produce the final

representation of the set. More specifically, it maps an input set $\chi = \{x_1, x_2, \ldots, x_n\}$ into a hidden representation following the equation

$$\text{DeepSets}(\chi) = f_\theta \left(\sum_{x \in \chi} \phi_{\tilde{\theta}}(x) \right), \tag{2.19}$$

where f_θ and $\phi_{\tilde{\theta}}$ are deep neural networks with trainable parameters θ and $\tilde{\theta}$, respectively. Obviously, the final hidden representation of the input set does not depend on the order of its elements. This makes DeepSets a suitable architecture for many set-input learning tasks. However, recent results suggest that attention-based network architectures such as Set-Transformer outperform DeepSets on many learning tasks.

Set-Transformer

Set-Transformer [118] is similar to the transformer architecture presented in Section 2.1.2, but it does not use positional encoding since it takes sets as input and the order of their elements does not matter. Set-Transformer employs a self-attention mechanism to encode an input set, which allows pair-wise and even higher-order interactions between its elements. Set-Transformer outperforms DeepSets on many set-input learning tasks with a comparable model size [118]. For this reason, our works in Chapters 5, 6, and 7 use the Set-Transformer architecture. Its building blocks are described below.

Multi-head Attention Block (MAB). Lee et al. [118] introduced a modified version of multi-head attention blocks tailored toward set-structured inputs. We employ the same attention blocks but without layer normalisation, as the latter did not improve the results in our learning tasks. For any $X,\ Y \in \mathbb{R}^{n \times d}$, we define MAB as:

$$H = X + \text{Multihead}(X, Y, Y), \tag{2.20}$$

$$\text{MAB}(X, Y) := H + \text{FF}(H), \tag{2.21}$$

where FF is a feedforward neural network layer and n is the number of elements in the input sets X and Y.

Set Attention Block (SAB). This module operates on a single set of embedding vectors. Following [118], we define SAB as:

$$\mathrm{SAB}(X) := \mathrm{MAB}(X, X). \tag{2.22}$$

SAB aims to capture interactions between the elements of a input set via an attention mechanism. However, the computational complexity of SAB grows quadratically with the size of the input set. This has lead to the introduction of the induced set attention block, as described below.

Induced Set Attention Block (ISAB). To reduce the computational complexity in attention mechanisms for especially large sets, Lee et al. [118] proposed using the so-called inducing points. The main idea is to introduce a (small) fixed-size trainable matrix $I \in \mathbb{R}^{m \times d}$ as a bridge in SAB. More specifically, ISAB is defined for any $X \in \mathbb{R}^{n \times d}$ as:

$$H = \mathrm{MAB}(I, X) \in \mathbb{R}^{m \times d}, \tag{2.23}$$

$$\mathrm{ISAB}(X) := \mathrm{MAB}(X, H) \in \mathbb{R}^{n \times d}. \tag{2.24}$$

Note that the computation cost in ISAB is $4 \times (m \times d \times n)$. This is less than that of SAB which is $2 \times (n \times d \times n)$, for $n >> m$.

Pooling by Multi-head Attention (PMA). As opposed to the pooling mechanism in [242], where the learned features from the encoder are summed (decoding), the decoder in Set-Transformer [118] applies a multi-head attention function between a trainable feature matrix and the features from the encoder. Formally, a Pooling by Multi-head Attention layer (PMA) with k seed vectors $S \in \mathbb{R}^{k \times d}$ is defined for any $X \in \mathbb{R}^{n \times d}$ by:

$$\mathrm{PMA}(X) := \mathrm{MAB}(\mathrm{S}, \mathrm{X}). \tag{2.25}$$

Unlike the set attention block SAB and the induced set attention block ISAB that output a matrix with n rows (i.e., the number of elements in the input set), the PMA module outputs a matrix with k rows. For class expression learning (see Chapters 5, 6, and 7), we use an additional linear layer on top of PMA to match the desired output shape.

Connection to Our Contributions. Our proposed approaches for class expression learning are based on deep neural networks. This choice is in part motivated by the fact that deep neural networks are universal function approximators [40, 90] and can therefore be trained to perform the task of class expression learning. Another reason is that deep learning models support fast computation hardware such as GPUs and can run on several computing resources in parallel (multiprocessing and multi-GPU computing), which is not the case for previous search-based approaches for class expression learning.

Our proposed approach CLIP (Chapter 4) uses LSTM, GRU, CNN, and MLP architectures. NCES (Chapter 5) supports three network architectures: LSTM, GRU, and Set-Transformer. NCES2 (Chapter 6) and ROCES (Chapter 7) use the Set-Transformer architecture to learn mappings between sets of input examples and class expressions in description logics. Training machine learning models can be seen as an empirical risk minimization problem, as described below.

2.1.4 Expected and Empirical Risk Minimization

Let $\mathcal{X}, \mathcal{Y}$ be an input (feature) space, and an output space of a learning task, respectively. Also let $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_+$ be a loss function that measures the discrepancy between two elements in $\mathcal{Y}$. Then, the expected risk of a hypothesis f on $\mathcal{X} \times \mathcal{Y}$ with respect to $\mathcal{L}$ is defined as [44, 126]

$$\mathbb{E}_{(x,y)\sim\mathcal{X}\times\mathcal{Y}}\mathcal{L}(f(x), y) = \int_{\mathcal{X}\times\mathcal{Y}} \mathcal{L}(f(x), y)\mathrm{d}\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}(x, y), \tag{2.26}$$

where $\mathbb{P}_{\mathcal{X}\times\mathcal{Y}} : \mathcal{X} \times \mathcal{Y} \rightarrow [0, 1]$ is the true distribution of the data $\mathcal{X} \times \mathcal{Y}$. Assuming that $\mathcal{F}$ is the set of all possible hypotheses, the *expected risk minimization* problem is defined as the task of finding a hypothesis $f^* \in \mathcal{F}$ with minimum risk on $\mathcal{X} \times \mathcal{Y}$:

$$f^* = \arg\min_{f\in\mathcal{F}} \int_{\mathcal{X}\times\mathcal{Y}} \mathcal{L}(f(x), y)\mathrm{d}\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}(x, y). \tag{2.27}$$

In the context of machine learning with real world datasets, the distribution $\mathbb{P}_{\mathcal{X}\times\mathcal{Y}}$ is often unknown, and only some part $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \subset \mathcal{X} \times \mathcal{Y}$ of the data is observed, with samples (x_i, y_i) assumed to be independent and identically distributed (i.i.d). Moreover, the set of hypotheses $\mathcal{F}$ is restricted to a

class of functions, e.g., parameterized functions $\{f_\theta\}_{\theta \in \Theta}$. In this case, the goal is to find parameters $\theta^* \in \Theta$ with minimum risk on $\mathcal{D}$:

$$\theta^* \approx \arg\min_{\theta \in \Theta} \int_{\mathcal{D}} \mathcal{L}(f_\theta(x), y) \mathrm{d}\mathbb{P}_{\mathcal{X} \times \mathcal{Y}}(x, y) \tag{2.28}$$

$$= \arg\min_{\theta \in \Theta} \sum_{(x,y) \in \mathcal{D}} \frac{1}{|\mathcal{D}|} \mathcal{L}(f_\theta(x), y) \tag{2.29}$$

$$= \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f_\theta(x_i), y_i). \tag{2.30}$$

This optimization problem is known as *empirical risk minimization* [126]. While in Equations 2.28–2.30 the goal hypothesis θ^* is assumed to achieve the lowest risk, it is rare to find such a hypothesis in practice. This is because computing resources or time for a given learning task are often limited, and hence the best hypothesis found within the allowed budget is retained.

In the case of linear regression, *maximum likelihood estimation* (MLE) and *maximum a posteriori probability estimation* (MAP) can be used to find an optimal hypothesis θ^* [34, 44]. MLE aims to find a hypothesis θ that maximizes the probability of the data $\mathcal{D}$. For instance, assuming a Gaussian model $\mathbb{P}(\mathcal{Y} = y | \mathcal{X} = x; \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\theta^T x - y)^2}{2\sigma^2}\right)$, the MLE is equivalent to the expected risk minimization problem with mean squared error loss:

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (\theta^T x_i - y_i)^2. \tag{2.31}$$

In MAP, there is a prior belief about the hypothesis class $\mathcal{F}$ (i.e., hypotheses are assumed to come from a known distribution), and the goal is to maximize the likelihood of observing a particular hypothesis given the data $\mathcal{D}$. Since MLE and MAP are not relevant for this thesis, we refer the reader to [44] (Section 2), and [58, 69] for further details.

Connection to Our Contributions. In the general case, a machine learning task is concerned with the search for a hypothesis (in a given hypothesis class) that explains (fits) some given data, i.e., the hypothesis achieves a low risk as defined in Equation 2.28. In this sense, any machine learning task can be modeled as an empirical risk minimization problem since it is usually the case that the data is incomplete, and the expected risk as defined in Equation 2.27 cannot be computed. Class expression learning with our proposed deep learning-based approaches is not an exception to this general setting, i.e., we make use of empirical risk minimization.

2.1.5 Loss Functions in Machine Learning

In machine learning, a loss function (also called error function, criterion, objective, or cost function) is a component used to quantify the difference between a predicted value and a target value [127, 158]. Formally, let the output space of a learning task be $\mathcal{Y}$. A loss function is a mapping $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ which returns a positive real number or zero for any input pair $(\hat{y}, y) \in \mathcal{Y} \times \mathcal{Y}$. The value of a loss function is often directly associated to the performance of the considered predictor. High loss values indicate poor performance, and low values indicate better performance in terms of predictive accuracy. However, when evaluating a machine learning model, it is also important to consider the contextual implications of the loss function. This is because different loss functions capture various aspects of predictive accuracy; some may prioritize large errors, while others might also consider smaller discrepancies. Several loss functions exist for different learning tasks. Below, we present a non-exhaustive list of frequently used loss functions.

Mean Squared Error Loss (MSE)

The mean squared error loss [158, 216] is a function suited for regression tasks, i.e., targets are real numbers.[3] Given a batch of n predicted real-valued outputs $\hat{y} \in \mathbb{R}^n$ and the corresponding targets $y \in \mathbb{R}^n$, the MSE loss computes the average squared difference between $\hat{y}$ and y:

$$\mathrm{MSE}(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2. \tag{2.32}$$

By squaring the difference between the target and the prediction, the MSE loss can effectively capture the magnitudes of prediction errors in a robust manner, ensuring that larger discrepancies (e.g., outliers) are given more weight in the loss calculation. This makes MSE an appropriate choice for regression tasks with medium to large target values (i.e., when target values are outside the unit interval $[0, 1]$) since any significant differences between predictions and targets are then strongly reflected in the loss values. An example of such regression tasks include the prediction of real estate price, with values often ranging between hundreds of thousands to millions of dollars.

[3] Targets can also be real-valued vectors, matrices, or higher dimensional arrays. In this case, the loss is applied along all dimensions, and the results are averaged.

Mean Absolute Error Loss (MAE)

Similarly to MSE, the mean absolute error (MAE) loss [158, 216] is used for regression tasks. For any batch of real-valued predictions $\hat{y} \in \mathbb{R}^n$ and its corresponding targets $y \in \mathbb{R}^n$, we have

$$\mathrm{MAE}(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|. \tag{2.33}$$

One difference between MAE and MSE is that the former treats all prediction errors linearly, as opposed to the latter which squares them. This makes MAE more robust in scenarios where outliers can negatively impact model performance. Consequently, while MAE may not penalize large errors as severely as MSE does, it reliably delivers an accurate representation of average model performance, which can be of particular importance in many real-world applications.

Binary Cross-Entropy Loss (BCE)

The binary cross-entropy loss [139, 216] is a function used for binary classification tasks. The binary cross-entropy loss between a batch of predictions $\hat{y} \in [0, 1]^n$ and targets $y \in \{0, 1\}^n$ is defined as

$$\mathrm{BCE}(\hat{y}, y) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i). \tag{2.34}$$

In Equation 2.34, $\hat{y}_i$ denotes the probability that the $i - th$ sample belongs to class 1. $1 - \hat{y}_i$ denotes the probability that the $i - th$ sample belongs to class 0. The BCE loss is a widely used loss function in machine learning, e.g., for skin cancer detection [91, 172, 220], due to its desirable properties. When predictions are confidently close to either 0 or 1, the loss value is low, reflecting better model performance. In contrast, when predictions are uncertain (e.g., close to 0.5), the loss value is high, indicating that the model needs improvement. However, BCE cannot handle multi-class classification tasks, e.g., classifying mammals into their species, since there are about 7000 mammal species in total. To overcome this limitation, categorical cross-entropy loss (or simply cross-entropy loss) has been proposed as a generalization of BCE loss.

Categorical Cross-Entropy Loss (CE)

The categorical cross-entropy loss or cross-entropy loss (CE) [139, 216] has been proposed as an extension of the BCE loss for multi-class classification. For a classification task with $K \geq 2$ classes, a batch of predicted probabilities is two-dimensional ($\hat{y} \in [0,1]^{n \times K}$) and the corresponding batch of targets is one-dimensional and non-binary for $K > 2$ ($y \in [0, K-1]^n$)[4] , where n denotes the batch size. In this case the cross-entropy loss is defined as

$$\mathrm{CE}(\hat{y}, y) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=0}^{K-1} \mathbb{1}(y_{ik} = k) \log \hat{y}_{ik}, \tag{2.35}$$

where $\mathbb{1}$ denotes the indicator function which returns 1 if the input condition is true and 0 otherwise. The indicator function ensures that the logarithm is only applied to the predicted probability of the true class for each datapoint. Consequently, the cross-entropy loss aims to maximize the probability of the correct class labels by minimizing the negative log-likelihood of the predicted probabilities. This formulation captures the essence of multi-class classification, where each instance is associated with a single class label.

The cross-entropy loss is widely used in image classification (e.g., on ImageNet [52]), but also in training large language models (LLMs). In the latter, the cross-entropy loss can be used to build a conditional probability distribution over a vocabulary of tokens, e.g., predict the next token in a sequence given the preceding context.

There are many other loss functions that we do not cover in this chapter; we refer the reader to [158, 216] for further readings. In the most general case, training a machine learning model can be seen as the problem of minimizing a loss function on the available data, see Section 2.1.4. This is achieved by optimization algorithms, as described in the next section.

Connection to Our Contributions. Our neural network-based approaches for class expression learning are trained using one or multiple loss functions. Concept length predictors in CLIP are trained using the cross-entropy loss (CE). The deep neural synthesizer in NCES is trained using the cross-entropy loss as well. NCES2 and ROCES incorporate two main components: an embedding model which provides embeddings for input examples, and a deep neural synthesizer responsible for producing description logic class expressions. Both components are trained jointly

[4] Here, we assume that the first class is labelled as 0, thus possible classes are $0, 1, \ldots, K-1$.

with different loss functions: the binary cross-entropy loss (BCE) for the embedding model, and the cross-entropy loss (CE) for the deep neural synthesizer.

2.1.6 Optimization Algorithms for Machine Learning

In the following, $\mathcal{D}$ is a finite dataset, and $\mathcal{B} \subseteq \mathcal{D}$ a batch of datapoints in $\mathcal{D}$. We denote the average loss achieved by a model f_θ on a batch $\mathcal{B}$ of datapoints by $\mathcal{L}_\mathcal{B}(\theta)$. As pointed out in Section 2.1.1, training machine learning models is usually achieved by using optimization algorithms. Most of these algorithms are gradient-based. Below, we present some of the most prominent optimization algorithms for machine learning, including stochastic gradient descent (SGD) [178], Momentum [166], adaptive gradients (AdaGrad) [57], root mean squared propagation (RMSProp) [202], and adaptive moment estimation (Adam) [104].

SGD

Stochastic gradient decent (SGD) [178] (in fact, minibatch stochastic gradient descent)[5] was one of the first optimization algorithms to be successful and widely adopted in machine learning. It provides a simple and efficient parameter update rule by using gradients computed on batches of uniformly sampled datapoints, and a fixed learning rate (also known as step size). Formally, at time step t, parameter updates are performed as follows:

$$g_t = \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_{t-1}), \quad \textit{\# Gradient at of the loss w.r.t } \theta \tag{2.36}$$

$$\theta_t = \theta_{t-1} - \alpha g_t, \quad \textit{\# Parameter update} \tag{2.37}$$

where $\mathcal{B}$ is a batch of training datapoints, and α is the learning rate. Note that the learning rate α is constant and the same for all parameters. However, in some learning scenarios, different parameters might require different learning rates for maximum performance. The optimization algorithms that we present below introduce the notions of adaptive learning rate, momentum, or adaptive moments to mitigate this issue. We denote the Hadamard product by $\circ$ and use the simplification $g_t \circ g_t = g_t^2$.

[5] In the literature, the term SGD is often used to refer to the optimization algorithm that performs parameter updates using a single datapoint drawn uniformly from the training data. However, this approach is rarely employed in contemporary machine learning; instead, SGD usually denotes minibatch stochastic gradient descent.

Momentum

Momentum [166] is a generalization of SGD that linearly combines a momentum vector with the gradient before parameter update. Its parameter update rule is as follows:

$$g_t = \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_{t-1}), \tag{2.38}$$

$$v_t = \mu v_{t-1} + (1-\mu) g_t, \quad \textit{\# Momentum update} \tag{2.39}$$

$$\theta_t = \theta_{t-1} - \alpha v_t, \tag{2.40}$$

where μ is the momentum parameter, and α the learning rate. Note that in some literature, momentum update is performed as $v_t = \mu v_{t-1} + g_t$. By using the momentum vector, this optimization algorithm significantly improves the convergence speed.

RMSProp

RMSProp [202] with momentum updates parameters after rescaling the gradients:

$$g_t = \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_{t-1}), \tag{2.41}$$

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2, \tag{2.42}$$

$$\theta_t = \theta_{t-1} - \alpha g_t / (\epsilon + \sqrt{v_t}), \tag{2.43}$$

where β is the rescaling parameter, ϵ a positive infinitesimal number (often used to prevent division by zero)[6] and α the learning rate. The rescaling step allows RMSProp to assign different learning rates to different parameters, and achieves state-of-the-art performance on several learning tasks, including image classification [198], and reinforcement learning [83].

[6] It has been reported that ϵ can also be used as a hyperparameter to further improve training performance, see for example [198] where $\epsilon = 1.0$.

AdaGrad

AdaGrad [57] is a stochastic optimization algorithm that rescales a given subgradient using all of the previous corresponding subgradients. Its update rule reads

$$g_t = \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_{t-1}), \tag{2.44}$$

$$\theta_t = \theta_{t-1} - \alpha g_t / \left(\epsilon + \sqrt{\sum_{i=1}^{t} g_i^2} \right), \tag{2.45}$$

where α is the learning rate, and ϵ an infinitesimal positive number. AdaGrad is known to work well with sparse gradients, and effectively handles vanishing and exploding gradients thanks to the adaptive rescaling factor.

Adam

Adam [104] is a stochastic optimization algorithm that uses first and second moment estimates of gradients to compute individual adaptive learning rates for different parameters. It combines the strengths of AdaGrad [57] (known to work well with sparse gradients) and RMSProp [202] (which excels at online and non-stationary settings). The update rule of Adam is as follows:

$$v_0 = m_0 = 0, \tag{2.46}$$

$$g_t = \nabla_\theta \mathcal{L}_\mathcal{B}(\theta_{t-1}), \tag{2.47}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{2.48}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2, \tag{2.49}$$

$$\hat{m}_t = m_t / (1 - \beta_1^t), \tag{2.50}$$

$$\hat{v}_t = v_t / (1 - \beta_2^t), \tag{2.51}$$

$$\theta_t = \theta_{t-1} - \alpha \hat{m}_t / (\epsilon + \sqrt{\hat{v}_t}), \tag{2.52}$$

where β_1 and β_2 are exponential decay rates, α the learning rate (step size), and ϵ a positive infinitesimal number preventing division by zero. ADOPT [199] is a modification of Adam which improves convergence by changing the order of the momentum update and using a different normalization term. Since its introduction, Adam has consistently ranked in the top best-performing and most popular optimization algorithms in deep learning.

Connection to Our Contributions. Our deep neural network-based approaches CLIP (Chapter 4), NCES (Chapter 5), NCES2 (Chapter 6), and ROCES (Chapter 7) are trained using the Adam optimizer. In our preliminary experiments, we tried different optimizers, and found out that Adam was the best-performing algorithm.

2.2 Description Logics

Description logics (DLs) [153, 154] are a family of languages for formal representation of knowledge in a variety of application domains such as artificial intelligence, bio-informatics, the semantic web, and automated reasoning. They are generally more expressive than propositional logic [28] but less expressive than first-order logic [18]. The $\mathcal{A}$ttributive $\mathcal{L}$anguage with $\mathcal{C}$omplements ($\mathcal{ALC}$) is the smallest (in terms of expressiveness) description logic that allows to expression the conjunction, disjunction, negation, full existential and universal role restrictions of concepts.

Definition 2.1 (Concepts in $\mathcal{ALC}$ [121, 153, 154])**.** *$\top$, $\bot$, and all atomic concepts are concepts in $\mathcal{ALC}$. If C, D are concepts in $\mathcal{ALC}$, and r a role, then the following are also concepts in $\mathcal{ALC}$:*

1. $\neg C$ *(negation)*
2. $C \sqcap D$ *(conjunction)*
3. $C \sqcup D$ *(disjunction)*
4. $\exists\, r.C$ *(existential quantification)*
5. $\forall\, r.C$ *(universal quantification)*

Definition 2.2 (Length of a Concept in $\mathcal{ALC}$ [107, 120])**.** *$\top$, $\bot$, and all atomic concepts are of length 1. In complex concepts, every constructor (e.g., $\neg, \exists, \forall$, etc, see other constructors in Table 2.1) and every role are counted. Hence, given concepts C, D, and a role r, we have the following:*

1. $length(\neg C) = 1 + length(C)$
2. $length(C \sqcap D) = length(C \sqcup D) = 1 + length(C) + length(D)$
3. $length(\exists\, r.C) = length(\forall\, r.C) = 2 + length(C)$

In Definition 2.1 above, we can observe that $\mathcal{ALC}$ concepts are constructed inductively, starting from the most basic ones. For instance, given the concepts `Person`, `Father`, and the role `hasChild`, the following are concepts in $\mathcal{ALC}$: `Person` $\sqcap$

$\texttt{Father}$, $\neg\texttt{Father}$, $\exists\ \texttt{hasChild.Person}$, $\texttt{Father} \sqcup \exists\ \texttt{hasChild.Person}$. Another important aspect of Definition 2.1 is that it does not restrict how long[7] $\mathcal{ALC}$ concepts can be, which suggests that infinitely many concepts can be constructed in $\mathcal{ALC}$. Consequently, $\mathcal{ALC}$ concepts can get infinitely long. Yet, this does not push the boundaries of the description logic in any way. As an example, there is no concept in $\mathcal{ALC}$ to describe *the set of people with at most one child*. Such limitations show the need for more expressive description logics—extensions of $\mathcal{ALC}$. Such extensions are obtained by adding one or more of the following:

1. $\boxed{\mathcal{S}}$ Equivalent to $\mathcal{ALC}$ with support for role transitivity
2. $\boxed{\mathcal{H}}$ Role hierarchies, e.g., $\texttt{marriedTo} \sqsubseteq \texttt{partnerOf}$
3. $\boxed{\mathcal{O}}$ Nominals, i.e., sets of objects are concepts, e.g., $\{\texttt{Peter}, \texttt{Anna}\}$
4. $\boxed{\mathcal{I}}$ Role inversions, e.g., $\texttt{hasChild} \equiv \texttt{hasParent}^-$
5. $\boxed{\mathcal{N}}$ Number restrictions, e.g., $\geq 3\ \texttt{hasLeg}$
6. $\boxed{\mathcal{Q}}$ Qualified number restrictions, e.g., $\leq\ \texttt{1 hasChild.Child}$
7. $\boxed{\mathcal{F}}$ Allows to express that a role r is functional, i.e., $\leq 1\ r \equiv \top$, e.g., $\leq 1\ \texttt{hasAge} \equiv \top$, which states that a thing (e.g., a person) can have at most one age
8. $\boxed{\mathcal{R}}$ Complex role inclusions, e.g., $\texttt{colleagueOf} \circ \texttt{worksFor} \sqsubseteq \texttt{worksFor}$, which states that if $\texttt{Person A}$ is a colleague of $\texttt{Person B}$, and $\texttt{Person B}$ works for $\{\texttt{Company C}\}$, then $\texttt{Person A}$ also works for $\{\texttt{Company C}\}$
9. $\boxed{\mathcal{D}}$ Use of data types, concrete roles, or data values. These allow to define concepts based on data values, e.g., $\geq 1.80m\ \texttt{hasHeight}$, which can be interpreted as the set of things with a height of at least 1.80 meters.

Obviously, with the addition of the constructor $\boxed{\mathcal{Q}}$ (i.e., in $\mathcal{ALCQ}$), our previous example about describing the set of people with at most one child can now be expressed: $\leq\ \texttt{1 hasChild.Child}$ or even $\leq\ \texttt{1 hasChild.Person}$ since a child is also (in principle) a person. Translating complex concepts into the sets of things they describe is often achieved via an interpretation as defined below.

Definition 2.3 (Interpretation [121, 153, 154])**.** *An interpretation is a tuple* $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$*, where* $\Delta^{\mathcal{I}}$ *is a non-empty set, and* $\cdot^{\mathcal{I}}$ *an interpretation function which assigns a subset* $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ *to every concept* C*, and a subset* $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ *to every role* r*.*

Note that in Definition 2.3, $\Delta^{\mathcal{I}}$ often excludes data values, and concrete roles are not taken into account. To account for these, a set of data values $\mathcal{D}$ can be added

[7] See Definition 2.2 for a definition of concept lengths. While the definition focuses on $\mathcal{ALC}$ concepts, it can be easily extended to more expressive description logics such as $\mathcal{SROIQ}^{(\mathcal{D})}$. For this, one would need to account for the lengths of complex roles, e.g., $length(\exists r \circ s.C) = 5, length(r^-) = 2$, etc.

and a concrete role r_c would then be interpreted as a subset $r_c^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \mathcal{D}$. The syntax and semantics of $\mathcal{ALCHIQ}^{(\mathcal{D})}$ are given in Table 2.1.

Table 2.1: Syntax and semantics of $\mathcal{ALCHIQ}^{(\mathcal{D})}$. $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is an interpretation where $\Delta^{\mathcal{I}}$ is its domain and $\cdot^{\mathcal{I}}$ is the interpretation function. $\mathcal{D}$ denotes the set of data values, e.g., strings, numbers, Boolean values, etc.

Syntax	Construct	Semantics
	$\mathcal{ALC}$	
r	abstract role	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
$\top$	top concept	$\Delta^{\mathcal{I}}$
$\bot$	bottom concept	$\emptyset$
C	atomic concept	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
$\neg C$	negation	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$C \sqcup D$	union	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$C \sqcap D$	intersection	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$\exists\, r.C$	existential restriction	$\{a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \mid \exists\, b^{\mathcal{I}} \in C^{\mathcal{I}}, (a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}\}$
$\forall\, r.C$	universal restriction	$\{a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \mid \forall\, b^{\mathcal{I}}, (a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}} \Rightarrow b^{\mathcal{I}} \in C^{\mathcal{I}}\}$
	$\mathcal{H}$	
$r_1 \sqsubseteq r_2$	abstract role hierarchy	$r_1^{\mathcal{I}} \subseteq r_2^{\mathcal{I}}$
	$\mathcal{I}$	
r^-	inverse abstract role	$\{(b^{\mathcal{I}}, a^{\mathcal{I}}) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}\}$
	$\mathcal{Q}$	
$\leq n\, r.C$	max. card. restriction	$\{a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \mid \lvert\{b^{\mathcal{I}} \in C^{\mathcal{I}} : (a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}\}\rvert \leq n\}$
$\geq n\, r.C$	min. card. restriction	$\{a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \mid \lvert\{b^{\mathcal{I}} \in C^{\mathcal{I}} : (a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}\}\rvert \geq n\}$
$= n\, r.C$	exact card. restriction	$\{a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \mid \lvert\{b^{\mathcal{I}} \in C^{\mathcal{I}} : (a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}\}\rvert = n\}$
	$(\mathcal{D})$	
r_c	concrete role	$(r_c)^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \mathcal{D}$
$r_c \leq v$	max. restriction	$\{a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \mid \exists\, w \in \mathcal{D}, (a^{\mathcal{I}}, w) \in (r_c)^{\mathcal{I}} \wedge w \leq v\}$
$r_c \geq v$	min. restriction	$\{a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \mid \exists\, w \in \mathcal{D}, (a^{\mathcal{I}}, w) \in (r_c)^{\mathcal{I}} \wedge w \geq v\}$
$r_c = v$	exact value restriction	$\{a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \mid (a^{\mathcal{I}}, v) \in r_c^{\mathcal{I}}\}$

Description logics can be easily translated into a machine readable format, the web ontology language (OWL). In OWL, atomic concepts are called classes, roles are called properties, and complex concepts are called class expressions [120]. OWL is built on top of other technologies such as RDF/XML[8], RDFS[9], etc. We refer the reader to the official page[10] of OWL for further details. Note that in the scope of this thesis, we employ the terms "concept" and "class expression" interchangeably.

Due to their readability and expressive power in knowledge representation, description logics are being massively used to create and maintain a special type of

[8] `https://www.w3.org/TR/rdf12-xml/`
[9] `https://www.w3.org/2001/sw/wiki/RDFS`
[10] `https://www.w3.org/TR/2012/REC-owl2-overview-20121211/`

knowledge bases—description logic (DL) knowledge bases, often stored as OWL documents. In such knowledge bases, which we formally define below, one part of the knowledge is expressed in terms of hierarchies between concepts or roles, while the other part describes the relationships between individuals or between individuals and concepts.

2.3 Knowledge Base in Description Logics

In this section, we introduce the notion of knowledge base in the context of description logics. We start with a formal definition before delving into different characteristics of DL knowledge bases and associated notions.

Definition 2.4 (DL Knowledge Base)**.** *A DL (description logic) knowledge base is an ordered pair $(\mathcal{T}, \mathcal{A})$, where*

1. *$\mathcal{T}$ (also called the terminological box or `TBox`) contains hierarchy information (axioms of the form $C \sqsubseteq D$) about concepts or roles,*
2. *$\mathcal{A}$ (known as the assertional box or `ABox`) contains membership information (statements of the form $C(a), (a, r, b)$) about individuals, and how they relate to each other.*

Elements of $\mathcal{T}$ are called axioms and those of $\mathcal{A}$ assertions. The set of individuals in a knowledge base is denoted by N_I, and the set of roles is denoted by N_R.

Note that in some literature, axioms about roles (e.g., role subsumption, role chaining, role transitivity, etc) are often separated into a new set called `RBox`, and a DL knowledge base is then defined a triplet (`TBox`, `RBox`, `ABox`). The reason for this is often to emphasize on the expressivity of the underlying description logic. In this thesis, however, we assume axioms from the `RBox` to be merged into the `TBox`.

Example 2.5 (DL Knowledge Base)**.** *Consider the sets $\mathcal{A}$ and $\mathcal{T}$ below:*

$$\begin{aligned}\mathcal{T} =& \{\mathit{Penguin} \sqsubseteq \neg \mathit{CanFly}, \mathit{hasParent} \equiv \mathit{hasChild}^{-},\\ & \exists\, \mathit{hasChild}.\mathit{Person} \sqsubseteq \mathit{Person}\},\\ \mathcal{A} =& \{\mathit{Person}(\mathit{Bob}), \mathit{Penguin}(\mathit{Peng\_0}), \mathit{hasChild}(\mathit{Peter}, \mathit{Bob})\}.\end{aligned}$$

The pair $\mathcal{K}$ defined by $\mathcal{K} := (\mathcal{T}, \mathcal{A})$ is a DL knowledge base. In this example, one can deduce that `Person(Peter)`, that is, `Peter` is an instance of the concept `Person`. This is achieved by using the two assertions in $\mathcal{A}$ and the last axiom in $\mathcal{T}$, which states that every individual that has a child that is a person is also a person. The process of combining assertions in the `ABox` and/or axioms in the `TBox` to deduce new facts is known as reasoning in DL knowledge bases. This is often achieved by using so-called DL (or OWL) reasoners. Example reasoners include Pellet [193], FaCT++ [208], HermiT [71], and RacerPro [77]. Reasoners support a number of services, including *consistency check*, *satisfiability*, *subsumption*, *instance retrieval*, etc. We introduce some of these notions below and provide examples where possible.

In the rest of this chapter, we simply write $\mathcal{I}$ to denote the interpretation $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, unless stated otherwise. A DL knowledge base is also simply called knowledge base.

Definition 2.6 (Axiom Satisfaction, Assertion Satisfaction [120, 153, 154])**.** *Let $\mathcal{T}$ be a `TBox`, $\mathcal{A}$ an `ABox`, C, D concepts, r, s roles, a, b individuals, and $\mathcal{I}$ an interpretation. An axiom of the form $C \sqsubseteq D$ (respectively, $r \sqsubseteq s$) is satisfied with respect to $\mathcal{I}$ if and only if $C^{\mathcal{I}} \sqsubseteq D^{\mathcal{I}}$ (respectively, $r^{\mathcal{I}} \sqsubseteq s^{\mathcal{I}}$).*

Similarly, an assertion of the form $C(a)$ (respectively, $r(a, b)$) is satisfied with respect to $\mathcal{I}$ if and only if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ (respectively, $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$).

Definition 2.7 (Model of a `TBox` [120, 153, 154])**.** *Let $\mathcal{T}$ be a `TBox`. An interpretation $\mathcal{I}$ is a model of $\mathcal{T}$ if and only if all axioms in $\mathcal{T}$ are satisfied with respect to $\mathcal{I}$.*

Definition 2.8 (Model of an `ABox` [120, 153, 154])**.** *Let $\mathcal{A}$ be an `ABox`, and $N_{\mathcal{A}}$ the set of all individuals appearing in $\mathcal{A}$. An interpretation $\mathcal{I}$ is a model of $\mathcal{A}$ if and only if*

1. *$\mathcal{I}$ maps every individual $a \in N_{\mathcal{A}}$ to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$,*
2. *all assertions in $\mathcal{A}$ are satisfied with respect to $\mathcal{I}$.*

Definition 2.9 (Model of a DL Knowledge Base [120, 153, 154])**.** *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, and N_I the set of all individuals in $\mathcal{K}$. An interpretation $\mathcal{I}$ is a model of $\mathcal{K}$ if and only if $\mathcal{I}$ maps every individual $a \in N_I$ to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, and $\mathcal{I}$ is a model of both $\mathcal{T}$ and $\mathcal{A}$.*

Note that $N_{\mathcal{A}}$ is always a subset of N_I, and the requirement to map every individual in N_I to an element of the domain $\Delta^{\mathcal{I}}$ does not conflict with Definition 2.8.

Example 2.10 (Model of a DL Knowledge Base)**.** *Consider the knowledge base in Example 2.5. Let* $\Delta^{\mathcal{I}} = \{\texttt{Bob}, \texttt{Peter}\}$*, and* $\cdot^{\mathcal{I}}$ *the function defined by*

$$\begin{cases} \cdot^{\mathcal{I}}(x) = x \text{ if } x \in \{\texttt{Bob}, \texttt{Peter}, \texttt{Peng\_0}\}, \\ \cdot^{\mathcal{I}}(\texttt{Person}) = \{\texttt{Bob}, \texttt{Peter}\}, \\ \cdot^{\mathcal{I}}(\texttt{Penguin}) = \cdot^{\mathcal{I}}(\neg\texttt{CanFly}) = \{\texttt{Peng\_0}\}, \\ \cdot^{\mathcal{I}}(\texttt{hasChild}) = \cdot^{\mathcal{I}}(\texttt{hasParent}^{-}) = \{(\texttt{Bob}, \texttt{Peter})\}, \\ \cdot^{\mathcal{I}}(\exists\ \texttt{hasChild.Person}) = \{Peter\}. \end{cases}$$

Then, $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *is an interpretation as per Definition 2.3. Moreover,* $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ *satisfies all properties in Definition 2.9 with respect to the example knowledge base* $\mathcal{K}$*. It is therefore a model of* $\mathcal{K}$*.*

Definition 2.11 (Knowledge Base Consistency [120, 153, 154])**.** *A knowledge base* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ *is consistent if and only if it has a model.*

Example 2.12 (Knowledge Base Consistency)**.** *The knowledge base in Example 2.5 is consistent since it has a model (see Example 2.10). However, if one adds the assertion* $\neg\texttt{Person}(\texttt{Peter})$ *to the* `ABox` *of the example knowledge base, then the latter becomes inconsistent. The reason for this is that we would have two contradicting assertions:* $\neg\texttt{Person}(\texttt{Peter})$ *and* $\texttt{Person}(\texttt{Peter})$ *while* $\texttt{Person} \sqcap \neg\texttt{Person} \sqsubseteq \bot$ *(this is satisfied for every concept in general). Remember the assertion* $\texttt{Person}(\texttt{Peter})$ *is derived from* $\texttt{Person}(\texttt{Bob})$*,* $\texttt{hasChild}(\texttt{Peter}, \texttt{Bob})$*, and* $\exists\ \texttt{hasChild.Person} \sqsubseteq \texttt{Person}$*.*

Definition 2.13 (Concept Satisfiability [121, 153, 154])**.** *Let* $\mathcal{T}, C$ *be a* `TBox` *and concept, respectively.* C *is satisfiable if and only if there exists an interpretation* $\mathcal{I}$ *that maps* C *to a non-empty set. We say that* C *is satisfiable with respect to* $\mathcal{T}$ *if and only if there is a model of* $\mathcal{T}$ *for which* C *is satisfiable.*

Example 2.14 (Concept Satisfiability)**.** *Considering our knowledge base in Example 2.5, the concepts* `Penguin` *and* `Person` *are satisfiable. However, the concept* $\texttt{Penguin} \sqcap \texttt{CanFly}$ *is not satisfiable with respect to* $\mathcal{T}$ *due to* $\texttt{Penguin} \sqsubseteq \neg\texttt{CanFly}$*.*

Definition 2.15 (Concept Subsumption [121, 153, 154])**.** *Let* $\mathcal{T}$ *be a* `TBox`*,* $\mathcal{A}$ *an* `ABox`*, and* C *and* D *concepts. We say that* C *is subsumed by* D *and write* $C \sqsubseteq D$ *if and only if for every model* $\mathcal{I}$ *of* $\mathcal{A}$ *and* $\mathcal{T}$*,* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$*.* C *is subsumed by* D *with respect to* $\mathcal{T}$ *(denoted* $C \sqsubseteq_{\mathcal{T}} D$*) if and only if for every model* $\mathcal{I}$ *of* $\mathcal{T}$*,* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$*.*

When $C \sqsubseteq D$ *(respectively,* $C \sqsubseteq_{\mathcal{T}} D$*) and* $D \sqsubseteq C$ *(respectively,* $D \sqsubseteq_{\mathcal{T}} C$*), we say that* C *is equivalent to* D *and write* $C \equiv D$ *(respectively,* $C \equiv_{\mathcal{T}} D$*).*

Definition 2.16 (Role Subsumption). *Let $\mathcal{T}$ be a* `TBox`*, $\mathcal{A}$ an* `ABox`*, and r and s roles. r is subsumed by s (denoted $r \sqsubseteq s$) if and only if for every model $\mathcal{I}$ of $\mathcal{T}$ and $\mathcal{A}$, $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$.*

When $r \sqsubseteq s$ and $s \sqsubseteq r$, we say that r is equivalent to s and write $r \equiv s$.

Example 2.17 (Concept Subsumption, Role Subsumption). *In the* `TBox` *of the knowledge base in Example 2.5, the concept* `Penguin` *is subsumed by the concept* $\neg$`CanFly`*, that is, no penguin can fly. In the same* `TBox`*,* $\texttt{hasChild} \equiv \texttt{hasParent}^{-}$ *is a double role subsumption, that is,* $(\texttt{hasChild} \sqsubseteq \texttt{hasParent}^{-}) \wedge (\texttt{hasParent}^{-} \sqsubseteq \texttt{hasChild})$.

Subsumptions allow to build the terminological box of knowledge bases and to reason about the membership of individuals to concepts (see instance retrieval check) and the relationships that they share.

The notions of instance check and instance retrieval are based on entailments, which we introduce first.

Definition 2.18 (`TBox` Entailment). *Let $\mathcal{T}$ be a* `TBox`*, asn an assertion, and axm an axiom. $\mathcal{T}$ entails asn (respectively, $\mathcal{T}$ entails axm) denoted $\mathcal{T} \models asn$ (respectively, $\mathcal{T} \models axm$) if and only if for every model $\mathcal{I}$ of $\mathcal{T}$ asn is satisfied (respectively, axm is satisfied) with respect to $\mathcal{I}$.*

Definition 2.19 (`ABox` Entailment). *Let $\mathcal{A}$ be an* `ABox` *and asn an assertion. $\mathcal{A}$ entails asn denoted $\mathcal{A} \models asn$ if and only if for every model $\mathcal{I}$ of $\mathcal{A}$ asn is satisfied with respect to $\mathcal{I}$.*

Definition 2.20 (Knowledge Base Entailment). *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, asn an assertion in $\mathcal{A}$, and axm an axiom in $\mathcal{T}$. We say that $\mathcal{K}$ entails asn (respectively, $\mathcal{K}$ entails axm) and write $\mathcal{K} \models asn$ (respectively, $\mathcal{K} \models axm$) if and only if $\mathcal{T} \models asn$ or $\mathcal{A} \models asn$ (respectively, $\mathcal{T} \models axm$).*

Definition 2.21 (Instance Check [120, 153, 154]). *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base, C a concept, and a an individual. a is an instance of C with respect to $\mathcal{A}$ (respectively, $\mathcal{K}$) if and only if $\mathcal{A} \models C(a)$ (respectively, $\mathcal{K} \models C(a)$).*

Definition 2.22 (Instance Retrieval [120, 153, 154]). *Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ be a knowledge base and C a concept. The instance retrieval of C with respect to $\mathcal{A}$ denoted $R_{\mathcal{A}}(C)$ is the set of all individuals a that satisfy $\mathcal{A} \models C(a)$.*

Similarly, the instance retrieval of C with respect to $\mathcal{K}$ denoted $R_{\mathcal{K}}(C)$ is the set of all individuals a that satisfy $\mathcal{K} \models C(a)$.

Note that the TBox alone cannot entail the membership of an individual to a concept; one would need additional information from the ABox. This is the reason why Definition 2.22 does not introduce the notion of instance retrieval with respect to the TBox.

Example 2.23 (Instance Check, Instance Retrieval)**.** *We consider the knowledge base in Example 2.5. Bob is a person according to the ABox (note the assertion Person(Bob) in $\mathcal{A}$). Contrarily, $\mathcal{A} \not\models$ Person(Peter), but $\mathcal{K} \models$ Person(Peter) (via Person(Bob), hasChild(Peter, Bob), and $\exists$ hasChild.Person $\sqsubseteq$ Person).*

2.4 Refinement Operators

Definition 2.24 (DL Refinement Operator [113, 121, 159])**.** *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 $C \preceq C'$). ρ is called DL refinement operator if $\mathcal{S}$ is a set of DL concepts, and $\preceq$ is the concept subsumption.*

Example 2.25 (DL Refinement Operator)**.** *Consider the knowledge base $\mathcal{K}$ in Example 2.27. Let $\mathcal{S} = (\{\top, \bot,$ Person, Penguin, $\neg$CanFly, $\exists$ hasChild.Person, $\exists$ hasWing.$\top\}, \sqsubseteq)$ and let $\rho : \mathcal{S} \to 2^{\mathcal{S}}$ be defined by:*

$$\rho(C) = \begin{cases} \mathcal{S} \text{ if } C = \top, \\ \emptyset \text{ if } C = \bot, \\ \{C\} \cup \{A \in \mathcal{S} | \mathcal{K} \models A \sqsubseteq C\} \text{ otherwise.} \end{cases}$$

ρ is a downward refinement operator.

Refinement operators can have a number of desirable properties: *completeness*, *properness*, *idealness*, and *non-redundancy* to mention only a few. We refer the reader to [120] for a theoretical analysis of refinement operators. Search-based approaches, e.g., CELOE [122], employ a refinement operator and a heuristic function to construct and traverse their search space.

Connection to Our Contributions. Our scientific contributions tackle class expression learning in description logics, where a learning problem is essentially defined using a DL knowledge base as background knowledge. To solve a learning problem, search-based methods (including our approach CLIP) often rely on refinement operators and heuristic functions to traverse their search space. Moreover, our neural class expression synthesizers employ a refinement operator to generate their training data on each input knowledge base. Therefore, our contributions are inherently linked to the notions of description logic, DL knowledge base, and refinement operator.

Having covered various notions on description logics and knowledge bases, we will now introduce key learning tasks on those, starting with class expression learning (also known as concept learning), which is at the core of this thesis, and briefly touching related tasks such as ontology learning.

2.5 Class Expression Learning

Class expression learning can be regarded as a symbolic machine learning task [191, 241] where the goal is to describe a set of individuals (examples) using description logic concepts. This is because a class expression is represented as a sequence of symbols (e.g., atomic concepts, roles, quantifiers) which are human-readable and interpretable [111], as opposed to neural networks which are black-box. Nonetheless, neural networks are *fast-thinkers* (i.e., they can yield approximate solutions within a fraction of a second), robust to noisy/imperfect inputs, and can handle unstructured data [241]. Our research contributions are *neuro-symbolic* in the sense that they combine the strengths of neural networks with the clear semantics of description logics for scalable class expression learning.

Classical Definition

Below, we give a classical definition of class expression learning in description logics. Later in Chapter 7, we will propose a modification of this definition to integrate desired properties in solutions to learning problems.

Definition 2.26 (Classical Learning Problem ($\mathcal{CLP}$) [107, 121])**.** *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 $\mathcal{K}'_C = \mathcal{K} \cup \{T \equiv C\}$, we have that $\mathcal{K}'_C \models C(E^+)$ and $\mathcal{K}'_C \not\models C(E^-)$.*

In the above definition, we write $\mathcal{K}'_C \models C(E)$ (respectively, $\mathcal{K}'_C \not\models C(E)$) to express that $\forall\ e \in E,\ \mathcal{K}'_C \models C(e)$ (respectively, $\forall\ e \in E,\ \mathcal{K}'_C \not\models C(e)$). We denote the classical learning problem defined by $\mathcal{K}$, T, E^+, and E^- by $\mathcal{CLP}(\mathcal{K}, T, E^+, E^-)$.

Example 2.27 (Class Expression Learning)**.** *Consider the knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, where*

$$\begin{aligned}\mathcal{T} =&\{\mathit{Penguin} \sqsubseteq \neg \mathit{CanFly}, \mathit{hasParent} \equiv \mathit{hasChild}^-,\\ &\exists\ \mathit{hasChild}.\mathit{Person} \sqsubseteq \mathit{Person}\},\\ \mathcal{A} =&\{\mathit{Person}(\mathit{Bob}), \mathit{CanFly}(\mathit{Eagle\_0}), \mathit{Penguin}(\mathit{Peng\_0}), \mathit{hasChild}(\mathit{Peter}, \mathit{Bob}),\\ &\mathit{hasWing}(\mathit{Peng\_0}, \mathit{Wing\_0}), \mathit{hasWing}(\mathit{Eagle\_0}, \mathit{Wing\_1})\}.\end{aligned}$$

Let $E^+ = \{\mathit{Peng\_0}\}$, $E^- = \{\mathit{Eagle\_0}, \mathit{Bob}, \mathit{Peter}\}$, $T = \mathit{Penguin}$. Then, $\mathcal{CLP}(\mathcal{K}, T, E^+, E^-)$ is a learning problem and each of the following class expressions is a solution

1. $\neg \mathit{CanFly}$,
2. $\neg \mathit{CanFly} \sqcap \exists\ \mathit{hasWing}.\top$.

Although the first solution may seem unnatural (because humans cannot fly either and therefore `Bob` and `Peter` could be instances of $\neg$`CanFly`), there is no information in $\mathcal{K}$ preventing it from being valid. In comparison, the second solution describes penguins better; not only they cannot fly but also they have wings.

The terminological box of many knowledge bases, which is essential for class expression learning, is often constructed by extracting information from one or multiples data sources, e.g., unstructured text. This process is known as ontology learning, and formally defined below.

Related Tasks

Ontology learning (OL) is a task closely related to class expression learning. It is defined as the automatic or semi-automatic creation of ontologies using a pre-existing source of information such as natural language text [226]. This creation process involves the extraction of domain-specific terms or concepts and the links between them. Links between concepts are represented using hierarchical data structures, e.g., class inclusions (CIs) in description logics. As an example, the statement "penguins are birds that can swim or animals that have wings and cannot fly" would be translated into `Penguin` $\sqsubseteq$ (`Bird` $\sqcap$ `CanSwim`) $\sqcup$ (`Animal` $\sqcap$ `HasWings` $\sqcap$ $\neg$`CanFly`). As a result, CI learning is a fundamental component in OL. CI learning

approaches are tasked to build a hierarchy between concepts in the provided source of information. A formal definition of *class inclusion* (basically the same as *concept subsumption*) is provided in Definition 2.15. Neural network-based approaches for CI [135, 165] leverage NLP techniques [29] to translate text into description logic concept hierarchies, which are essential for DL knowledge base `TBoxes`. Inductive logic programming (ILP) approaches [148] learn CIs from positive and negative examples. These examples are provided by an oracle, e.g., a domain expert or even a neural network [221]. Association rule mining (ARM) approaches [5, 65, 143] rely on support and confidence score functions to learn CIs.

Class subsumption axioms learned in OL can be used to enrich the `TBox` of relevant knowledge bases. Class expression learning also has applications in knowledge base enrichment (specifically, ontology engineering [122]), as new axioms about the learned class expressions (e.g., `France` is an instance of the expression $\exists$ `hasOrganizedOlympics` $\sqcap \geq 16$ `hasWon.{GoldMedal}`) can be added to the `TBox` or `ABox` of fitting knowledge bases. In this thesis, we focus on class expression learning from given positive and negative examples. We propose approaches that employ knowledge graph embedding techniques to obtain continuous vector representations of input examples, and deep learning techniques to efficiently learn class expressions from these embeddings. Below, we introduce the notions of "knowledge graph" and "knowledge graph embedding", and present some of the existing techniques for representation learning on knowledge graphs.

2.6 Knowledge Graph Embeddings

A Knowledge graph (KG) is a "graph of data intended to accumulate and convey knowledge of the real world, whose nodes represent entities of interest and whose edges represent relations between these entities" [86]. They can also be defined as collections of assertions in the form of triples. In the knowledge graph in Figure 2.6, an example fact is "Da Vinci painted Mona Lisa" which can be represented as a triple: $(\text{Da Vinci}, \text{painted}, \text{Mona Lisa})$. Triples that are present in a given knowledge graph are often said to be correct, assuming the knowledge graph contains no false information. In this context, knowledge graphs can be used to facilitate information retrieval [55, 132, 173], question answering [92, 239], and even retrieval-augmented generation (RAG) with LLMs [141, 233]. For instance, the question "*Who painted Mona Lisa?*" can be efficiently answered using our example knowledge graph in Figure 2.6, as the relevant fact is present in the graph. For this question, the answer is "Da Vinci". To further facilitate applications in other

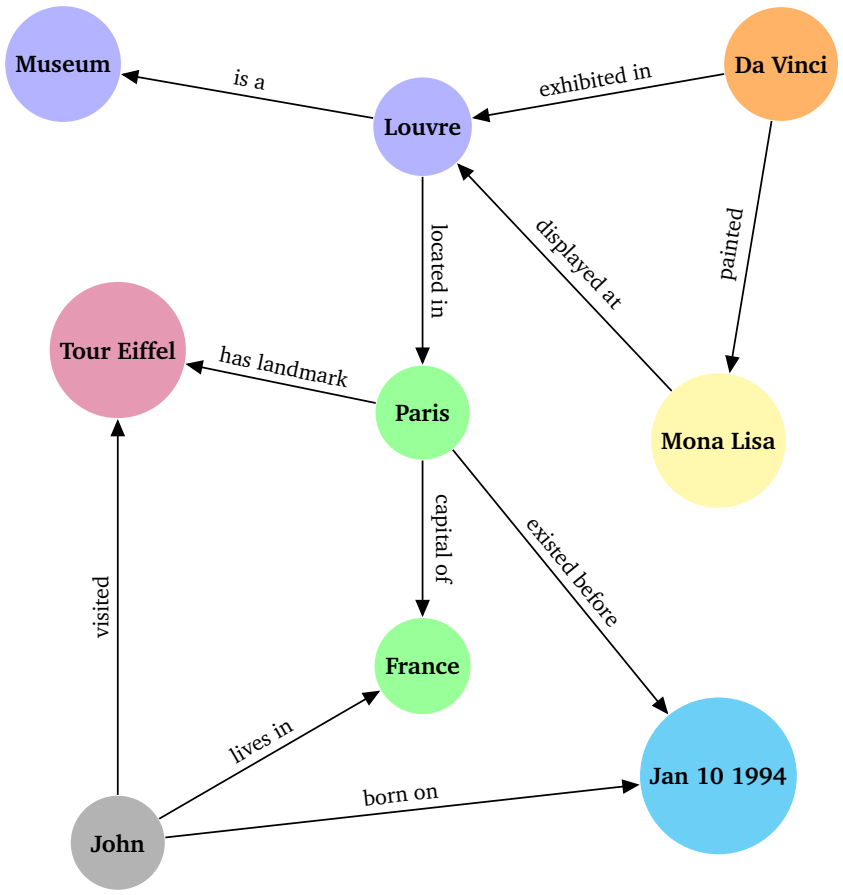


Figure 2.6: Example knowledge graph

downstream tasks, KGs are often projected onto continuous vector spaces such as $\mathbb{R}^d$. This process is known as *knowledge graph embedding* (KGE) [41, 49, 215, 230].

Several KGE techniques have been developed ever since the introduction of the pioneering approaches. Basically, embeddings are computed by learning the vector representations of nodes and relation types in a way that the relationships between nodes can be established by using the learned vectors. Knowledge graph embeddings are useful for downstream tasks such as link prediction [23], knowledge completion [129], recommendation systems [243], and natural language processing [33]. Some KGE approaches solely use facts observed in the input knowledge graph [24, 157] while others leverage additional information about entities and relations, such as textual descriptions [219, 229] or `sameAs` links to external sources [110]. Our synthesis-based approaches for class expression learning (NCES [113], NCES2 [109], and ROCES [111]) use KGEs to compute solutions to learning problems without a search process. We experimented with several embedding models, including ConEx [45] and TransE [23]. ConEx applies convolutions

Table 2.2: State-of-the-art KGE models. d denotes the dimension of an embedding space. $\mathbf{e}$ denotes an embedding, and $\overline{\mathbf{e}} \in \mathbb{C}^d$ its complex conjugate. $*, f, [;], \text{vec}(), \text{vec}()^{-1}$, and $\triangleleft$ denote convolution operation, rectified linear unit (ReLU), flattening, reshaping of a vector into a matrix, and unit vector normalization, respectively. $\circ, \otimes, \cdot$ denote Hadamard, Hamilton, and inner products, respectively. $\text{conv}(\cdot,\cdot)$ denotes a 2D convolution operation followed by affine transformations. For KECI, $m = d/(1+p+q)$, and for DECAL, $m = d/(1+p+q+r)$ with $p, q, r \in \mathbb{N} \setminus \{0\}$.

Model	Scoring Function	Vector Space
RESCAL [156]	$\mathbf{e}_h \cdot \mathcal{W}_r \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_t \in \mathbb{R}^d, \mathcal{W}_r \in \mathbb{R}^{d \times d}$
TransE [23]	$\|\mathbf{e}_h + \mathbf{e}_r - \mathbf{e}_t\|$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{R}^d$
DistMult [235]	$\mathbf{e}_h \cdot (\mathbf{e}_r \circ \mathbf{e}_t)$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{R}^d$
ComplEx [207]	$\text{Re}\left(\sum_{i=1}^{d} \mathbf{e}_{r_i} \mathbf{e}_{h_i} \overline{\mathbf{e}_{t_i}}\right)$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{C}^d$
ConvE [53]	$f(\text{vec}(f([\mathbf{e}_h; \mathbf{e}_r] * \omega)) \cdot \mathbf{W}) \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{R}^d$
HyPER [17]	$f(\text{vec}(\mathbf{e}_h * \text{vec}^{-1}(\mathbf{e}_r \cdot \mathbf{H})) \cdot \mathbf{W}) \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{R}^d$
RotatE [197]	$\|\mathbf{e}_h \circ \mathbf{e}_r - \mathbf{e}_t\|$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{C}^d$
QuatE [246]	$\mathbf{e}_h \otimes \mathbf{e}_r^{\triangleleft} \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{H}^d$
CONEX [45]	$\text{Re}\left(\langle \text{conv}(\mathbf{e}_h, \mathbf{e}_r), \mathbf{e}_h, \mathbf{e}_r, \overline{\mathbf{e}_t} \rangle\right)$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in \mathbb{C}^d$
KECI [46]	$(\mathbf{e}_h \circ \mathbf{e}_r) \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in Cl_{p,q}(\mathbb{R}^m)$
DECAL [100]	$(\mathbf{e}_h \circ \mathbf{e}_r) \cdot \mathbf{e}_t$	$\mathbf{e}_h, \mathbf{e}_r, \mathbf{e}_t \in Cl_{p,q,r}(\mathbb{R}^m)$

on complex-valued vector representations of nodes and relation types to model interactions, whereas TransE models interactions as translations of head entities to tail entities via relation type embeddings. In Table 2.2, we present the scoring technique of some knowledge graph embedding models as well as their embedding spaces.

All models in Table 2.2 belong to the family of transductive KGE models. They learn embeddings only for entities and relations present in a given KG, and cannot handle unseen entities or relations. To overcome these limitations, other types of embedding approaches (known as inductive approaches) have been proposed. Such approaches learn embeddings in a way that can be transferred to unseen entities or relations. Examples include GraphSAGE [78], LAN [214], NodePiece [66], ULTRA [67], and BytE [51]. Specifically, GraphSAGE learns entity embeddings by aggregating information from their neighbors. LAN further attaches attention weights to neighbors to model their importance. NodePiece assumes a predefined set of anchor nodes, and represents an entity as a hash of its closest anchor nodes and its immediate outgoing relation types. ULTRA introduces a graph of relations to learn interactions between relations in the original graph. BytE uses a pretrained LLM's tokenizer to transform tranductive KGE models into inductive ones. To this end, it decomposes every entity and relation into a sequence of subword units and applies a token level ensembling (e.g., using a linear mapping) to construct a low-dimensional embedding vector as input to any transductive KGE model. As

opposed to the aforementioned inductive approaches which require a subgraph around a given unseen triple, BytE can handle unseen triples without any additional information.

Connection to Our Contributions. Our approaches leverage knowledge graph embeddings to learn mappings between sets of individuals and class expressions or class expression lengths. Specifically, NCES, NCES2 and ROCES use knowledge graph embeddings to synthesize class expressions in $\mathcal{ALC}$, and $\mathcal{ALCHID}^{(\mathcal{D})}$, respectively. CLIP takes embeddings of positive and negative examples to predict the lengths of the corresponding target class expressions.

3 Related Work

This chapter is composed of two sections. In the first section, we present some of the existing works in inductive logic programming (ILP), a research field closely related to class expression learning in the way its learning problems are formulated. We also include works that apply ILP techniques to tackle class expression learning, and discuss their limitations. In the second section, we present state-of-the-art approaches for class expression learning, and discuss their strengths and limitations, and how they related to our proposed approaches.

3.1 Inductive Logic Programming

Inductive logic programming [148, 159] is a branch of machine learning that focuses on building models based on examples and background knowledge. It utilizes logic programming languages such as Prolog [37, 38], which is a declarative language, to represent hypotheses (e.g., rules) and data (e.g., existing facts or examples). The primary goal in ILP is to infer general rules from examples (usually a set of positive examples and a set of negative examples), enabling systems to learn patterns and relationships within complex datasets. By leveraging both positive and negative examples, ILP approaches can generate logic-based theories that can be applied to various domains, including bioinformatics [162], automated reasoning, and social network analysis. Several approaches have been proposed for inductive learning tasks. Each approach is characterized by its unique methodologies and techniques in addressing specific aspects of a learning problem. Below, we present some of the most prominent approaches in ILP.

3.1.1 First Order Inductive Learner

FOIL (First Order Inductive Learner) [170, 171] is a flexible system for learning function-free Horn clause [89] definitions of relations (n-ary predicates) from examples. An input to FOIL is a relational database $\mathcal{K}$ (often given as a set of tuples) and a target relation R. The target relation R comes with its known tuples (positive

Algorithm 1 FOIL algorithm [171]

Input: $\{\mathcal{K}, (R, \bigoplus, \bigotimes)\}$
Output: $C^* := R(V_1, V2, \ldots, V_m) \leftarrow L_1, L_2, \ldots, L_p$

1: Initialize the training set $T = \{\bigoplus, \bigotimes\}$
2: Initialize a target clause $C^* = R(V_1, V2, \ldots, V_m) \leftarrow$
3: **while** T contains positive tuples **do**
4: Find a clause $R(V_1, V2, \ldots, V_m) \leftarrow L_j$ that characterizes some part of the target R
5: Form a local training set $T_j = \{\bigoplus_j, \bigotimes_j\}$ were all tuples that satisfy the right-hand side of the clause are removed
6: Append L_j to the right-hand side of C^*
7: $L_{j_1}, L_{j_2}, \ldots, L_{j_k} =$ INNERLOOP$(\mathcal{K}, T_j, R)$
8: Append $L_{j_1}, L_{j_2}, \ldots, L_{j_k}$ to the right-hand side of C^*
9: Remove tuples in T_j that satisfy any of $L_{j_1}, L_{j_2}, \ldots, L_{j_k}$
10: Reset the training set $T = T_j$
11: **end while**
12: **return** the clause $C^* := R(V_1, V2, \ldots, V_m) \leftarrow L_1, L_2, \ldots, L_p$

tuples $\bigoplus$). Additional information may be given for the target relation such as tuples known not to belong to it (negative tuples $\bigotimes$)[1]. Given an input $\{\mathcal{K}, (R, \bigoplus, \bigotimes)\}$, the goal is to find a set of Horn clauses (about the target relation R) that covers all of the positive tuples $\bigoplus$, but none of the negative tuples $\bigotimes$. The construction of a solution by FOIL is described in Algorithms 1 and 2.

In Algorithm 1, FOIL looks for clauses that cover positive tuples. The algorithm starts by initializing a clause with an empty body, and a training set composed of $\bigoplus$ and $\bigotimes$. At each iteration, it finds a literal to append to the right-hand side of the running clause. The goal is to ensure that each clause covers as much positive tuples as possible. Once a clause is chosen, it removes all tuples that are currently covered, keeps the remaining tuples as local training set, and enters the inner loop (Algorithm 2). At the conclusion of the inner loop, the search process restarts with a new training set where all tuples that are covered by any previous literal in C^* are removed.

In the inner loop (Algorithm 2), FOIL seeks clauses that cover local positive tuples and eliminate negative tuples. The algorithm starts with the local training set in Step 5 of Algorithm 1, and a close with an empty body. At each iteration, a literal is chosen and appended to the right-hand side of the running clause. The construction of a new training set then begins: any tuple from the current training set whose extension with the bindings of the literal's new variables satisfies the

[1]When negative tuples are not available, the closed-world assumption (CWA) may be invoked to create them.

Algorithm 2 INNERLOOP [171]

Input: $\{\mathcal{K}, T_j, R\}$
Output: $L_{j_1}, L_{j_2}, \ldots, L_{j_k}$

Initialize a clause $R(V_1, V2, \ldots, V_m) \leftarrow$
Initialize the training set $T = T_j$
while T contains negative tuples and is not too complex **do**
    Find a literal L_{j_i} to add to the right-hand side of the clause
    Form a new training set $T' = \{\}$:
    for any tuple t in T **do**
        for any binding b of new variables introduced by L_{j_i} **do**
            if the tuple $t.b$ (concatenation of t and b) satisfies L_{j_i} **then**
                Add $t.b$ to T' with the same label as t
            end if
        end for
    end for
    Replace the training set T by T': $T = T'$
end while
Prune the clause by removing any unnecessary literals
return the literals $L_{j_1}, L_{j_2}, \ldots, L_{j_k}$

literal is extended with those bindings and added to the new training set along with its label. The execution of the algorithm stops when there are no more negative tuples, or when the training set is overly complex (e.g., its tuples become too long). The output of the inner loop is a sequence of literals which are appended to the goal clause C^* in Algorithm 1. The choice of this sequence of literals is carried out by using a heuristic function based on information gain. We refer the reader to the original paper [170] for further details.

Despite FOIL's backup mechanism, and the high-quality heuristics in its exploration strategy, there are cases in which it is expected to be considerably slow. For example, Pazzani and Kibler [163] have shown that the number of literals to add to the goal clause grows exponentially with the arity of predicates and the number of variables. Moreover, the number of examples in local training sets can get large although this number is bounded by the power of predicates.[2] Pazzani and Kibler [163] therefore proposed to add knowledge into the learning framework of FOIL, resulting in a new system called FOCL, that we elucidate next.

[2]The power of a predicate in this sense is the maximum number of possible bindings (i.e., variable assignments) when one variable is fixed.

3.1.2 First Order Combined Learner

FOCL (First Order Combined Learner) [163] is an extension of FOIL that adds domain knowledge into the exploration strategy of the latter. More specifically, FOCL comes with three main extension options: 1) use constraints to limit the search space of FOIL, 2) use intensionally defined predicates (also called non-operational predicates), and 3) accept as input a partial, possibly incorrect rule that approximates the target predicate. Additionally, FOCL introduces an iterative widening search strategy to reduce the exponential exploration cost of FOIL. In iterative widening, the learning process starts with no free variables. If this fails with zero variabilization gain, then additional free variables can be considered. On the subsequent learning steps, each failure is granted access to one free variable until the maximum arity of a predicate is reached. In Algorithms 3, 4, and 5, we summarize the learning procedure of FOCL.

In Algorithms 3–5, $\mathcal{K}$ denotes a relational database together with intensionally defined predicates and constraints. IR is the initial rule that approximates the target relation, and $\bigoplus, \bigotimes$ are the sets of positive and negative tuples, respectively. The main algorithm (3) iteratively calls the clause learning algorithm (4) which in turn calls the body extension algorithm (5). At every iteration, the goal is to find a clause body that covers as much positive tuples as possible while discarding many negative tuples. The convergence speed of the overall algorithm as well as the quality of the computed clauses depend mainly on the quality of the approximation rule IR, and that of the constraints (e.g., typing knowledge). High-quality intensionally defined predicates can also boost the overall performance of the learning algorithm. However, one should be cautious when adding those predicates as a large number of them often increases the search space.

Algorithm 3 FOCL algorithm [163]

Input: $\mathcal{K}, \bigoplus, \bigotimes, IR$
Output: $ClauseBody$

1: Initialize empty clause body $ClauseBody = \ldots$
2: **while** $\bigoplus$ is not empty **do**
3: $\quad Body = \textsc{LearnClauseBody}(\mathcal{K}, \bigoplus, \bigotimes, IR, ClauseBody)$
4: $\quad$ Remove from $\bigoplus$ and $\bigotimes$ those tuples covered by $Body$
5: $\quad$ Set $Body$ to empty
6: **end while**
7: **return** $ClauseBody$

Both FOIL and FOCL are very specific when it comes to the definitions they can build for a given target relation: they are restricted to function-free Horn rules. This limits

Algorithm 4 LEARNCLAUSEBODY [163]

Input: $\mathcal{K}, \bigoplus, \bigotimes, IR, ClauseBody$
Output: $Body$

if any $subBody$ in IR has positive gain **then**
  Choose best $subBody$
  Operationalize and delete superfluous literals from it
  Conjoin result with $ClauseBody$
  Update $\bigoplus$ and $\bigotimes$ by removing tuples covered by the best $subBody$
  $Body = \text{EXTENDBODY}(\mathcal{K}, \bigoplus, \bigotimes, IR, ClauseBody)$
else
  Choose best non-operational *literal*
  Operationalize and delete superfluous literals from it
  Conjoin result with $ClauseBody$
  Update $\bigoplus$ and $\bigotimes$ by removing tuples covered by best *literal*
  $Body = \text{LEARNCLAUSEBODY}(\mathcal{K}, \bigoplus, \bigotimes, IR, ClauseBody)$
end if
return $Body$

Algorithm 5 EXTENDBODY [163]

Input: $\mathcal{K}, \bigoplus, \bigotimes, IR, ClauseBody$
Output: $Body$

while $\bigotimes$ is not empty **do**
  Choose best non-operational *literal*
  Operationalize and delete superfluous literals from it
  Conjoin the resulting $Body$ with $ClauseBody$
  Update $\bigoplus$ and $\bigotimes$ by removing tuples covered by $Body$
end while
return $Body$

their applicability to a wide range of related learning tasks. The clausal discovery algorithm CLAUDIEN, described below, provides a declarative language bias to learn regularities from Herbrand interpretations.

3.1.3 Clausal Discovery Engine

CLAUDIEN (Clausal Discovery Engine) [43] is an engine that combines inductive logic programming and descriptive data mining techniques to address characteristic induction from interpretations. More specifically, it finds logically maximally general hypotheses (represented as clausal theories) from Herbrand interpretations. A Herbrand interpretation corresponds to a set of ground truths in first order logic, or a set of statements (e.g., `{Person(Bob), loves(Bob, Music)}`) in description logics.

Algorithm 6 ClausalDiscovery [43]

Input: $\mathcal{O}, \rho$
Output: Characteristic hypothesis H

```
1:  H = ∅
2:  Q = {□}
3:  while Q ≠ ∅ do
4:     Get c from Q and delete c from Q
5:     if c is valid on O and not prune1(c) then
6:        Add c to H
7:     else
8:        for c' in ρ(c) do
9:           if not prune2(c') then
10:             Add c' to Q
11:          end if
12:       end for
13:    end if
14: end while
15: reduce H
16: return H
```

Given a language $\mathcal{L}$ equipped with a declarative language bias $\mathcal{D}_{LAB}$, a set of closed observations $\mathcal{O}$ (Herbrand interpretations), and the most general hypothesis $\square \in \mathcal{L}$, $\mathcal{C}$LAUDIEN iteratively applies a refinement operator ρ [3] to construct a set of target hypotheses H. In Algorithm 6, we describe the learning procedure of $\mathcal{C}$LAUDIEN. The algorithm starts with an empty hypothesis set H and a queue Q containing the most general hypothesis $\square$. It then iteratively calls the functions *delete*, *valid*, *prune1*, and *prune2* on the refinements of $\square$. Each of these functions plays a crucial role in the overall behaviour of the algorithm. For example, if the function *delete* is *first-in-first-out*, then the algorithm runs a breadth-first search, when it is *first-in-last-out*, it runs a depth-first search, and when it is according to a ranking criterion of the clauses, the algorithm runs a best-first search. The function *valid* checks for the validity of a clause as solution; *prune1* ensures that the clause is maximally general, and *prune2* enforces the clause's injectivity, non triviality, and non redundancy. Finally, the function *reduce* enforces the compactness of the set of clauses H by employing the theorem prover SATCHMO [138].

Many other inductive learning algorithms exist [116, 146, 147, 149, 150, 195], but they are similar to those presented above in the way they compute solutions. For instance, Progol [149] employs a covering approach like FOIL, and computes a solution by progressively generalizing an initial clause which is pre-selected based

[3] ρ is a mapping from $\mathcal{L}$ to $2^{\mathcal{L}}$ which refines a given hypothesis into a set of smaller (in the sense of θ-subsumption) hypotheses.

on seed examples. The popular Aleph [195] system is a successor of Progol whose main aim is to serve as a prototype for exploring ideas and simulating other ILP systems's behaviors. We refer to respective references for more details on each ILP system, and to *List of ILP Systems*[4] for a non-exhaustive list of known ILP systems.

Having reviewed some of the existing works in inductive logic programming, we can now explore the state-of-the-art approaches for class expression learning in description logics, which is the main focus of this thesis.

3.2 State of the Art in Class Expression Learning

In class expression learning, two sets of examples (positive and negative examples) are given, and the goal is to find a description logic expression that describes positive examples, but none of the negative examples [82, 107, 113, 122]. Class expression learning has applications in bio-medicine [22, 81, 124, 187], ontology engineering [122], and Industry 4.0 [50]. Several approaches have been proposed to tackle class expression learning. Some of these approaches use techniques from inductive logic programming, e.g., DL-FOIL [62], and DL-FOCL [177] are direct adaptations of FOIL, and FOCL, respectively. Others use evolutionary algorithms, DL refinement operators, set operations, or deep Q-learning to construct and traverse their search space. Below, we present some of the most prominent approaches for class expression learning and discuss their strengths and limitations.

3.2.1 Description Logic First Order Inductive Learner

DL-FOIL (Description Logic First Order Inductive Learner) [62] is an adaptation of FOIL for class expression learning in description logics. Given sets of positive, negative, and unlabeled examples, DL-FOIL computes partial generalizations (DL concepts) which aim to cover as many positive examples as possible, while avoiding negative examples. Whenever a partial generalization covers negative examples, a specialization routine is invoked to refine the generalization and discard negative examples. We provide a pseudo-code of DL-FOIL in Algorithms 7 and 8.

Algorithm 7 starts by initializing a generalization with the bottom concept $\bot$, and the set of positives to cover are the initial positive examples $Positives$. It then iteratively constructs partial definitions (in the same way FOIL constructs its clauses) which

[4] Accessed on January 6th, 2025: `http://www-ai.ijs.si/~ilpnet2/systems/`

cover some part of positive examples, and adds them to the initial generalization via the disjunction operator $\sqcup$. During the construction of a partial definition, a specialization function (SPECIALIZE, see Algorithm 8) is called whenever negative examples are covered. This ensures that the final generalization does not cover any of (or most of) the negative examples.

Algorithm 7 DL-FOIL [62]

Input: $\mathcal{K}, Positives, Negatives, Unlabeled$
Output: $Generalization$: concept definition

1: $Generalization = \bot$
2: $PositivesToCover = Positives$
3: **while** $PositivesToCover \neq \emptyset$ **do**
4: $\quad PartialDef = \top$
5: $\quad CoveredNegatives = Negatives$
6: $\quad$ **while** $CoveredNegatives \neq \emptyset$ **do**
7: $\quad\quad PartialDef$ = SPECIALIZE($\mathcal{K}, PartialDef, PositivesToCover, CoveredNegatives, Unlabeled$)
8: $\quad\quad CoveredNegatives = \{n \in Negatives \mid \mathcal{K} \models PartialDef(n)\}$
9: $\quad$ **end while**
10: $\quad CoveredPositives = \{p \in PositivesToCover \mid \mathcal{K} \models PartialDef(p)\}$
11: $\quad Generalization = Generalization \sqcup PartialDef$
12: $\quad PositivesToCover = PositivesToCover \setminus CoveredPositives$
13: **end while**
14: **return** $Generalization$

The function SPECIALIZE looks for refinements (specializations or partial definitions) of a given generalization that achieve a minimum gain ($MINGAIN$). The gain of a refinement covering p_1 positive examples and n_1 negative examples, with u_1 unlabeled examples, is defined as

$$gain = p_1 \left(\log \frac{p_1 + w_1 u_1}{p_1 + n_1 + u_1} - \log \frac{p_0 + w_0 u_0}{p_0 + n_0 + u_0} \right), \tag{3.1}$$

where p_0 and n_0 represent the number of positive and negative examples covered by the previous refinement, with u_0 unlabeled examples, respectively. The weights w_0 and w_1 are determined by a prior probability distribution of positive examples. This definition of "gain" encourages refinements to cover many positives and discard many negatives.

Evaluated on the BIOPAX, Semantic Bible (NTN), and FINANCIAL ontologies, DL-FOIL shows a relatively good predictive performance in a ten-fold-cross-validation setting, with average F_1 scores ranging from 59.1% to 69.6%. One limitation of DL-FOIL is its *scalability* to large datasets. As pointed out in our work [107], DL-FOIL does not terminate in a reasonable amount of time on medium to large datasets such

Algorithm 8 Function SPECIALIZE [62]

Input: $\mathcal{K}, PartialDef, PositivesToCover, CoveredNegatives, Unlabeled$
Output: $Refinement$: concept definition
Parameters: $MAXNUM, MINGAIN, NUMSPECS$

$bestGain = -MAXNUM$
while $bestGain < MINGAIN$ **do**
  for $i = 1, 2, \ldots, NUMSPECS$ **do**
    $specialization =$ GETRANDOMREFINEMENT$(\rho, PartialDef)$
    $CoveredPositives = \{p \in PositivesToCover \mid \mathcal{K} \models PartialDef(p)\}$
    $Negatives = \{n \in CoveredNegatives \mid \mathcal{K} \models PartialDef(n)\}$
    $thisGain =$ GAIN $(CoveredPositives, Negatives, Unlabeled, PositivesToCover, CoveredNegatives)$
    if $thisGain > bestGain$ **then**
      $Refinement = specialization$
      $bestGain = thisGain$
    end if
  end for
end while
return $Refinement$

as Carcinogenesis. Moreover, it may enter infinite loops during the search process: in [107], it has been reported that DL-FOIL made over five thousand unsuccessful attempts in the refinement of the concept `GeographicLocation` with respect to some input positive and negative examples. The approaches we propose in this thesis, namely NCES, NCES2, and ROCES, are guaranteed never to enter such infinite loops as they compute solutions without a search process, but rather via fast matrix and vector operations.

3.2.2 Description Logic First Order Combined Learner

Just as FOCL is a knowledge-enhanced and faster version of FOIL, DL-FOCL (Description Logic First Order Combined Learner) [177] is a better (faster or more accurate depending on the extension version, see below) alternative to DL-FOIL which employs meta-heuristics to reduce the search space. DL-FOCL comes with three versions: DL-FOCL I, DL-FOCL II, and DL-FOCL III, each with a different focus in the overall improvement.

DL-FOCL I employs a repeated sequential covering strategy to improve the predictive performance of a generalization. In other terms, after a solution is computed by DL-FOIL, DL-FOCL I checks its predictive performance and, if unsatisfactory (less than a given threshold), re-runs the DL-FOIL algorithm an additional number of

times until the desired quality is met. Indeed, DL-FOCL I is slower than DL-FOIL, but is often more accurate. With the so-called lookahead strategy, DL-FOCL II considers refinements at higher levels, i.e., which cannot be reached with a single refinement step. More specifically, at each step of the search process, DL-FOCL II evaluates direct refinements and their k-th grand children to mitigate myopia in the search space. This allows DL-FOCL II to directly choose future refinements as candidate solutions, and hence speeds up the search process. DL-FOCL III integrates a local memory (tabu list) to avoid reconsidering choices that lead to dissatisfactory regions. To this end, at each search step, DL-FOCL III initializes an empty tabu list which is progressively populated with poor refinements. Once a refinement with an acceptable classification score is found, the tabu list is reinitialized and the search process continues, starting from the newly validated refinement. In this way, DL-FOCL III is able to prune the search space by exploring only the most promising regions, which often leads to high-quality solutions within a few exploration steps.

While on average DL-FOCL I outperforms DL-FOIL in terms of predictive accuracy, its overall runtime leaves much to be desired. The runtime of DL-FOCL I is n-times (n is the number of hill climbing repetitions) higher than that of DL-FOIL. DL-FOCL II and DL-FOCL III are faster than DL-FOIL but often underperform DL-FOIL in certain situations. For instance, the lookahead strategy in DL-FOCL II often leads to overfitting. Our proposed approaches for class expression learning do not employ the lookahead strategy nor a tabu list. However, they can be set to compute multiple candidate solutions per learning problem in parallel, which usually leads to improved predictive performance.

3.2.3 OWL Class Expression Learner

OCEL (OWL Class Expression Learner) [120] (Section 5.1) is an algorithm for class expression learning which uses a top-down DL refinement operator to construct and traverse its search space. OCEL navigates its search space by relying on a scoring function which takes into account the number of uncovered/covered positive/negative examples as well as the horizontal expansion (i.e., maximum length of child refinements) of the current node. In its search process, OCEL removes redundant concepts, i.e., concepts that are equivalent to those previously explored. Therefore, a node in the search tree is represented as a tuple (C, n, b), where C is a concept, n the horizontal expansion of C, and b a Boolean variable indicating whether the node is redundant or not.

The scoring function of OCEL is defined as

$$uncov\_pos = |E^+ \setminus R(C)| \quad \# \textit{ uncovered positives} \tag{3.2}$$

$$cov\_neg = |R(C) \cap E^-| \quad \# \textit{ covered negatives} \tag{3.3}$$

$$accuracy(C) = 1 - \frac{uncov\_pos + cov\_neg}{|E|} \tag{3.4}$$

$$acc\_gain(N) = accuracy(C) - accuracy(C') \tag{3.5}$$

$$score(N) = accuracy(C) + \alpha \cdot acc\_gain(N) - \beta \cdot n, \tag{3.6}$$

where $E = E^+ \cup E^-$ is the set of examples, C' is the concept in a parent node of N, $R(C)$ is the set of instances of C, and α, β are positive constants. In fact, only nodes whose numbers of uncovered positives $uncov\_pos$ do not reach a certain amount are considered for further exploration. OCEL therefore utilizes a noise parameter ϵ to discard nodes fulfilling $uncov\_pos > \lfloor \epsilon \cdot |E| \rfloor$, where $\lfloor . \rfloor$ denotes the integer part function. The redundancy check function $checkRed$ also contributes to pruning the search space.

Algorithm 9 OCEL algorithm [120]

Input: $\mathcal{K}, E = E^+ \cup E^-, \rho$
Output: $\{N_1, N_2, \ldots, N_k\}$ (best nodes)
Parameters: $\epsilon \in [0, 1]$ (noise parameter)

1: $ST = \{(\top, 0, False)\}$
2: **while** ST does not contain a node with quality less than $\lfloor \epsilon \cdot |E| \rfloor$ **do**
3: Choose a node $N = (C, n, b)$ with highest score in ST
4: Add all nodes $N' = (C', n + 1, checkRed(ST, C'))$ to ST, where $C' \in \rho(C)$
5: Evaluate all child-nodes N'
6: **end while**
7: **return** Top-k nodes in ST

The complete learning algorithm of OCEL is presented in Algorithm 9. The algorithm shows that OCEL starts its search process with the top concept $\top$, as opposed to DL-FOIL which starts with the bottom concept. Another difference between OCEL and other learning systems is the horizontal expansion parameter, which restricts the maximum length of a node's children at each step of the search process. Our approach CLIP employs a similar technique by setting a global threshold on the length of future refinements. This is achieved by using deep neural networks which are trained to predict the radius within which a solution can be found.

3.2.4 $\mathcal{EL}$ Tree Learner

ELTL ($\mathcal{EL}$ Tree Learner) [120] (Section 5.2) is an algorithm for learning $\mathcal{EL}$ trees (i.e., $\mathcal{EL}$ concepts represented as trees) using an ideal refinement operator (see Section 4.2 of [120] for details). Because $\mathcal{EL}$ does not support disjunctions, the base ELTL algorithm has been extended to the *disjunctive* ELTL algorithm whose solutions are disjunctions of multiple base ELTL-like solutions (computed in the inner loop of Algorithm 10, lines 4–8). We consider the disjunctive ELTL algorithm in this thesis and simply call it ELTL.

Algorithm 10 ELTL algorithm [120]

Input: $\mathcal{K}, E = E^+ \cup E^-, \rho^{ideal}$
Output: C (an $\mathcal{EL}$ concept)
Parameters: $\epsilon \in [0, 1]$ (noise parameter), $secondsPerTree$ (maximum time to spend on a tree), $minTreeScore \in [0, 1]$ (minimum tree score)

1: Initialize $C = \bot$ (initial guess) and $bestTreeScore = 1$
2: **while** $bestTreeScore \geq minTreeScore$ **do**
3: $ST = \{\top, False\}$
4: **while** the runtime spent is less than $secondsPerTree$ and ST does not contain a solution **do**
5: Choose a childless node $N = (D, b)$ with highest score in ST
6: Add all nodes $N' = (C', checkRed(ST, C'))$ as children of N, where $C' \in \rho^{ideal}(D)$
7: Evaluate all non-redundant child-nodes N' (i.e., $checkRed(ST, C') = False$)
8: **end while**
9: Select C' from ST with highest score and assign this score to $bestTreeScore$
10: **if** $bestTreeScore \geq minTreeScore$ **then**
11: $C = C \sqcup C'$
12: $E^+ = E^+ \setminus R_{\mathcal{K}}(C')$ # $R_{\mathcal{K}}(\cdot)$ is an instance retriever in K
13: $E^- = E^- \setminus R_{\mathcal{K}}(C')$
14: **end if**
15: **end while**
16: **return** $simplify(C)$

ELTL can be considered a simplification of OCEL for two main reasons: 1.) a node in ELTL is represented as (C, b), i.e., the horizontal expansion parameter n is omitted, and 2.) the formula of the heuristic function ($score(\cdot)$ in Equation 3.6) remains the same, with the horizontal expansion n replaced by the length of the node. The construction of a solution by ELTL is described in Algorithm 10. ELTL starts the search process by initializing its candidate solution as the bottom concept $\bot$, and the best tree score ($bestTreeScore$) is set to 1. In the outer-most loop, it looks for a sequence of concepts $C_1, C_2, \ldots, C_k$ with $checkRed(ST, C_i) = False \; \forall \; i$, and the associated

remaining sets of examples $E_1, E_2, \ldots, E_k$ (after removing those covered by the C_i's) such that $E_k \subseteq E_{k-1} \subseteq \ldots \subseteq E_1$ and $score(C_{k-1}, False) \geq minTreeScore$ and $score(C_k, False) < minTreeScore$. In the inner-most loop (lines 4–8), ELTL computes one of the C_i's by iteratively applying the ideal refinement operator ρ^{ideal} on running nodes until timeout (i.e., the $secondsPerTree$ are elapsed) or until a child node is qualified as solution. The final solution is a *simplified* disjunction of concepts $\bot \sqcup C_1 \sqcup C_2 \sqcup \ldots \sqcup C_k$. The simplification function removes redundant terms (i.e., those whose removal does not change the interpretation of the concept) from the final expression, e.g., $\bot$ is always removed.

3.2.5 Class Expression Learner for Ontology Engineering

CELOE (Class Expression Learner for Ontology Engineering) [122] is an extension of OCEL tailored towards ontology engineering. Its main goal is to learn concept definitions which can be suggested to a knowledge engineer, e.g., `Capital` $\equiv$ `City` $\sqcap (\exists$ `isCapitalOf.Country`$)$. Because in ontology creation and maintenance it is unlikely that extremely long concepts are used, CELOE employs a stronger length penalty as compared to OCEL, and biases its search towards even shorter concepts. The scoring function (heuristic) of CELOE is defined as

$$acc(C,t) = \frac{1}{t+1}\left(t \cdot \frac{|R(A) \cap R(C)|}{|R(A)|} + \sqrt{\frac{|R(A) \cap R(C)|}{|R(C)|}}\right) \tag{3.7}$$

$$acc\_gain(C) = acc(C,t) - acc(C',t) \tag{3.8}$$

$$score(C) = acc(C,t) + \alpha \cdot acc\_gain(C) - \beta \cdot |C|, \tag{3.9}$$

where $R(\cdot)$ is the instance retrieval function, A is the class expression to learn (usually an atomic concept), C a candidate solution ($|C|$ denotes the length of C as defined in Definition 2.2), C' the concept in a parent node of C, and $t > 1, \alpha, \beta \geq 0$ hyperparameters which control the behavior of the algorithm. For example, β is used to penalize the algorithm whenever it generates long concepts as those are less readable for a knowledge engineer. With its improved scoring function, CELOE remains one of the most capable class expression learning algorithms in DL-Learner [27].

3.2.6 DRILL

DRILL [48] is a class expression learning algorithm that uses deep-Q-learning to prune the search space of refinement operator-based approaches. Instead of measuring the quality (heuristic score) of a concept based merely on the amount of covered/uncovered positive/negative examples, DRILL uses a deep-Q-network to estimate the total discounted future rewards (i.e., heuristic scores) of trajectories starting from the current concept. To achieve this goal, DRILL represents each node in CELOE's search tree as a reinforcement learning environment state s. The reward function of DRILL (for transitioning from a state s to another state s') is defined as

$$\mathbf{R}(s, s') = \begin{cases} maxreward & \text{if } F_1(s') = 1 \\ score(s') & \text{otherwise,} \end{cases} \tag{3.10}$$

where $maxreward$ is the maximum reward the RL agent in DRILL can obtain (e.g., 1) when it takes an action, $score(\cdot)$ is the scoring function of CELOE as defined in Equation 3.9, and F_1 is defined as

$$F_1(s) = \frac{|E^+ \cap R(s)|}{|E^+ \cap R(s)| + \frac{1}{2}(|E^- \cap R(s)| + |E^+ \setminus R(s)|)}. \tag{3.11}$$

The heuristic function of DRILL is defined as

$$\phi_{\text{DRILL}}([\mathbf{s}, \mathbf{s}', \mathbf{e}_+, \mathbf{e}_-]; \Theta) = f\Big(\text{vec}(f(\Psi([\mathbf{s}, \mathbf{s}', \mathbf{e}_+, \mathbf{e}_-]) * \omega)) \cdot \mathbf{W}\Big) \cdot \mathbf{H}, \tag{3.12}$$

where $f(\cdot)$ is the rectified linear unit activation function, $*$ denotes a convolution operation, and $\mathbf{s} = \mathbf{E}(R(s)) \in \mathbb{R}^{|R(s)| \times d}$ and $\mathbf{s}' = \mathbf{E}(R(s')) \in \mathbb{R}^{|R(s')| \times d}$ are the embedding matrices for the states s and s', respectively. $\Theta = [\Psi, \omega, \mathbf{W}, \mathbf{H}]$ are trainable model parameters, $\mathbf{e}_+ \in \mathbb{R}^{|E^+| \times d}$, and $\mathbf{e}_- \in \mathbb{R}^{|E^-| \times d}$ are the embeddings for positive and negative examples, respectively.

To accelerate class expression learning, DRILL is often pretrained in an unsupervised manner. This involves generating class expressions of various lengths which serve as learning problems from which DRILL learns the search space traversal by attempting to compute their respective solutions (i.e., reach goal states). Algorithm 11 presents the training procedure of DRILL. The algorithm starts with the top concept $\top$ as the initial state. It then applies the refinement operator ρ to obtain a set of possible next states (concepts with a non-empty instance set) from which it either randomly selects one state s' if $\epsilon > 0.1$ or selects the best heuristic-based state otherwise. Next,

the reward of s' is computed and the history $[\mathbf{s}, \mathbf{s}', \mathbf{e}_+, \mathbf{e}_-, reward]$ is appended to the experience replay dataset $\mathcal{D}$. This process is repeated m times by considering the most recent states every time, with the model parameters being updated every fifth iteration using samples from $\mathcal{D}$. Post-training, DRILL converges to goal states (i.e., solutions to learning problems) at least 2.7× faster than state-of-the-art algorithms such as CELOE.

Algorithm 11 DRILL with deep Q-learning training procedure [48]

Require: E^+, E^-, ρ, $\mathbf{R}$, R, $\mathbf{E}$, Θ, M, T
for m := 1, M **do**
  $\mathbf{s}, s ::= \mathbf{E}(R(\top)), \top$
  for $t := 1, T$ **do**
    $\mathbf{z} := \{s' \in \rho(s) | \; |(R(s'))| > 0\}$
    if $\epsilon > .1$ **then**
      Select random $\mathbf{s}' \in \mathbf{z}$
    else
      Select $\mathbf{s}' := \arg\max_{\mathbf{s}' \in \mathbf{z}} \phi_{\text{DRILL}}([\mathbf{s}, \mathbf{s}', \mathbf{e}_+, \mathbf{e}_-]); \Theta)$
    end if
    Compute reward $reward := \mathbf{R}(\mathbf{s}, \mathbf{s}')$
    Append $[\mathbf{s}, \mathbf{s}', \mathbf{e}_+, \mathbf{e}_-, reward]$ to $\mathcal{D}$
    Set $s, \mathbf{s} ::= s', \mathbf{s}'$
  end for
  Reduce ϵ with a constant
  if m % 5 = 0 **then**
    Sample random minibatches from $\mathcal{D}$
    Compute loss of minibatches w.r.t. Θ
    Update parameters Θ accordingly
  end if
end for

While DRILL shows significant improvements in runtime over the previous state of the art, it still requires to be pretrained on vast amounts of data. Moreover, the use of a refinement operator during the pretraining and inference phases further limits the scalability of DRILL. The neural synthesis-based approaches we propose in this thesis do not use a refinement operator at inference time, and are several orders of magnitude faster than DRILL.

All approaches described above are refinement operator-based. We will now introduce two other state-of-the-art approaches which do not use a refinement operator, but employ different techniques to construct and traverse their search space.

3.2.7 Efficient Concept Induction from Instances

ECII (Efficient Concept Induction from Instances) [185] is a search-based algorithm that uses disjunctions, conjunctions and negations of pre-selected complex class expressions to construct and traverse its search space. More specifically, given a background knowledge $\mathcal{K}$, and sets of positive and negative examples E^+ and E^-, ECII first computes a set of statements $\mathcal{A}_a = \{A(a), r(a,b), B(b)\}$ occurring in $\mathcal{K}$ for every individual $a \in E^+ \cup E^-$. It then chooses one of the following complex expressions as candidate solution $\mathcal{S}$ and evaluates it:

$$(1) \quad A \sqcap \left(\prod_{i=1}^{l} \exists\, r_i.C_i \right) \tag{3.13}$$

$$(2) \quad A \sqcap \left(\prod_{i=1}^{l} \exists\, r_i. \left(\bigsqcup_{j=1}^{m} B_{j_i} \sqcap \neg(D_1 \sqcup D_2 \sqcup \ldots \sqcup D_{j_{i_k}}) \right) \right), \tag{3.14}$$

where C_i are candidate class expressions constructed with $\sqcup, \sqcap$ or $\neg$, and $A, B_{j_i}, D_{j_{i_k}}$ are atomic concepts. Each r_i is a role occurring in the statements $\mathcal{A}_a$ of at least one example $a \in E^+ \cup E^-$. If the evaluation of the candidate solution yields unsatisfactory results (e.g., poor coverage of positive examples), ECII then chooses another candidate expression $\mathcal{S}$ but with the same form as (1) or (2). This process is repeated until the maximum number of iterations is reached or until a candidate solution perfectly covers all positive examples while ruling out negative ones, i.e., $\mathcal{K} \models S(a)\ \forall\ a \in E^+$ and $\mathcal{K} \not\models S(a)\ \forall\ a \in E^-$.

Because ECII does not employ a refinement operator, and invokes a reasoner only once for each learning problem, it is often faster than CELOE, but tends to compute lower quality solutions on average. However, it still remains relatively slow when compared to our neural synthesis-based approaches, as we will see in Chapters 5 and 6.

3.2.8 Evolutionary Concept Learning

EvoLearner (Evolutionary Concept Learning) [82] is an approach for class expression learning which employs evolutionary algorithms. Given a background knowledge base $\mathcal{K}$ and a learning problem defined by positive examples E^+ and negative examples E^-, EvoLearner initializes its population (candidate solutions) by random walks on $\mathcal{K}$, which is viewed as a graph beforehand. It then subsequently applies mutations, cross-over operations, and a heuristic function to construct the next generations of the initial population until a solution is found.

Specifically, EvoLearner starts from a positive example $e^+ \in E^+$ and performs a random walk of length k, yielding a sequence of roles $r_i,\ 1 \leq i \leq m$ which are completed with quantifiers (e.g., $\exists,\ \forall$) and types ($\top,\ \bot$, or atomic concepts). The result of a random walk is a concept $\exists\ r_1.(\exists\ r_2.(\exists \ldots \forall\ r_m.C_m))$, which is represented as a tree. This process is repeated until the maximum population size n is reached. Then, a series of n tournaments starts, where the best tree is chosen every time among a randomly and uniformly constructed sub-population of size k. Simultaneously, crossover and mutation operations are invoked to swap nodes in two sub-trees, and randomly replace an atomic concept with another relevant atomic concept (e.g., a second type for the considered example if any), respectively. At the conclusion of tournaments, the first generation of the population is obtained. This process is then repeated for future generations until a solution to the considered learning problem is found, or until the maximum number of generations is reached.

EvoLearner outperforms many state-of-the-art approaches, including CELOE, SPaCEL [206], and Aleph [195] in predictive performance, and runtime. Our approach CLIP using concept length predictors is comparable to EvoLearner, as they both lag behind our synthesis-based approaches NCES, NCES2, and ROCES in terms of runtime.

In this chapter, we have covered several works tackling various problems in inductive logic programming, including the state of the art for class expression learning in description logics. We will now present our scientific contributions towards scalable class expression learning on large knowledge bases.

4 Concept Learner with Integrated Length Prediction

Preamble. This chapter is based on Kouagou et al. [107] and answers our first research question (see Section 1.2.1).

4.1 Methodology

In this section, we introduce CLIP, our first approach that leverages concept length predictors to accelerate class expression learning (CEL). We start by defining the problem we aim to solve before describing our proposed approach. Afterwards, we present and discuss experimental results where CLIP is compared against state-of-the-art approaches.

4.1.1 Learning Problem

Below, we recall the definition of the classical learning problem for CEL in description logics. This definition is based on previous works [121] with slight modifications in notations for more clarity.

Definition 4.1 (Classical Learning Problem ($\mathcal{CLP}$))**.** *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 $\mathcal{K}'_C = \mathcal{K} \cup \{T \equiv C\}$, we have that $\mathcal{K}'_C \models C(E^+)$ and $\mathcal{K}'_C \not\models C(E^-)$.*

We write $\mathcal{K}'_C \models C(E)$ (respectively, $\mathcal{K}'_C \not\models C(E)$) to express that $\forall\, e \in E,\ \mathcal{K}'_C \models C(e)$ (respectively, $\forall\, e \in E,\ \mathcal{K}'_C \not\models C(e)$). Here, $\mathcal{K}'_C \models C(e)$ means *e is an instance of C according to $\mathcal{K}'_C$*. Note that in some literature, e.g., [121], an additional constraint is often imposed on the solution C, i.e., requiring that T does not occur in C. In the scope of this thesis, we ignore this constraint as we do not explicitly provide the

target concept T to a learning approach. Hence, an approach for CEL is tasked to find a solution with solely the provided positive and negative examples.

Previous works for CEL, e.g., [122, 185] mainly focused on developing search-based approaches with hard-coded heuristics to solve the classical learning problem introduced in Definition 4.1. The primary drawbacks of these approaches are long runtimes especially on large datasets where they often fail to find a reasonable solution within a set timeout. Our approach CLIP, described below, improves the runtime of search-based approaches by employing concept length predictors rooted in deep neural networks.

4.1.2 Overview of the Approach

The intuition behind CLIP is that *if we have a reliable concept length predictor, then our concept learner only needs to test concepts of length up to the predicted length*. Figure 4.1 illustrates CLIP's search strategy. Refinements that exceed the predicted length are ignored during search. In the figure, the concept $\texttt{Person} \sqcap (\forall\, \texttt{attendsSome}.(\texttt{Workshop} \sqcup \texttt{Conference}))$ is of length 7 and is therefore neither tested nor added to the search tree.

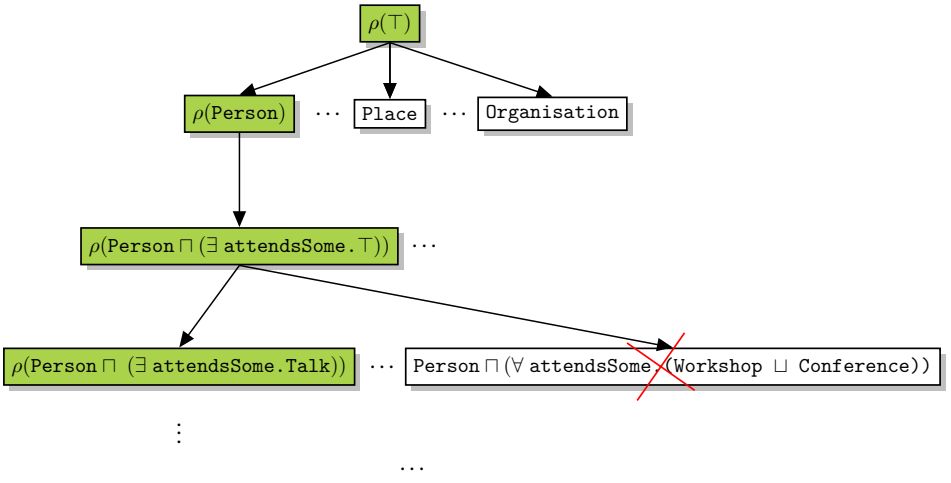


Figure 4.1: Search tree of CLIP when the predicted length is 5. After each refinement, CLIP discards all concepts whose lengths are larger than the set threshold.

Remark 4.2. *For concept length prediction during concept learning, we sample n_1 positive examples and n_2 negative examples from the considered learning problem such that $n_1 + n_2 = n$, as described in Section 4.1.3 (4).*

4.1.3 Concept Length Prediction

In this section, we address the following learning problem: Given a knowledge base $\mathcal{K}$, a set of positive examples E^+ and negative examples E^-, predict the length of the shortest concept C that is a solution to the learning problem defined by $\mathcal{K}$, E^+, and E^- according to Definition 4.1. To achieve this goal, we devise a generator that creates training data for our prediction algorithm based solely on $\mathcal{K}$ and a user-given number of learning problems to use at training time.

Training Data for Length Prediction

Let $\mathcal{K}$ be a knowledge base whose sets of individuals, atomic concepts, and roles are denoted N_I, N_C, and N_R, respectively.

Data Generation. Given a knowledge base, the construction of training data (concepts with their positive and negative examples) is carried out as follows:

1. Generate concepts of various lengths using the length-based refinement operator described in Algorithm 12 and 13. In this process, short concepts are preferred over long concepts, i.e., when two concepts have the same set of instances, the longest concept is discarded.
2. Compute the sets $\mathcal{E}_C^+$ and $\mathcal{E}_C^-$ for each generated concept C[1].
3. Define a hyperparameter $n \in [1, |N_I|]$ that represents the total number of positive and negative examples a length predictor can take as input.
4. Sample positive and negative examples as follows:
   - If $|\mathcal{E}_C^+| \geq \frac{n}{2}$ and $|\mathcal{E}_C^-| \geq \frac{n}{2}$, then we randomly sample $\frac{n}{2}$ individuals from each of the two sets $\mathcal{E}_C^+$ and $\mathcal{E}_C^-$.
   - Otherwise, we take all individuals in the minority set and sample the remaining number of individuals from the other set.

Training Data Features. A knowledge graph is commonly defined as $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, where $\mathcal{E}$ is a set of entities and $\mathcal{R}$ is a set of relation types. We convert a given knowledge base $\mathcal{K}$ into a knowledge graph by converting $ABox$ statements of the form $r(a, b)$ into (a, r, b). Statements of the form $C(a)$ are converted into $(a, \texttt{rdf:type}, C)$. In our experimental data, the $TBox$es contained only subsumptions $C \sqsubseteq D$ between atomic concepts C and D, which were converted into

[1] $\mathcal{E}_C^+$ denotes the set of instances of C, and $\mathcal{E}_C^- = N_I \setminus \mathcal{E}_C^+$

triples $(C, \texttt{rdfs:subClassOf}, D)$. Hence, in our experiments, $\mathcal{E} \subseteq N_C \cup N_I$ and $\mathcal{R} = N_R \cup \{\texttt{rdf:type}, \texttt{rdfs:subClassOf}\}$. The resulting knowledge graph $\mathcal{G}$ is then embedded into a continuous vector space to serve for the prediction of concept lengths. We use the embedding model ConEx [45] for embedding computation because of its computational efficiency and high predictive performance. On the vector representation of entities, we create an extra dimension at the end of the entries, where we insert $+1$ for positive examples and -1 for negative examples. Formally, we define an injective function f_C for each target concept C

$$
\begin{aligned}
& f_C : \mathbb{R}^d \longrightarrow \mathbb{R}^{d+1} \\
& \mathbf{x} = (x_1, \ldots, x_d) \longmapsto \begin{cases} (x_1, \ldots, x_d, \quad 1) & \text{if } ent(\mathbf{x}) \in \mathcal{E}_C^+, \\ (x_1, \ldots, x_d, -1) & \text{otherwise,} \end{cases}
\end{aligned} \tag{4.1}
$$

where d is the dimension of the embedding space, and $ent(\mathbf{x})$ is the entity whose embedding is $\mathbf{x}$. The extra dimension aims to facilitate the distinction between positive and negative examples for each class expression. Thus, a data point in the training, validation, and test datasets is a tuple $(M_C, length(C))$, where M_C is a matrix of shape $n \times (d+1)$ constructed by concatenating the embeddings of positive examples followed by those of negative examples. Formally, assume n_1 and n_2 are the numbers of positive and negative examples for C, respectively. Further, assume that the embedding vectors of positive examples are $x^{(i)},\ i = 1 \ldots, n_1$ and those of negative examples are $x^{(i)},\ i = n_1 + 1, \ldots, n_1 + n_2 = n$. Then, the $i-th$ row of M_C is given by

$$
M_C[i,:] = \begin{cases} (x_1^{(i)}, \ldots, x_d^{(i)}, \quad 1) & \text{if } 1 \leq i \leq n_1, \\ (x_1^{(i)}, \ldots, x_d^{(i)}, -1) & \text{if } n_1 + 1 \leq i \leq n_1 + n_2 = n. \end{cases} \tag{4.2}
$$

We view the prediction of concept lengths as a classification problem with classes $1, 2, \ldots, L$, where L is the length of the longest concept in the training dataset. As shown in Table 4.1, the concept length distribution can be imbalanced. To prevent concept length predictors from overfitting on the majority classes, we used the weighted cross-entropy loss

$$
\mathcal{L}_w(\bar{y}, y) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{L} w_k \mathbb{1}(k, y^i) \log(\bar{y}_k^i), \tag{4.3}
$$

where N is the batch size, $\bar{y}$ is the batch matrix of predicted probabilities (or scores), y is the batch vector of targets, w is a weight vector, and $\mathbb{1}$ is an indicator function which returns 1 if its inputs are identical, and 0 otherwise. The weight vector w_k

Algorithm 12 Function REFINEHELPER

Input: Knowledge base $\mathcal{K}$, atomic concept A
Parameters: k, default 5
Output: Initial refinements of A

1: $\mathcal{S} \leftarrow \text{SUBCONCEPTS}_{\mathcal{K}}(A)$ *# Subconcepts of A in $\mathcal{K}$*
2: $\neg\mathcal{S} \leftarrow \{\neg C | C \in \mathcal{S}\}$
3: $Restrict \leftarrow \{\}$
4: **if** $|\mathcal{S}| < k$ **then**
5: $\quad\mathcal{F} \leftarrow \{\top, \bot, A\}$ # Set of fillers
6: **else**
7: $\quad\mathcal{F} \leftarrow \{\top, \bot, A\} \cup \text{RANDSAMPLE}(\mathcal{S}, n = k) \cup \text{RANDSAMPLE}(\neg\mathcal{S}, n = k)$
8: **end if**
9: **for** C in $\mathcal{F}$ **do**
10: $\quad$**for** r in N_R **do**
11: $\quad\quad Restrict \leftarrow Restrict \cup \{\exists\ r.C\}$
12: $\quad\quad Restrict \leftarrow Restrict \cup \{\forall\ r.C\}$
13: $\quad$**end for**
14: **end for**
15: **return** $\mathcal{S} \cup \neg\mathcal{S} \cup Restrict$

is defined as: $w_k = 1/\sqrt{[k]}$, where $[k]$ is the number of concepts of length k in the training dataset.

4.1.4 Express Refinement

We implemented the intuition behind CLIP by extending CELOE's refinement operator. Our refinement operator differs from CELOE's in how it refines atomic concepts (see Algorithms 12 and 13). For example, it considers all refinements $A' \sqsubset A$ of an atomic concept A whereas CELOE's refinement operator only considers $A' \sqsubset A$ such that there is no A'' with $A' \sqsubset A'' \sqsubset A$. Omitting this expensive check allows more concepts to be tested in the same amount of time. In the following, we describe our method for refining atomic concepts and refer the reader to [121, 122] for details on CELOE. In Algorithms 12 and 13, the hyperparameters max_length, k, and $sample\_frac$ control the expressivity of the refinement operator: max_length specifies how long the refinements can become (Algorithm 13, lines 8, 11, 13); k controls the number of fillers sampled without replacement for universal and existential role restrictions (Algorithm 12, line 7); $sample\_frac \in (0, 1]$ specifies the fraction of initial refinements to be sampled for further exploration (Algorithm 13, lines 1–3). Given a knowledge base $\mathcal{K}$, the refinement of an atomic concept A is carried out as follows: (1) obtain the subconcepts $\mathcal{S}$ of A in $\mathcal{K}$; (2) compute the negations $\neg\mathcal{S}$ of all subconcepts of A; (3) construct existential and universal role restrictions $Restrict$ where the fillers $\mathcal{F}$ are in the set made of $\top, \bot, A$, and samples from $\mathcal{S}$ and $\neg\mathcal{S}$; (4) compute the union of $\mathcal{S}$, $\neg\mathcal{S}$, and $Restrict$ as output of Algorithm 12, and (5) use Algorithm 13 to compute

Algorithm 13 Function REFINEATOMICCONCEPT

Input: Knowledge base $\mathcal{K}$, atomic concept A
Parameters: $sample\_frac \in (0, 1]$, max_length
Output: $\{C_1, \ldots, C_n\}$ which are refinements of A

$Init\_Ref$ = REFINEHELPER($\mathcal{K}, A$)
$m \leftarrow \lfloor sample\_frac \times |Init\_Ref| \rfloor$ *# $\lfloor \cdot \rfloor$ is the integer part function*
$Init\_Ref \leftarrow$ RANDSAMPLE($Init\_Ref$, $n = m$) *# Sample without replacement*
$\mathcal{S} \leftarrow$ SUBCONCEPTS$_{\mathcal{K}}(A)$ *# Subconcepts of A in $\mathcal{K}$*
$Refinements \leftarrow \mathcal{S}$ *# All subconcepts are of length 1*
for S_1 in $\mathcal{S}$ **do**
  for S_2 in $Init\_Ref$ **do**
    if $S_1 \neq S_2$ and $length(S_1 \sqcap S_2) \leq max\_length$ **then**
      $Refinements \leftarrow Refinements \cup \{S_1 \sqcap S_2\}$
    end if
    if $S_1 \neq S_2$ and $S_2 \in \mathcal{S}$ and $length(S_1 \sqcup S_2) \leq max\_length$ **then**
      $Refinements \leftarrow Refinements \cup \{S_1 \sqcup S_2\}$
    else if $S_1 \neq S_2$ and $length((S_1 \sqcup S_2) \sqcap A) \leq max\_length$ **then**
      $Refinements \leftarrow Refinements \cup \{(S_1 \sqcup S_2) \sqcap A\}$
    end if
  end for
end for
return $Refinements$

and return final refinements as intersections and unions of sub-concepts $\mathcal{S}$ and the output of Algorithm 12, with the generated refinements having length at most max_length. The refinement operator is designed to yield numerous meaningful downward refinements from a single atomic concept.

4.2 Experiments

In this section, we design experiments to evaluate our proposed approach. We start with experimental settings, where we describe the datasets and evaluation metrics used as well as hyperparameter configurations. We then present and discuss our experimental results.

4.2.1 Experimental Setup

We conducted two sets of experiments. In the first set of experiments, we assess the ability of neural networks to effectively predict the length concepts in the description logic $\mathcal{ALC}$ based on positive and negative examples. In the second set of experiments, we evaluate the effect of concept length prediction on class expression learning in terms of runtime and quality of the computed solutions.

Datasets

We ran experiments on four benchmark datasets: Carcinogenesis [224], Mutagenesis [224], Semantic Bible[2], and Vicodi [152]. Carcinogenesis and Mutagenesis are knowledge bases about chemical compounds and how they relate to each other. The Semantic Bible knowledge base describes each named object or thing in the New Testament. The Vicodi knowledge base was developed as part of a funded project and describes the European history.

Training Data Generation

We construct training data for concept length prediction as described in Section 4.1.3. The generated data is then split into three subsets: training, validation, and test datasets, the complete statistics of which are reported in Table 4.1. Though the maximum length for the generation of concepts is fixed to 15, many long concepts were equivalent to shorter ones. Hence, they were removed from the generated data, and the longest remaining are of length 11 (see Table 4.1).

Table 4.1: Statistics of the generated data on each benchmark dataset

Length	Carcinogenesis			Mutagenesis			Semantic Bible			Vicodi		
	Train	Val.	Test	Train	Val.	Test	Train	Val.	Test	Train	Val.	Test
3	3,647	405	1,013	1,038	115	288	487	54	135	3,952	439	1,098
5	782	87	217	1,156	129	321	546	61	152	2,498	278	694
6	0	0	0	0	0	0	162	18	45	335	37	93
7	1,143	127	318	1,310	146	364	104	12	29	3,597	400	999
8	0	0	0	0	0	0	0	0	0	747	83	207
9	0	0	0	0	0	0	73	8	21	0	0	0
11	0	0	0	0	0	0	41	5	11	0	0	0

Concept Length Predictors

We consider four neural network architectures: Long Short-Term Memory (LSTM) [85], Gated Recurrent Unit (GRU) [36], Multi-Layer Perceptron (MLP), and Convolutional Neural Network (CNN), see Chapter 2 for more details on these architectures. Recurrent neural networks (LSTM, GRU) take as input a sequence of embeddings of positive and negative examples (all positive examples followed by all negatives). The CNN model takes the same input as recurrent networks but it views the input as a gray image (i.e., with a single channel). The MLP model takes as input the average embeddings of positive and negative examples. We also consider a random model which predicts the length of a concept based on the distribution of lengths. It assigns high probabilities to the most represented lengths in the a given dataset. The motivation behind the random model is to see how it compares to neural network-based length predictors.

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

Evaluation Metrics

We considered four metrics in our experiments. The first metric is the *accuracy* which is used during training to compare predicted concept lengths to actual ones in the training, validation, and test sets. The second metric is the *macro* F_1 *score* which also compares the predicted concept lengths to the ground truth, but is more sensitive to errors made on minority classes. The third and fourth metrics (Acc and F_1) are used to measure the quality of solutions to learning problems in the second set of experiments, i.e., class expression learning. Acc and F_1 are defined as

$$Acc = \frac{|\mathcal{E}_C^+ \cap E^+| + |\mathcal{E}_C^- \cap E^-|}{|E^+| + |E^-|}, \tag{4.4}$$

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}, \tag{4.5}$$

$$Precision = \frac{|\mathcal{E}_C^+ \cap E^+|}{|\mathcal{E}_C^+ \cap E^+| + |\mathcal{E}_C^+ \cap E^-|}, \tag{4.6}$$

$$Recall = \frac{|\mathcal{E}_C^+ \cap E^+|}{|\mathcal{E}_C^+ \cap E^+| + |E^+ \setminus \mathcal{E}_C^+|}, \tag{4.7}$$

where $\mathcal{E}_C^+$ is the set of instances of the candidate solution C, $\mathcal{E}_C^- = N_I \setminus \mathcal{E}_C^+$, and E^+ and E^- the sets of positive and negative examples of a learning problem, respectively.

Hyperparameter Optimization and Hardware

In our preliminary experiments, we used a random hyperparameter search method [20] to select the best values (as summarized in Table 4.3). This method also helped us find out that two recurrent layers (for LSTM and GRU) followed by two linear layers, batch normalization, and dropout layers is sufficient to achieve high predictive performance with low computational costs. Similarly, we chose two convolution, linear, and dropout layers with one batch normalization for the CNN omodel. The MLP model consists of four linear layers, one batch normalization and dropout layers. The rectified linear unit (ReLU) is used in the intermediate layers of all models, whereas the *Sigmoid* function is used in the output layers. We ran experiments in a 10-fold cross-validation setting with ten repetitions. Table 4.3 gives an overview of the hyperparameter settings on each of the four knowledge bases considered. The number of epochs is set based on the training speed and the performance achieved on validation datasets. For example, on the Carcinogenesis knowledge base, most length predictors are able to reach 90% accuracy with just 50 epochs, which suggests that more epochs would probably lead to overfitting. Adam optimizer [104] is used to train the length predictors. We varied the number of examples n between 200 and 1000, and the embedding dimension d from 10 to 100, but we finally chose $n = \min(1000, \frac{|N_I|}{2})$ and $d = 40$ for their high predictive accuracy and low computation costs.

We trained all concept length predictors on a single 11GB memory NVIDIA K80 GPU with 4 Intel Xeon E5-2670 CPUs at 2.60GHz, and 24GB RAM. During concept learning with

Table 4.2: Model size (|*Parameters*|) and training time (*Time*)

	Carcinogenesis		Mutagenesis	
	|*Parameters*|	*Time (sec.)*	|*Parameters*|	*Time (sec.)*
LSTM	160,208	188.42	160,208	228.13
GRU	125,708	191.16	125,708	228.68
CNN	838,968	16.77	838,248	44.74
MLP	61,681	10.04	61,681	14.29
	Semantic Bible		Vicodi	
	|*Parameters*|	*Time (sec.)*	|*Parameters*|	*Time (sec.)*
LSTM	161,012	196.20	160,409	362.28
GRU	125,512	197.86	125,909	367.55
CNN	96,684	18.43	839,377	71.95
MLP	61,933	9.56	61,744	24.61

Table 4.3: Hyperparameter setting. Lr is the learning rate during training, d the number of embedding dimensions (a.k.a. embedding dimension), N the batch size during training, and n the total number of examples (both positive and negative ones) used to predict concept lengths.

	Epochs	Lr	d	N	n
Carcinogenesis	50	0.003	40	512	1,000
Mutagenesis	100	0.003	40	512	1,000
Semantic Bible	200	0.003	40	256	362
Vicodi	50	0.003	40	512	1,000

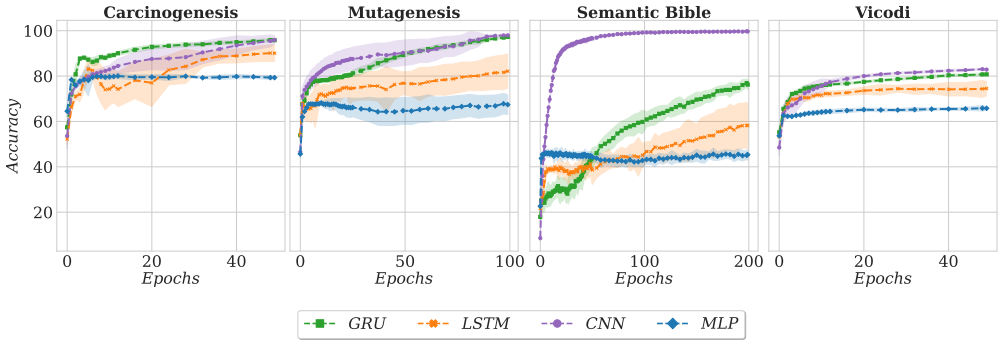
CELOE, OCEL, ELTL, and CLIP, we used a CPU only server with 8-core Intel Xeon E5-2695 at 2.30GHz, and 16GB RAM to ensure a fair comparison, since baseline approaches do not support GPUs.

4.2.2 Results and Discussion

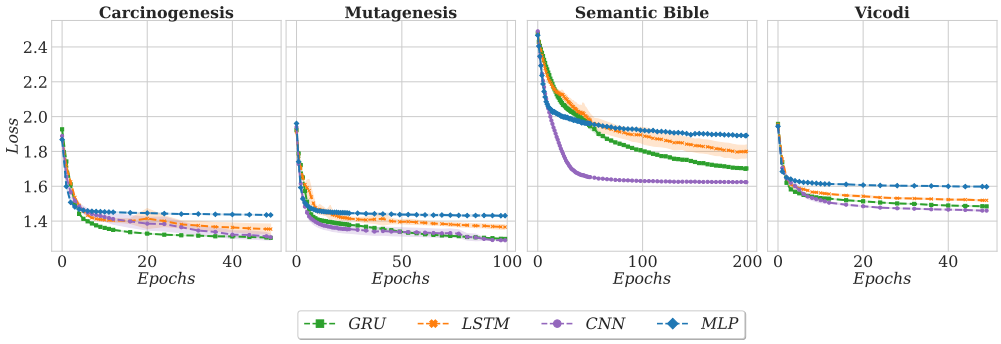
Concept Length Prediction

Table 4.2 shows the number of parameters and training time of LSTM, GRU, CNN, and MLP architectures on each of the datasets. From the table, we can observe that our concept length predictors can be trained in less than an hour and be used for efficient concept learning on corresponding knowledge bases.

In Figures 4.2 and 4.3, we show training curves for each model on all datasets. We can observe a decreasing loss on all knowledge bases (see Figures 4.2b and 4.3b), which suggest that models effectively learn from the provided data. Moreover, the Gated Recurrent Unit (GRU) model outperforms the other models on all datasets, see Figure 4.3a and Table 4.4. Inputs to the MLP model are average embeddings of positive and negative examples of given concepts. This may have caused loss of information in the inputs. As shown in Figure 4.2a, MLP curves tend to saturate in the early stages of training. We also assessed the element-wise multiplication of the embeddings and obtained similar results. However, as reflected in Table



(a) Cross-validation accuracy on training sets



(b) Cross-validation loss on training sets

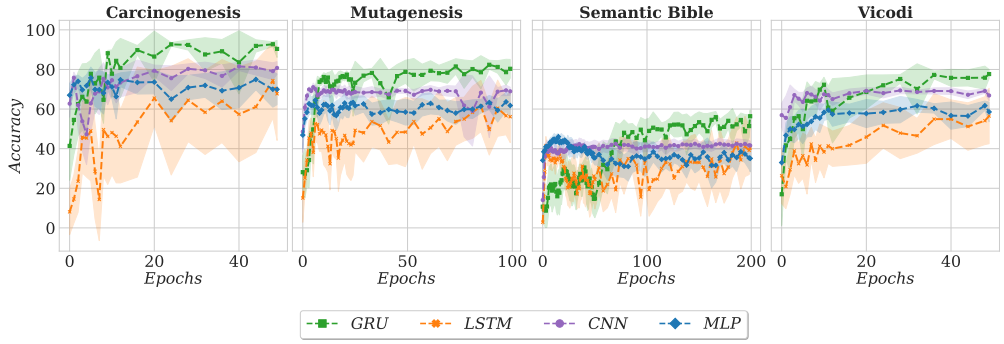
Figure 4.2: Training curves. Colored areas represent standard deviations across 10 runs in a 10-fold cross-validation setting.

4.4, all our proposed architectures outperform a random model that knows the distribution of the lengths of concepts in the training dataset[3]. A modified version of MLP where the embedding of each example is processed independently with a linear map before averaging is performed (no interaction) also achieved a poor predictive performance. This suggests that the full interaction between examples, captured by recurrent units, is key to effectively predicting concept lengths.

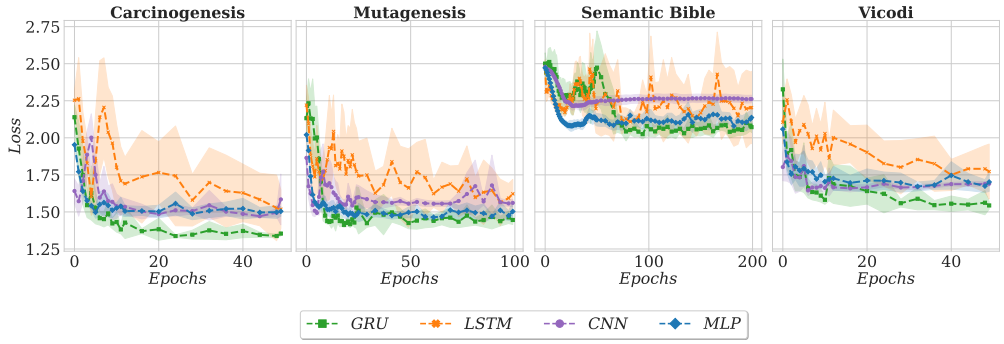
Table 4.4 compares our chosen neural network architectures and a random model on the Carcinogenesis, Vicodi, Mutagenesis, and Semantic Bible knowledge bases. From the table, it appears that recurrent neural network models (GRU, LSTM) outperform the other two models (CNN and MLP) on three out of four datasets, with the only exception that the LSTM model slightly dropped in performance on Vicodi compared to CNN.

While the CNN model tends to overfit on all knowledge bases, the MLP model is unable to extract meaningful information from the average embeddings. On the Semantic Bible knowledge base, which appears to be the smallest dataset, all our proposed networks performed less well than expected. This suggests that our learning approach is more suitable for large knowledge bases. Nonetheless, all our proposed models are clearly better than a

[3]The random model predicts the length of a concept by sampling from a distribution that assigns high probability values to the most represented concept lengths in the training set.



(a) Cross-validation accuracy on validation sets



(b) Cross-validation loss on validation sets

Figure 4.3: Validation curves. Colored areas represent standard deviations across 10 runs in a 10-fold cross-validation setting.

distribution-aware random model with a minimum performance difference (macro F_1 score) on average between 21.25% (MLP) and 41% (GRU).

Class Expression Learning

The maximum runtime is set to 2 minutes per learning problem.[4] For all knowledge bases, we generate 100 random learning problems by (1) creating random $\mathcal{ALC}$ concepts C of maximum length 15, (2) computing the sets of instances $\mathcal{E}_C^+$ and $\mathcal{E}_C^-$, (3) providing $\mathcal{E}_C^+$ and $\mathcal{E}_C^-$ to each of the approaches, and (4) measuring the accuracy, the F-measure, the runtime, and the length of the best solution generated within the set timeout. We ran all approaches on the same hardware (see Section 4.2.1). CLIP is configured to use our best concept length predictor (GRU). Note that a predictor is trained for each dataset (see Table 4.1). Also note that we add the ELTL algorithm—a concept learner for the description logic $\mathcal{EL}$—to investigate whether our randomly generated concepts is equivalent to concepts in a simpler description logic.

[4]The implementations of OCEL and ELTL in the DL-Learner framework, which we used for our experiments, fail to consider the set threshold accurately. Hence, Table 4.5 contains values larger than 2 minutes for these two algorithms.

Table 4.4: Effectiveness of concept length prediction. RM is a random model that makes predictions according to the length distribution in the training dataset, and F_1 is the macro F-measure.

	Carcinogenesis					Mutagenesis				
	LSTM	GRU	CNN	MLP	RM	LSTM	GRU	CNN	MLP	RM
Train. acc.	0.89	0.96	0.97	0.80	0.48	0.83	0.97	0.98	0.68	0.33
Val. acc.	0.76	0.93	0.82	0.77	0.48	0.70	0.82	0.71	0.65	0.35
Test acc.	0.92	**0.95**	0.84	0.80	0.49	0.78	**0.85**	0.70	0.68	0.33
Test F_1	0.88	**0.92**	0.71	0.59	0.33	0.76	**0.85**	0.70	0.67	0.32
	Semantic Bible					Vicodi				
	LSTM	GRU	CNN	MLP	RM	LSTM	GRU	CNN	MLP	RM
Train. acc.	0.85	0.93	0.99	0.68	0.33	0.73	0.81	0.83	0.66	0.28
Val. acc.	0.49	0.58	0.44	0.46	0.26	0.55	0.77	0.70	0.64	0.30
Test acc.	0.52	**0.53**	0.37	0.40	0.25	0.66	**0.80**	0.69	0.66	0.29
Test F_1	0.27	**0.38**	0.20	0.22	0.16	0.45	**0.50**	0.45	0.38	0.20

Table 4.5 presents a comparison of the results achieved by CLIP, CELOE, OCEL, and ELTL; results are formatted *mean* $\pm$ *standard deviation*. Note that the table does not contain DL-FOIL because it could not solve the learning problems that we considered. For instance, the first learning problem on the Semantic Bible knowledge base targets $\texttt{SonOfGod} \sqcup (\exists\ \texttt{locationOf.StateOrProvince})$. Here, DL-FOIL freezes on the refinement of `GeographicLocation` with over 5×10^3 unsuccessful trials. Similar observations were made on other datasets. We also tried running DL-FOCL, but it was not possible using the documentation provided. Our results suggest that CLIP outperforms the other three algorithms in F_1 and in runtime on most datasets. The ELTL algorithm appears to be faster than CELOE and OCEL but slower than CLIP. However, its short runtime stems from the fact that it computes its solutions in the description logic $\mathcal{EL}$, and returns the top concept most of the time; this is reflected in the average length of the solutions it computes. Moreover, some of our learning problems can only be solved in $\mathcal{ALC}$ or a more expressive description logic, which explains the poor performance of ELTL in terms of F-measure across all datasets.

We used a Wilcoxon Rank test to check whether the difference in performance between CLIP, CELOE, and OCEL is significant. Significant differences are marked with an asterisk ($*$). The null hypothesis for our test was that "the two distributions that we compare are the same". The significance level is $\alpha = 0.05$. The performance differences in F_1 between CLIP and the other algorithms are significant on 3 out of 4 datasets.[5] With respect to runtimes, we significantly outperform all other algorithms on all datasets. Large time differences correspond to scenarios where CLIP detects short solution concepts while other algorithms explore longer concepts. Low time differences correspond to either simple learning problems, where all algorithms find a solution in a short period of time, or complex learning problems where CLIP explores long concepts as other algorithms.

The average runtimes of CELOE, OCEL, and CLIP across all datasets are 0.85, 8.08, and 0.1 minutes, respectively. Given that the average training time of the length predictor GRU

[5]Note that we ran OCEL with its default settings and F_1 scores are not available.

Table 4.5: Performance of CLIP compared with CELOE, OCEL, and ELTL on 100 learning problems per knowledge base. The presence of an asterisk (∗) indicates that the performance difference between CLIP and the other algorithms is significant (p-value < 0.05). The upward arrow (↑) indicates that the higher is better, whereas the downward arrow (↓) indicates the opposite. All results are average results per knowledge base. The average time is in minutes due to the long runtime of some baselines.

	Carcinogenesis			
	CELOE	OCEL	ELTL	CLIP
Acc. ↑	0.78 ± 0.27	0.89 ± 0.31	0.58 ± 0.46	**0.99 ± 0.00**
F_1 ↑	0.62 ± 0.46	–	0.51 ± 0.47	**0.96∗± 0.10**
Runtime ↓	0.93 ± 0.94	3.01 ± 0.72	0.75 ± 0.07	**0.10∗± 0.09**
Length ↓	1.69 ± 0.89	7.81 ± 6.88	**1.04∗± 0.39**	2.00 ± 1.28
	Mutagenesis			
	CELOE	OCEL	ELTL	CLIP
Acc. ↑	**0.99 ± 0.00**	0.71 ± 0.45	0.37 ± 0.43	**0.99 ± 0.00**
F_1 ↑	0.81 ± 0.35	–	0.29 ± 0.40	**0.93∗± 0.18**
Runtime ↓	0.70 ± 0.77	2.39 ± 0.18	0.29 ± 0.16	**0.07∗± 0.05**
Length ↓	2.79 ± 1.17	12.63 ± 7.03	**1.10∗± 0.81**	2.20 ± 1.16
	Semantic Bible			
	CELOE	OCEL	ELTL	CLIP
Acc. ↑	0.99 ± 0.02	0.66 ± 0.47	0.59 ± 0.37	**0.99 ± 0.00**
F_1 ↑	0.97 ± 0.10	–	0.57 ± 0.38	**0.98 ± 0.05**
Runtime ↓	0.47 ± 0.80	22.15 ± 96.55	0.09 ± 0.07	**0.06∗± 0.05**
Length ↓	3.85 ± 2.44	9.54 ± 5.73	**1.38∗± 1.76**	2.52 ± 1.26
	Vicodi			
	CELOE	OCEL	ELTL	CLIP
Acc. ↑	0.29 ± 0.44	0.25 ± 0.43	0.28 ± 0.44	**0.99∗± 0.00**
F_1 ↑	0.25 ± 0.44	–	0.25 ± 0.44	**0.97∗± 0.09**
Runtime ↓	1.30 ± 0.71	4.78 ± 1.12	1.81 ± 0.46	**0.16∗± 0.12**
Length ↓	10.79 ± 6.30	11.54 ± 6.00	11.14 ± 6.11	**1.68∗± 0.98**

is 4.10 minutes, we can conjecture the following: (1) the expected number of learning problems from which CLIP should be preferred over CELOE is 5, and (2) CLIP should be preferred over OCEL for any number of learning problems.

4.2.3 Real-world Problems

We applied CLIP to a set of real-world learning problems from different datasets. We considered three datasets: Carcinogenesis (1 learning problem), Mutagenesis (1 learning problem), Family (7 learning problems). On Carcinogenesis and Mutagenesis, the goal is to describe chemical compounds that cause cancer and those that cause gene mutation, respectively. On the Family dataset, we aim to learn the concepts Uncle, Aunt, Cousin, Great-grandfather, Great-grandmother, Great-granddaughter, and Great-grandson. Note that

each of the concepts corresponding to solutions of the given learning problems does not appear in the background knowledge nor in the training datasets.

Table 4.6: 10-fold cross-validation results on real-world problems. "–" denotes unknown learning problems.

	Train-F_1	*Test-F_1*	*Runtime (sec.)*
Family			
Aunt	0.92	0.86	14.06
Cousin	0.80	0.78	9.05
Great-granddaughter	1.00	1.00	0.64
Great-grandfather	1.00	1.00	0.75
Great-grandmother	1.00	1.00	0.82
Great-grandson	1.00	1.00	0.94
Uncle	0.94	0.93	17.68
Carcinogenesis			
–	0.74	0.70	48.48
Mutagenesis			
–	0.92	0.92	26.31

In Table 4.6, we report the results obtained on the above learning problems. We conducted experiments in a 10-fold cross-validation setting. Each of the ten folds contains some part of the positive and negative examples. For each run, CLIP solves a learning problem using 9 folds and is evaluated on the 10-th fold. We report average results across the 10 folds. From the results, we can observe that CLIP maintains a high predictive accuracy while being relatively fast.

4.3 Conclusion

We investigated the prediction of concept lengths in the description logic $\mathcal{ALC}$, to speed up the concept learning process using refinement operators. To this end, four neural network architectures were evaluated on four benchmark knowledge bases. The evaluation results suggest that all of our proposed models are superior to a random model, with recurrent neural networks performing best at this task. We showed that integrating our concept length predictors into a concept learner can reduce the search space and improve the runtime and the quality (F-measure) of solution concepts. While our proposed approach is effective on benchmark datasets, it may still not be scalable enough on very large knowledge bases where multiple learning problems are to be solved. We hence propose in the next chapter a novel family of approaches (dubbed synthesis-based approaches) that do not require a search process to compute the solution to a learning problem. This novel family of approaches can solve multiple learning problems in parallel and have shown to be several orders of magnitude faster than the state of the art.

5 Neural Class Expression Synthesis

Preamble. This chapter is based on Kouagou et al. [113] and tackles our second research question (see Section 1.2.2).

5.1 Methodology

In this section, we present our proposed family of synthesis-based approaches for class expression learning from examples. We begin with preliminaries, then formally define the problem we aim to solve, and finally present our approach in detail.

5.1.1 Preliminaries

DNN stands for deep neural network. Unless otherwise specified, $\mathcal{K} = (\texttt{TBox}, \texttt{ABox})$ is a knowledge base, N_I and N_R its sets of individuals and roles, respectively. 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 $\mathcal{K}$. As in Chapter 4, we use the representation of knowledge bases as sets of triples to compute embeddings for individuals, atomic concepts and roles, which are essential to our proposed approach (see Figure 5.1). 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 $\mathcal{V}$ 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 "PAD", which are all referred to as tokens. $\mathcal{V}[i]$ is the token at position i in $\mathcal{V}$. These constructs are used by NCES to synthesize class expressions in $\mathcal{ALC}$ (see Section 5.1.3 for details). For any class expression C, $\bar{C}$ and $\hat{C}$ are the list (in the order they appear in C) and the set of tokens in C, respectively.

5.1.2 Learning Problem

We adapt the classical definition of a learning problem (see Definition 4.1) to our setting of class expression synthesis (Definition 5.1).

Definition 5.1. *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 tokens (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 4.1 and Definition 5.1; our NCES generate only one, though we can also, in principle, generate multiple solutions, e.g., by randomly removing some of the input examples and computing a new solution every time. 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 synthesize class expressions with low numbers of false positives and false negatives.

5.1.3 Neural Class Expression Synthesizers

The fundamental hypothesis behind neural class expression synthesizers (NCES) is that one should be able to extract enough semantics from latent representations (embeddings) of examples to directly synthesize class expressions. This approach, analogous to machine translation, eliminates the necessity for computationally expensive explorations as currently done in search-based approaches. Our hypothesis is supported by the significant improvement in the performance of machine translation approaches brought about by neural machine translation (NMT) [35, 228]. 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 (e.g., description logics).

We propose the following recipe to implement the idea behind NCES. First, given a knowledge base $\mathcal{K}$, generate numerous class expressions of various lengths together with their sets of positive (actual instances) and negative examples, as done in the previous chapter, Section 4.1.3. Construct set of negative examples for each class expression by using the closed world assumption. Second, extract assertions and axioms of the form $C(a) \equiv (a, \texttt{rdf:type}, C)$, $r(a,b) \equiv (a, r, b)$, and $C \sqsubseteq D \equiv (C, \texttt{rdfs:subClassOf}, D)$ to form a set of triples, i.e., a knowledge graph. 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 [45] which applies convolutions on complex-valued vectors, and TransE [23] which projects entities (individuals in $\mathcal{K}$) and relations (roles in $\mathcal{K}$) onto a Euclidean space and uses the Euclidean distance to model interactions. The computed embeddings are then used as features for an NCES instance, i.e., a model able to take a set of embeddings as input and encode a sequence of

tokens as output (see Figure 5.1). Again, any embedding model can be used; we chose ConEx because of its computational efficiency and expressive power. We employ the embedding model TransE to demonstrate that NCES can perform relatively well even with embeddings computed by simple models. Finally, train NCES instances on the generated data to obtain fast concept learners.

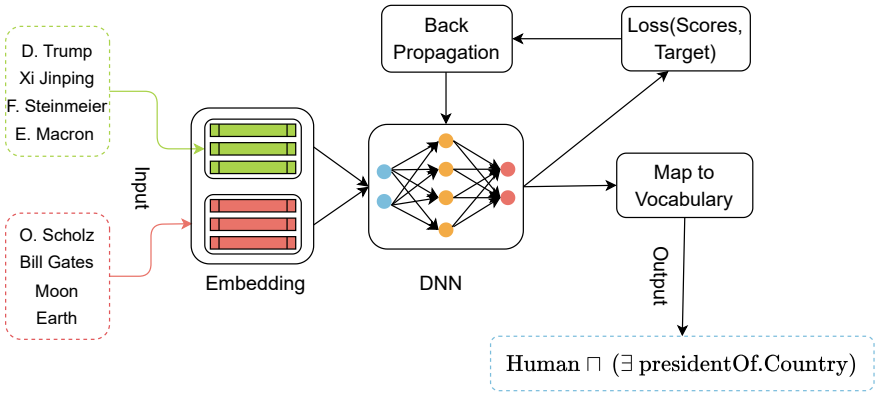


Figure 5.1: NCES architecture. DNN is a deep neural network that produces a sequence of tokens in the vocabulary (e.g., a sequence-to-sequence or a set-to-sequence model). The input consists of positive examples (upper left, dotted green box) and negative examples (bottom left, dotted red box). Positive examples are presidents of countries while negative examples are not. For these specific inputs, NCES is trained to output an expression describing positive examples, e.g., `Human` $\sqcap$ ($\exists$ `presidentOf.Country`).

Neural Network Architectures

We conducted our experiments using the following network architectures: Long Short-Term Memory (LSTM) [85], Gated Recurrent Unit (GRU) [36], and Set-Transformer [118]. 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 [247, 250]. 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 5.2.1).

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, ..., 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:

$$h_1^{pos}, \ldots, h_{n_1}^{pos} = RNN(x_{pos});\ h_1^{neg}, \ldots, h_{n_2}^{neg} = RNN(x_{neg}) \tag{5.1}$$

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:

$$h_{pos} := \sum_{t=1}^{T_1} h_t^{pos};\ h_{neg} := \sum_{t=1}^{T_2} h_t^{neg};\ h := \mathrm{Concat}(h_{pos}, h_{neg}); \tag{5.2}$$

$$s = W_3(\mathrm{bn}(W_2 f(W_1 h + b_1) + b_2)) + b_3. \tag{5.3}$$

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 (see more details in Chapter 2, Section 2.1.3) comprises an encoder Enc and a decoder Dec. The encoder is a stack of two ISAB layers with 4 attention heads, $m = 32$ inducing points. The decoder operates with 4 attention heads, one PMA layer with a single seed vector ($k = 1$), and one 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:

$$s_{pos} = Enc(x_{pos}),\ s_{neg} = Enc(x_{neg}); \tag{5.4}$$

$$s = Dec(\mathrm{Concat}(s_{pos}, s_{neg})). \tag{5.5}$$

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 s from 5.3 and 5.5 is reshaped into a $|\mathcal{V}| \times L$ 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 5.6) and update model weights through gradient descent during training.

Loss Function

We train our NCES instances using the multi-dimensional cross-entropy[1] loss function $\mathcal{L}$ defined by

$$\mathcal{L}(x, y) = -\frac{1}{NL} \sum_{i=1}^{N} \sum_{j=1}^{L} \log \left(\frac{\exp(x_{i,y_{ij},j})}{\sum_{c=1}^{C} \exp(x_{i,c,j})} \right), \tag{5.6}$$

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 tokens (i.e., the output scores s from Equations 5.5 and 5.3), and $y \in [1, 2, \ldots, C]^{N \times L}$ is the minibatch of actual class indices. Minimizing $\mathcal{L}$ constrains the model to assign a high score to the entries corresponding to the correct tokens ($\exp(x_{i,y_{ij},j}) \approx 1$) 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, $C = |\mathcal{V}|$. Note that $\mathcal{V}$ contains the special token "PAD" that we use 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. In 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 [244].

Metrics for Training

Apart from the loss function above, 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 tokens in the target expression, while the latter additionally measures the correct ordering of the selected tokens. Formally, let T and P be the target and predicted class expressions, respectively. Recall the notation $\bar{C}$ and $\hat{C}$ introduced in Section 5.1.1 for any class expression C. The soft (Acc_s) accuracy and hard accuracy (Acc_h) are defined as follows:

$$Acc_s(T, P) = \frac{|\hat{T} \cap \hat{P}|}{|\hat{T} \cup \hat{P}|}; \; Acc_h(T, P) = \frac{\sum_{i=1}^{\min(l_1, l_2)} \mathbb{1}(\bar{T}[i], \bar{P}[i])}{\max(l_1, l_2)}, \tag{5.7}$$

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

[1] Here, multi-dimensional refers to the fact that at each position along the output sequence dimension, we compute the cross-entropy as defined in Chapter 2

Class Expression Synthesis and Model Ensembling

We synthesize class expressions by mapping the output scores s (see Equations 5.3 and 5.5) to the vocabulary. More specifically, we select the highest-scoring tokens in the vocabulary for each position along the sequence dimension:

$$\mathrm{id}_j = \underset{c \in \{1,\dots,C\}}{\arg\max}\ s_{c,j} \text{ for } j = 1, \dots, L, \tag{5.8}$$

$$synthesized\_token_j = \mathcal{V}[\mathrm{id}_j]. \tag{5.9}$$

Ensemble learning has proven to be one of the most robust approaches for tasks involving complex noisy data [56, 183]. In this work, we combine class expression synthesizers' predictions post training by averaging the predicted scores. Specifically, given the reshaped[2] output scores $s_i \in \mathbb{R}^{C \times L}$ $(i = 1, 2, 3)$ as defined in 5.3 and 5.5 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:

$$s = \frac{\sum_{i \in \mathcal{I}} s_i}{|\mathcal{I}|} \text{ with } \mathcal{I} \in 2^{\{1,2,3\}} \text{ and } |\mathcal{I}| \geq 2. \tag{5.10}$$

Then, the synthesized expression is constructed following Equations 5.8 and 5.9 using the average scores s.

5.2 Experiments

In this section, we conduct experiments to evaluate neural class expression synthesizers. We first describe our dataset construction method before presenting hyperparameter configurations and the hardware used in our experiments. Finally, we present and discuss our experimental results.

5.2.1 Experimental Setup

Datasets

We evaluated our proposed approach on the Carcinogenesis [224], Mutagenesis [224], Semantic Bible[3], and the Vicodi [152] knowledge bases. Carcinogenesis and Mutagenesis are knowledge bases about chemical compounds and how they relate to each other. The

[2] Direct outputs from neural networks are vectors of size $C * L$ which we reshape into matrices of size $C \times L$.

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

Semantic Bible knowledge base describes each named 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 5.1.

Table 5.1: Detailed information about the datasets used for evaluation. $|N_I|$, $|N_C|$, and $|N_R^a|$ are the numbers of individuals, atomic concepts, and abstract roles in the knowledge base, respectively. $|Train|$, $|LPs|$, and $|\mathcal{V}|$ are the training set size, test set size, and number of tokens in the vocabulary, respectively.

	$\lvert N_I\rvert$	$\lvert N_C\rvert$	$\lvert N_R^a\rvert$	$\lvert$TBox$\rvert$	$\lvert$ABox$\rvert$	$\lvert Train\rvert$	$\lvert LPs\rvert$	$\lvert\mathcal{V}\rvert$
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

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. [107] 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 a 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 [82]. 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 hyperparameter $n = n_1 + n_2$ 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.

Hyperparameter Optimization

We employed random search on the hyperparameter space since it often yields good results while being computationally more efficient than grid search [20]; the selected values—those with the best results—are reported in Table 5.2. In the table, it can be seen that most knowledge bases share the same optimal values of hyperparameters: the minibatch size N (a.k.a. batch size), the number of training epochs $Epochs$, the optimizer $Opt.$, 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, $n = \min(\frac{|N_I|}{2}, 1000)$ appears to work well with

our evaluation datasets. This suggests that one can effortlessly find fitting hyperparameters for new datasets.

Table 5.2: Hyperparameter settings per dataset. Recall that m is the number of inducing points in the Set-Transformer model, and n is the number of examples.

	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

At inference time, 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.

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 5.3. 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.

Table 5.3: 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

5.2.2 Results and Discussion

Syntactic Accuracy

Our neural class expression synthesizers are trained for 300 epochs on each knowledge base. In Figures 5.2, 5.3 and 5.4, we show the training curves of NCES on all benchmark datasets.

These curves suggest that NCES instances train fast with an exponential growth (decrease) in accuracy (in loss) within the first 10 epochs. All models achieve over 95% syntactic accuracy on large knowledge bases (Carcinogenesis, Mutagenesis, and Vicodi). On the

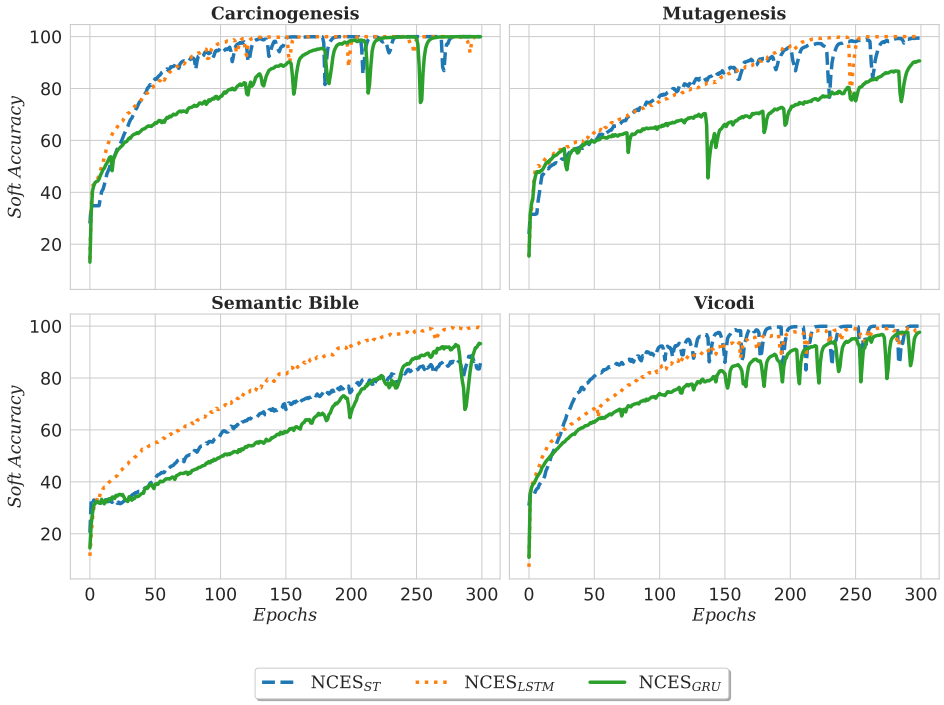


Figure 5.2: Soft accuracy curves on the training set

smallest knowledge base, Semantic Bible, we observe that NCES$_{ST}$ drops in performance as it achieves only 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 5.4), where NCES instances significantly outperform search-based approaches, including CELOE, on large datasets.

Comparison to the State of the Art

We compare our approach against EvoLearner, CELOE, ECII, ELTL. The maximum execution time for CELOE and EvoLearner is set to 300 seconds per learning problem while ECII and ELTL are executed with their default settings, as they do not have the maximum execution time parameter in their original implementation. From Table 5.4, 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 5.5 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

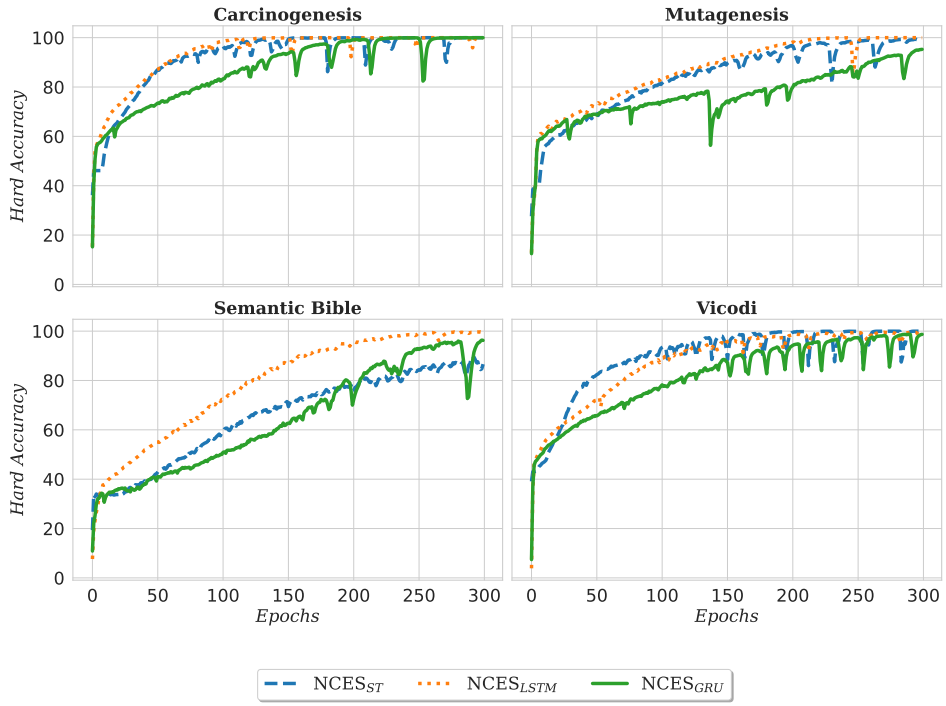


Figure 5.3: Hard accuracy curves on the training set

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 (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.

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 instances are 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`) $\sqcap$ ($\neg$`Engineer`). The solutions computed by each of the approaches are as follows: CELOE: `Flavour` $\sqcap$ ($\neg$`Battle`) $\sqcap$ ($\neg$`Person`) [F_1: 0.02], ELTL: `Flavour` $\sqcap$ ($\exists$ `related`.($\exists$ `related.Role`)) [F_1: 0.00], ECII: `Organisation` $\sqcup$ $\neg$`VicodiOI` [F_1: 0.13], EvoLearner: `Military-Organisation` $\sqcap$ `Military-Organisation` [F_1: 0.73],

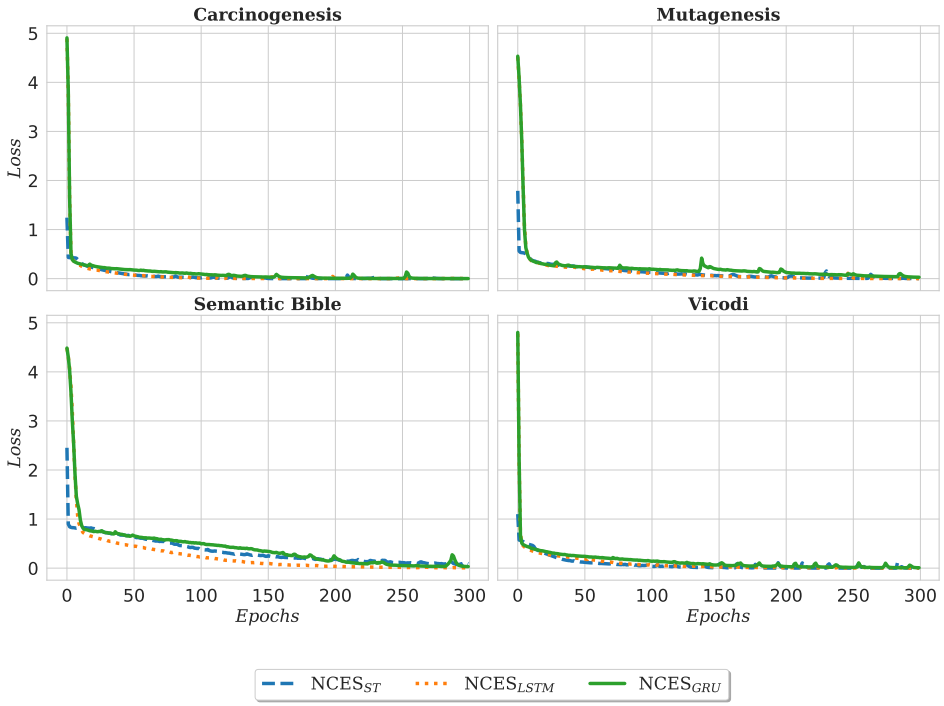


Figure 5.4: Loss curves on the training set

and NCES$_{ST+GRU}$[4]: (`Disaster` ⊔ `Military-Organisation`) ⊓ (¬`Engineer`) [F_1: 1.00]. 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 0.73. 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.

5.2.3 Real-world Problems

We applied ROCES to a set of real-world learning problems from different datasets. We considered three datasets: Carcinogenesis (1 learning problem), Mutagenesis (1 learning

[4]Here, NCES uses ConEx embeddings

Table 5.4: 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 higher is better, and ↓ indicates the opposite. Bold values correspond to the best performance. Underlined values are the second best.

	F_1 ↑			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
CELOE	0.38±0.44	0.83±0.33	**0.93±0.18***	0.36±0.42
ELTL	0.13±0.26	0.29±0.34	0.43±0.38	0.17±0.33
ECII	0.16±0.28	0.27±0.32	0.34±0.38	0.44±0.36
EvoLearner	0.91±0.14	**0.93±0.13**	0.92±0.10	0.93±0.10
NCES_{LSTM}	0.82±0.29	0.81±0.28	0.72±0.34	0.72±0.35
NCES_{GRU}	0.90±0.25	0.78±0.31	0.52±0.40	0.87±0.27
NCES_{ST}	0.90±0.25	0.81±0.34	0.73±0.39	0.78±0.37
NCES_{ST+GRU}	0.97±0.13	0.90±0.26	0.84±0.27	0.93±0.20
$\text{NCES}_{ST+LSTM}$	0.97±0.13	0.89±0.26	0.81±0.28	0.91±0.22
$\text{NCES}_{GRU+LSTM}$	0.95±0.15	0.81±0.31	0.76±0.32	0.88±0.24
$\text{NCES}_{ST+GRU+LSTM}$	**0.97±0.13***	0.91±0.23	0.87±0.24	**0.96±0.12***
	Accuracy ↑			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
CELOE	0.67±0.25	0.94±0.12	0.98±0.05	0.85±0.18
ELTL	0.25±0.33	0.38±0.39	0.47±0.38	0.33±0.42
ECII	0.26±0.36	0.32±0.38	0.30±0.38	0.77±0.35
EvoLearner	0.99±0.01	**0.99±0.02**	**0.99±0.03***	**0.99±0.05**
NCES_{LSTM}	0.98±0.11	0.98±0.07	0.90±0.24	0.94±0.21
NCES_{GRU}	0.99±0.04	0.98±0.04	0.81±0.31	0.98±0.11
NCES_{ST}	0.97±0.14	0.91±0.25	0.84±0.33	0.90±0.28
NCES_{ST+GRU}	0.99±0.09	0.97±0.12	0.92±0.21	0.97±0.13
$\text{NCES}_{ST+LSTM}$	0.99±0.06	0.97±0.12	0.90±0.22	0.96±0.16
$\text{NCES}_{GRU+LSTM}$	**1.00±0.00**	0.98±0.05	0.87±0.27	0.97±0.15
$\text{NCES}_{ST+GRU+LSTM}$	0.99±0.09	0.98±0.06	0.95±0.18	0.97±0.12
	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
$\text{NCES}_{ST+LSTM}$	0.23±0.00	0.23±0.00	0.11±0.00	0.11±0.00
$\text{NCES}_{GRU+LSTM}$	0.24±0.00	0.32±0.00	0.13±0.00	0.17±0.00
$\text{NCES}_{ST+GRU+LSTM}$	0.27±0.00	0.31±0.00	0.15±0.00	0.15±0.00

problem), Family (7 learning problems). On Carcinogenesis and Mutagenesis, the goal is to describe chemical compounds that cause cancer and those that cause gene mutation, respectively. On the Family dataset, we aim to learn the concepts Uncle, Aunt, Cousin, Great-grandfather, Great-grandmother, Great-granddaughter, and Great-grandson. Note that each of the concepts corresponding to solutions of the given learning problems does not appear in the background knowledge nor in the training datasets.

Table 5.5: Evaluation results using TransE embeddings. The star (∗) indicates statistically significant differences between the best search-based and the best synthesis-based approaches. ↑ indicates that the higher the better, and ↓ indicates the opposite. Bold values correspond to the best performance. Underlined values are the second best.

	F_1 ↑			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
CELOE	0.38±0.44	0.83±0.33	**0.93±0.18**∗	0.36±0.42
ELTL	0.13±0.26	0.29±0.34	0.43±0.38	0.17±0.33
ECII	0.16±0.28	0.27±0.32	0.34±0.38	0.44±0.36
EvoLearner	0.91±0.14	**0.93±0.13**	0.92±0.10	**0.93±0.10**
NCES$_{LSTM}$	0.84±0.27	0.76±0.34	0.60±0.35	0.79±0.30
NCES$_{GRU}$	0.87±0.27	0.78±0.36	0.66±0.33	0.80±0.33
NCES$_{ST}$	0.87±0.30	0.79±0.34	0.68±0.37	0.78±0.36
NCES$_{ST+GRU}$	0.97±0.10	0.91±0.21	0.75±0.33	0.89±0.25
NCES$_{ST+LSTM}$	0.93±0.22	0.90±0.20	0.76±0.34	0.89±0.25
NCES$_{GRU+LSTM}$	0.92±0.21	0.81±0.33	0.76±0.32	0.88±0.25
NCES$_{ST+GRU+LSTM}$	**0.98±0.12**∗	0.91±0.21	0.86±0.25	0.90±0.24
	Accuracy ↑			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
CELOE	0.67±0.25	0.94±0.12	0.98±0.05	0.85±0.18
ELTL	0.25±0.33	0.38±0.39	0.47±0.38	0.33±0.42
ECII	0.26±0.36	0.32±0.38	0.30±0.38	0.77±0.35
EvoLearner	**1.00±0.01**	**0.99±0.02**	**0.99±0.03**∗	**0.99±0.05**
NCES$_{LSTM}$	0.97±0.14	0.97±0.11	0.84±0.25	0.97±0.12
NCES$_{GRU}$	0.98±0.13	0.95±0.15	0.83±0.29	0.96±0.16
NCES$_{ST}$	0.94±0.23	0.93±0.21	0.82±0.33	0.92±0.25
NCES$_{ST+GRU}$	0.99±0.06	0.99±0.03	0.86±0.28	0.96±0.15
NCES$_{ST+LSTM}$	0.98±0.11	0.99±0.03	0.89±0.24	0.95±0.18
NCES$_{GRU+LSTM}$	0.99±0.09	0.96±0.14	0.89±0.25	0.97±0.13
NCES$_{ST+GRU+LSTM}$	0.99±0.09	0.99±0.05	0.93±0.18	0.95±0.19
	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$_{LSTM}$	0.09±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

In Table 5.6, we report the results obtained on the above learning problems. We conducted experiments in a 10-fold cross-validation setting. Each of the ten folds contains some part of the positive and negative examples. For each run, NCES solves a learning problem (100 predictions per problem) using 9 folds and is evaluated on the 10-th fold. We report average results across the 10 folds. From the results, we can observe that NCES maintains a high

Table 5.6: 10-fold cross-validation results on real-world problems. "–" denotes unknown learning problems.

	Train-F_1	*Test-F_1*	*Runtime (sec.)*
Family			
Aunt	0.76	0.81	0.63
Cousin	0.68	0.61	0.63
Great-granddaughter	1.00	1.00	0.51
Great-grandfather	0.91	0.93	0.51
Great-grandmother	0.92	0.95	0.63
Great-grandson	0.91	0.91	0.60
Uncle	0.82	0.85	0.54
Carcinogenesis			
–	0.71	0.71	1.72
Mutagenesis			
–	0.70	0.70	1.61

predictive accuracy while being scalable[5]. The carcinogenesis and mutagenesis problems are known to be challenging (see, e.g., [27], where a search-based approach achieves only 72% accuracy on Carcinogenesis) which also explains the relatively low performance achieved by NCES on these problems.

5.3 Conclusion

In this chapter, we presented 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 learning problem, 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 the next chapter, we will show how NCES approaches can be extended to more expressive description logics, e.g., $\mathcal{ALCHIQ}^{(\mathcal{D})}$. We will also investigate ways to reduce the dependence of NCES on pretrained knowledge base embeddings.

[5] For each run, we perform 100 predictions per problem and select the class expression with the highest F_1 score. Hence, the runtime per prediction is ≈0.006 second on Family and 0.01 second on Mutagenesis and Carcinogenesis.

6 Neural Class Expression Synthesis in $\mathcal{ALCHIQ}^{(\mathcal{D})}$

Preamble. This chapter is based on Kouagou et al. [109] and answers the third and fourth research questions (see Section 1.2.3).

6.1 Methodology

In this section, we present NCES2, a new instance of our family of neural class expression synthesizers (NCES) which supports the description logic $\mathcal{ALCHIQ}^{(\mathcal{D})}$ and does not require pretrained embeddings of input knowledge bases.

6.1.1 Preliminaries

A knowledge base is denoted $\mathcal{K} = (\texttt{TBox}, \texttt{ABox})$, and N_I, N_C, and N_R represent its sets of individuals, atomic concepts, and roles, respectively. $\mathcal{V}_\mathcal{K}$ (or simply $\mathcal{V}$ when there is no ambiguity) is a vocabulary of tokens consisting of all atomic concept and role names in $\mathcal{K}$, together with the following atoms which are necessary for our target description logic $\mathcal{ALCHIQ}^{(\mathcal{D})}$: "$\top$" (top concept), "$\bot$" (bottom concept), "False", "True" (Boolean values), "$^-$" (for inverse properties), ":", "xsd", "double", "integer", "date" (for time data values), "$\leq$", "$\geq$", " " (white space), "." (dot), "$\sqcup$", "$\sqcap$", "$\exists$", "$\forall$", "$\neg$", "[", "]", "{", "}", "(", and ")". We also add numeric data values to the vocabulary. These values are obtained by creating evenly spaced bins ranging from the lowest to the highest value observed in the knowledge base, following the information gain approach in [82]. As in the previous chapter, we include the special token "PAD" to pad all class expressions in a batch of training examples to the same, predefined maximum length. The token also serves as the end token at inference time when parsing the output of NCES2.

We choose a fixed ordering for the elements of the vocabulary $\mathcal{V}$ and use them to synthesize class expressions (more details in Section 6.1.3). In fact, class expressions in $\mathcal{ALCHIQ}^{(\mathcal{D})}$ are written using tokens in $\mathcal{V}$ as can be seen in the learning problems below, which are extracted from test datasets: LP_1 = `Man` $\sqcap$ ($\forall$ `knows.`($\neg$`SonOfGod`)) $\sqcap$ ($\leq$ `2 visitedPlace.`$\top$) (Semantic Bible), LP_2 = `Fluorine-92` $\sqcup$ `Sulfur-74` $\sqcup$ ($\exists$ `drosophila_rt.`{False}) (Carcinogenesis), LP_3 = (`Atom` $\sqcap$ (`Tin` $\sqcup$ ($\neg$`Carbon-25`))) $\sqcup$ ($\exists$ `inBond.`($\neg$`Carbon-10`)) (Mutagenesis), and LP_4 = `Measurable-Trend` $\sqcup$ ($\exists$ `related.`(`Idea` $\sqcup$ `Uprising`)) (Vicodi). We

discuss the solutions computed by different class expression learning approaches for each of these learning problems in Section 6.2.

6.1.2 Learning Problem

Definition 6.1 (Solution by NCES2). *Given a knowledge base $\mathcal{K}$ and sets of positive/negative examples $E^+ = \{e_1^+, e_2^+, \ldots, e_{n_1}^+\}$ and $E^- = \{e_1^-, e_2^-, \ldots, e_{n_2}^-\}$, the learning problem is to compute a class expression C in $\mathcal{ALCHIQ}^{(\mathcal{D})}$ (using tokens in the vocabulary $\mathcal{V}$) that maximizes the F-measure and Accuracy defined by*

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}, \tag{6.1}$$

$$Precision = \frac{|\mathcal{E}_C^+ \cap E^+|}{|\mathcal{E}_C^+ \cap E^+| + |\mathcal{E}_C^+ \cap E^-|}, \tag{6.2}$$

$$Recall = \frac{|\mathcal{E}_C^+ \cap E^+|}{|\mathcal{E}_C^+ \cap E^+| + |E^+ \setminus \mathcal{E}_C^+|}, \tag{6.3}$$

$$Accuracy = \frac{|\mathcal{E}_C^+ \cap E^+| + |\mathcal{E}_C^- \cap E^-|}{|E^+| + |E^-|}, \tag{6.4}$$

where $\mathcal{E}_C^+$ is the set of instances of C, and $\mathcal{E}_C^- = N_I \setminus \mathcal{E}_C^+$.

The metrics $Accuracy$ and F_1 are used to compare different approaches on class expression learning problems—see Table 5.4. One difference between Definition 4.1 and Definition 6.1 is for example that the latter targets a specific description logic (in this case $\mathcal{ALCHIQ}^{(\mathcal{D})}$) while the former is general, i.e., applicable to any description logic. Moreover, Definition 6.1 allows for approximate solutions to be returned when the exact solution is not found, while Definition 4.1 does not.

6.1.3 Proposed Approach

We now present the main components of NCES2. The first component is the embedding model which is integrated into the training and inference pipelines. This component, represented by the three yellow bars in Figure 6.1, takes input examples and converts them into vectors in $\mathbb{R}^d$. The second component, also involved in both training and inference phases, is a deep neural network able to take a set of vectors and return a sequence of tokens (see for example the output `Human` $\sqcap \geq$ 2 `hasWon`.NobelPrize in Figure 6.1). The two components interact during training and inference (class expression learning) as elucidated below.

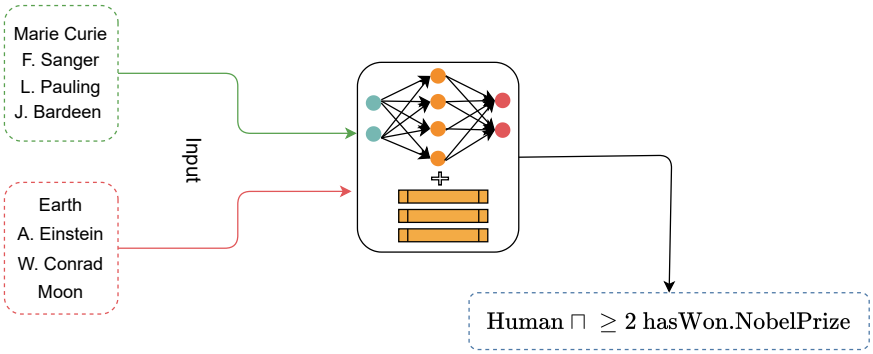


Figure 6.1: Architecture of NCES2

Encoding Positive and Negative Examples

We use Set-Transformer [118] which is an encoder-decoder architecture with attention mechanism (see Chapter 2, Section 2.1.3 for more details). The encoder Enc consists of two ISAB layers. The decoder Dec is composed of one PMA layer (with $k = 1$), and a linear layer which helps us obtain the desired output shape. During training, the embedding model component provides embeddings for positive examples x_{pos} and negative examples x_{neg}. These two embedding matrices are fed to the encoder independently. The outputs are then concatenated row-wise and fed to the decoder which produces the final scores s for all tokens in the vocabulary $\mathcal{V}$:

$$O_{pos} = Enc(x_{pos}),\ O_{neg} = Enc(x_{neg}), \tag{6.5}$$

$$s = Dec(\text{Concat}(O_{pos}, O_{neg})). \tag{6.6}$$

The outputs from the decoder are used in the loss function (see Equation 6.8) to train the neural synthesizer. They also serve as the basis to select the correct tokens in the output sequence at inference time (see Equation 6.10).

Loss Function

Our approach is trained by minimizing two joint loss functions: (1) The loss $\mathcal{L}_1$ from the embedding model, and (2) the loss $\mathcal{L}_2$ from the class expression synthesizer. Formally, let $\mathcal{G} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ be the knowledge graph representation of the input knowledge base (see Section 5.1.3 for how the conversion is carried out), and let $h \in \mathcal{E}, r \in \mathcal{R}$ be a head entity and a relation. We define $\mathcal{L}_1$ to be the binary cross-entropy loss (assuming a KvsAll training technique [53]):

$$\mathcal{L}_1(y^{hr}, \hat{y}^{hr}) = -\frac{1}{|\mathcal{E}|} \sum_{i=1}^{|\mathcal{E}|} y_i^{hr} \log(\hat{y}_i^{hr}) + (1 - y_i^{hr}) \log(1 - \hat{y}_i^{hr}). \tag{6.7}$$

Here, $y^{hr} \in \{0,1\}^{|\mathcal{E}|}$ is the binary representation of $\{(h,r,t)|t \in \mathcal{E}\}$ in $\mathcal{G}$, i.e., $y^{hr}[\mathrm{id}(t)] = 1$ if $(h,r,t) \in \mathcal{G}$, and $y^{hr}[\mathrm{id}(t)] = 0$ otherwise. Accordingly, $\hat{y}^{hr} \in [0,1]^{|\mathcal{E}|}$ is the vector of scores predicted by the embedding model for all candidate tail entities. On the other hand, $\mathcal{L}_2$ is defined by

$$\mathcal{L}_2(s,t) = -\frac{1}{L}\sum_{i=1}^{L} \log\left(\frac{\exp(s_{t_i,i})}{\sum_{c=1}^{|\mathcal{V}|}\exp(s_{c,i})}\right), \tag{6.8}$$

where L is the maximum length of class expressions our approach can generate, $|\mathcal{V}|$ the total number of tokens in the vocabulary, $s \in \mathbb{R}^{|\mathcal{V}|\times L}$ the matrix of predicted scores for each position in the target sequence of tokens, and $t \in \{1,2,\ldots,|\mathcal{V}|\}^L$ the vector of target token indices in the input class expression.

Our total loss $\mathcal{L}$ is defined as the average of $\mathcal{L}_1$ and $\mathcal{L}_2$, computed on the inputs $(y^{hr}, \hat{y}^{hr})$ and (s,t):

$$\mathcal{L}(y^{hr}, \hat{y}^{hr}, s, t) = \frac{\mathcal{L}_1(y^{hr}, \hat{y}^{hr}) + \mathcal{L}_2(s,t)}{2}. \tag{6.9}$$

During training, we alternatively sample a minibatch of N_1 training datapoints for the embedding model, and a minibatch of N_2 training datapoints for the neural synthesizer to compute $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. We then compute the gradient of $\mathcal{L}$ w.r.t. both the parameters of the embedding model and those of the synthesizer. To prevent gradient explosion and to reduce overfitting, we use gradient clipping [244], and dropout [196]. Both parameter sets are updated using the Adam [104] optimization algorithm. Note that the embeddings of positive and negative examples used by the synthesizer—see Section 6.1.3—come from the embedding model and are hence dynamically updated during training. In this way, we are able to learn embeddings that are not only finetuned for class expression learning, but also faithful to the input background knowledge.

Class Expression Synthesis

We synthesize class expressions by mapping the output scores s (see Equation 6.6) to the vocabulary. Specifically, we select the highest-scoring token in the vocabulary for each position i along the sequence dimension:

$$\mathrm{id}_i = \underset{c\in\{1,\ldots,|\mathcal{V}|\}}{\arg\max}\ s_{c,i}, \tag{6.10}$$

$$\texttt{synthesized\_token}_i = \mathcal{V}[\mathrm{id}_i]. \tag{6.11}$$

The predicted tokens are concatenated to construct a class expression. Note that we ignore all tokens appearing after the special token “PAD”.

6.1.4 Model Ensembling

Several works have highlighted that combining different neural models trained even on the same dataset usually performs better than each individual model [56, 183]. This technique is known as model ensembling. In this work, we trained three instances of our approach, NCES2, which all use the Set-Transformer architecture but with different numbers of inducing points: $m = 32$, $m = 64$, and $m = 128$. We compute ensemble predictions by averaging the predicted scores for each token post training. Overall, we consider four ensemble models: NCES2$_{m=\{32,64\}}$, NCES2$_{m=\{32,128\}}$, NCES2$_{m=\{64,128\}}$, and NCES2$_{m=\{32,64,128\}}$. A class expression is then synthesized as described in Section 6.1.3 using the average scores.

6.2 Experiments

In this section, we conduct experiments to evaluate our proposed approach. We start with experimental settings and then present and discuss results.

6.2.1 Experimental Setup

Datasets

We used four benchmark datasets in our experiments: Vicodi [152], Carcinogenesis [224], Mutagenesis [224], and Semantic Bible[1]. The Carcinogenesis and Mutagenesis knowledge bases describe chemical compounds and how they relate to each other. The Semantic Bible knowledge base describes the New Testament, and the Vicodi knowledge base describes the European history. We summarize the statistics of each dataset in Table 6.1.

Training Data Generation

Training NCES2 requires numerous class expressions with their sets of positive and negative examples[2]. To this end, we extend the refinement operator we developed in Chapter 4 to the description logic $\mathcal{ALCHIQ}^{(\mathcal{D})}$ so that we can generate all forms of class expressions supported by NCES2 (see Table 2.1 for syntax and semantics of $\mathcal{ALCHIQ}^{(\mathcal{D})}$). Moreover, we improve upon the training data generation method used in [113]. Since most knowledge bases contain thousands to millions of individuals, previous NCES approaches subsample the initial sets of positive/negative examples for each learning problem in the training set. This technique is inefficient because only a few examples are seen during training which

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

[2] Positive examples are instances of a class expression while negative examples are the rest of the individuals in N_I.

Table 6.1: Statistics of benchmark datasets. In the table, we use the following notations and abbreviations: Set of individuals (N_I), set of atomic concepts (N_C), set of abstract roles ($Obj.\ Pr.$), set of concrete roles ($D.\ Pr.$), vocabulary ($\mathcal{V}$), and learning problems in the test set (LPs).

	$\lvert N_I\rvert$	$\lvert N_C\rvert$	$\lvert Obj.\ Pr.\rvert$	$\lvert D.\ Pr.\rvert$	$\lvert TBox\rvert$	$\lvert ABox\rvert$	$\lvert\mathcal{V}\rvert$	$\lvert Train\rvert$	$\lvert LPs\rvert$
Carcinogenesis	22,372	142	4	15	144	74,223	198	19,635	100
Mutagenesis	14,145	86	5	6	82	47,722	133	9,705	100
Semantic Bible	724	48	29	9	56	3,106	125	11,069	100
Vicodi	33,238	194	10	2	204	116,181	242	46,094	100

results in poor performance on learning problems where, e.g., a different random seed is used to construct the sets of examples. To alleviate this issue, we construct multiple copies (2 copies in our experiments) of a given learning problem and assign different subsets of examples to each copy. The clear advantage of this new sampling technique is that it allows each learning problem to be seen from different perspectives and hence better understood. Note that this sampling technique is only applied to the training set. The statistics of the generated data are given in Table 6.1.

Measuring Performance during Training

In Chapter 5 (Section 5.1.3), we introduced two metrics to quantify the performance of neural synthesizers during training[3]. We use the same metrics in this chapter. The first metric is called "*Soft Accuracy*" and is equivalent to the *Jaccard index* between the set of predicted tokens and the set of true tokens in the input expression. The second metric is called "*Hard Accuracy*", and compares the tokens in the prediction and target expressions position-wise, i.e., taking into account their order of appearance.

Hyperparameter Search

Following [113], we employ a random search [20] to find the best hyperparameter values for NCES2. Specifically, we find the best values on one dataset (we used Carcinogenesis for this purpose) and use them on the rest of the datasets. Note that the total number of examples n is an exception as it depends on the size of N_I, i.e., the total number of individuals in the given knowledge base. Nonetheless, we used the same formula to compute the optimal value for $n : \min\left(\frac{|N_I|}{2}, 1000\right)$. The selected values for hyperparameters are presented in Table 6.2. From the table, we can observe that optimal values are mostly the same across all datasets, suggesting that our approach NCES2 does not require an expensive search over the hyperparameter space.

[3]These metrics are used only during training. When comparing NCES2 to state-of-the-art approaches on class expression learning on the test sets, we use metrics based on the number of covered/ruled-out positive/negative examples for all approaches.

Table 6.2: Hyperparameter settings per dataset. L is the maximum length of expressions synthesized by NCES2, Lr the learning rate, N_1 the minibatch size for the embedding model, N_2 the minibatch size for the synthesizer, n the number of (positive and negative) examples, d the embedding dimension, and gc the gradient clipping value.

Dataset	*Epochs*	*Optimizer*	*Lr*	d	N_1	N_2	L	n	gc
Carcinogenesis	200	Adam	0.001	50	1,024	512	48	1,000	5
Mutagenesis	200	Adam	0.001	50	1,024	512	48	1,000	5
Semantic Bible	200	Adam	0.001	50	1,024	512	48	362	5
Vicodi	200	Adam	0.001	50	1,024	512	48	1,000	5

Hardware, Model Size and Training Time

We trained NCES2 using 24GB RAM, 16 AMD EPYC 7713 CPUs @3.10GHz, and a single NVIDIA RTX A5000 GPU with 24GB memory. Because search-based approaches do not support GPU computation, we conduct experiments on class expression learning on the test sets (see Table 6.4) using a server with 16 Intel Xeon E5-2695 CPUs @2.30GHz and 128GB RAM.

We report the runtime and model size in Table 6.3.

Table 6.3: Model size and training time. The runtime is in hours (h).

	Carcinogenesis		Mutagenesis		Semantic Bible		Vicodi	
	|Params|	*Time (h)*	*|Params|*	*Time (h)*	*|Params|*	*Time (h)*	*|Params|*	*Time (h)*
NCES2$_{m=32}$	2,747,376	1.61	1,974,682	0.84	1,233,552	0.39	3,610,516	3.79
NCES2$_{m=64}$	2,755,568	1.86	1,982,874	0.94	1,241,744	0.45	3,618,708	4.38
NCES2$_{m=128}$	2,771,952	2.40	1,999,258	1.20	1,258,128	0.55	3,635,092	5.61

6.2.2 Results and Discussion

Training Curves

NCES2 is trained for 200 epochs on each dataset. Training curves are shown in Figures 6.3, 6.2, and 6.4. From these figures, we can observe that NCES2 is able to accurately map instance data (positive/negative examples) to the corresponding class expressions on the training set. This is witnessed by a performance of over 95% in the two metrics *Soft Accuracy* and *Hard Accuracy* (recall the definition in Section 6.2.1), and a decreasing loss which approaches zero on all datasets. In addition, the convergence rates are higher on the largest datasets (Carcinogenesis and Vicodi), which suggests that NCES2 learns faster on large datasets.

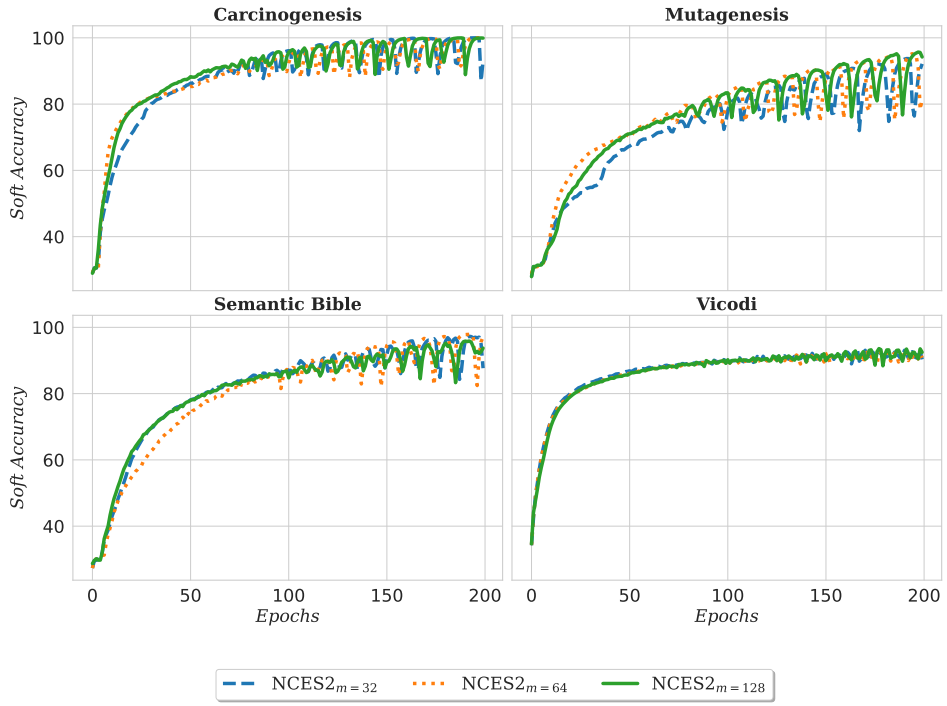


Figure 6.2: Training soft accuracy (in %) curves. m is the number of inducing points.

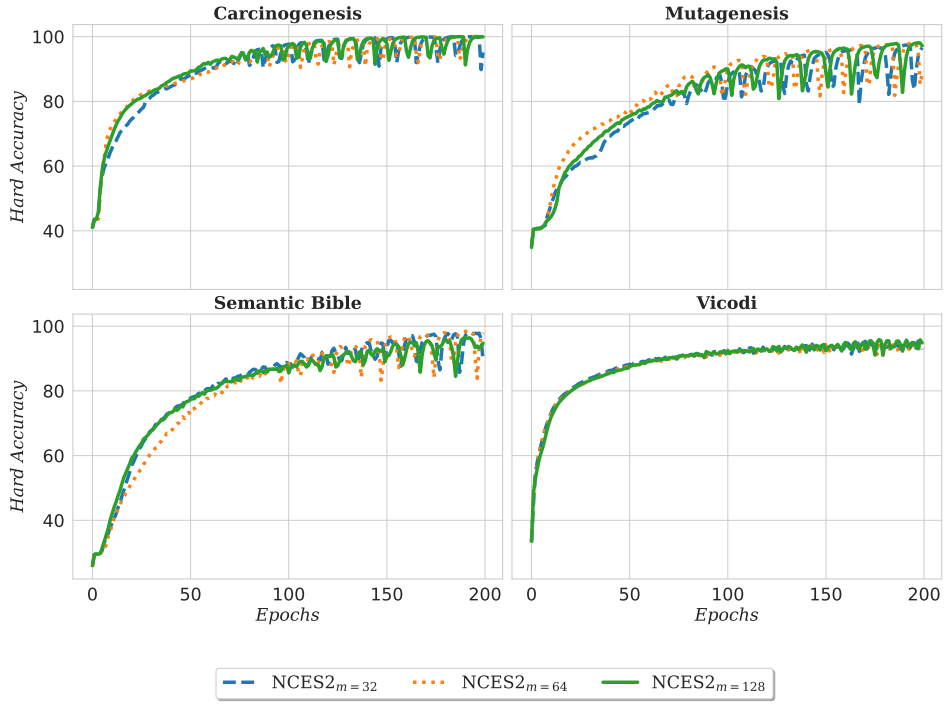


Figure 6.3: Training accuracy (in %) curves using the ConEx embedding model. m is the number of inducing points.

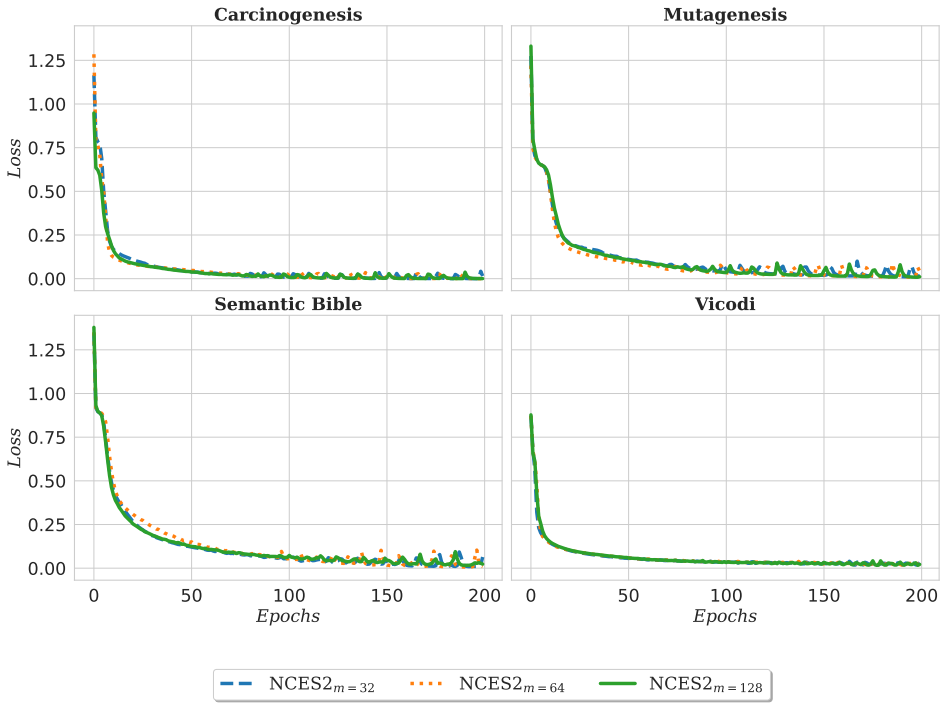


Figure 6.4: Loss curves. m is the number of inducing points.

Comparison to the State of the Art

Search-based Approaches. We ran extensive experiments comparing NCES2 to the search-based approaches EvoLearner, CELOE, ELTL, and ECII. The results are presented in Table 6.4. As done in [82, 113], we employ a timeout of 5 minutes per approach on each learning problem. ECII and ELTL do not support a timeout configuration and were therefore executed with their default settings. On the one side, the results in Table 6.4 suggest that NCES2 significantly outperforms search-based approaches in runtime on all datasets as it synthesizes a solution in less than a second on average. The standard deviation of NCES2's prediction runtime is zero because it computes solutions for all learning problems at the same time as a single forward pass of a batch of inputs. This leads to the same prediction time for all learning problems and therefore a zero standard deviation. We employed the Wilcoxon Rank Sum Test to check for performance difference significance. The significance level is 5% and the null hypothesis that the compared quantities share the same distribution. We also achieve better performance in terms of F-measure on large datasets (Carcinogenesis and Vicodi) while remaining the second best on the Mutagenesis dataset with NCES2$_{m=\{32,128\}}$ behind EvoLearner. Meanwhile, we observe a poor performance on the Semantic Bible dataset with an average F-measure of 0.79 (NCES2$_{m=\{32,64\}}$) compared to 0.89 for CELOE. We attribute this to the data hunger of deep learning models since Semantic Bible is the smallest dataset with only 724 individuals and 48 atomic classes (cf. Table 6.1).

Table 6.4: Evaluation results per dataset. The star (∗) indicates statistically significant differences between the best search-based and the best synthesis-based approaches. Underlined values are the second best. ↑ indicates that the higher the better, and ↓ indicates the opposite. Here, NCES2 uses the embedding model ConEx.

	F_1 ↑			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
CELOE	0.29±0.39	0.74±0.38	**0.89±0.20**∗	0.23±0.35
EvoLearner	0.89±0.16	**0.95±0.08**	0.88±0.13	0.77±0.26
ELTL	0.14±0.28	0.36±0.35	0.35±0.32	0.09±0.23
ECII	0.19±0.31	0.34±0.32	0.33±0.32	0.29±0.31
NCES2$_{m=32}$	0.84±0.33	0.77±0.39	0.71±0.34	0.82±0.32
NCES2$_{m=64}$	0.84±0.33	0.78±0.37	0.72±0.34	0.83±0.31
NCES2$_{m=128}$	0.86±0.33	0.73±0.38	0.70±0.36	0.83±0.31
NCES2$_{m=\{32,64\}}$	0.92±0.25	0.83±0.34	0.79±0.32	0.91±0.24
NCES2$_{m=\{32,128\}}$	0.91±0.27	0.86±0.32	0.78±0.35	0.88±0.26
NCES2$_{m=\{64,128\}}$	**0.93±0.24**∗	0.84±0.32	0.79±0.33	0.86±0.29
NCES2$_{m=\{32,64,128\}}$	0.91±0.25	0.85±0.32	0.77±0.35	**0.91±0.24**∗
	Accuracy ↑			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
CELOE	0.63±0.23	0.87±0.18	0.96± 0.09	0.79±0.16
EvoLearner	**1.00± 0.01**∗	**1.00± 0.02**∗	**0.97± 0.05**	**0.98± 0.07**∗
ELTL	0.19±0.32	0.41±0.35	0.39±0.30	0.42±0.44
ECII	0.27±0.38	0.33±0.33	0.29±0.33	0.71±0.39
NCES2$_{m=32}$	0.94±0.20	0.89±0.25	0.88±0.20	0.94±0.21
NCES2$_{m=64}$	0.95±0.17	0.89±0.26	0.86±0.24	0.95±0.18
NCES2$_{m=128}$	0.93±0.22	0.89±0.24	0.89±0.21	0.96±0.16
NCES2$_{m=\{32,64\}}$	0.96±0.18	0.91±0.25	0.88±0.25	0.96±0.17
NCES2$_{m=\{32,128\}}$	0.95±0.17	0.92±0.23	0.91±0.20	0.96±0.17
NCES2$_{m=\{64,128\}}$	0.96±0.17	0.91±0.24	0.90±0.22	0.96±0.19
NCES2$_{m=\{32,64,128\}}$	0.95±0.18	0.91±0.24	0.89±0.24	0.96±0.17
	Runtime (sec.) ↓			
	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
CELOE	268.90±116.04	165.27±145.11	172.04±140.27	334.99±43.87
EvoLearner	62.21±26.11	70.77±47.53	18.44±5.53	236.92±80.90
ELTL	26.15±2.11	15.83±16.56	4.73±0.98	335.90±205.39
ECII	25.62±6.11	20.40±4.00	6.73±1.67	37.12±25.12
NCES2$_{m=32}$	**0.02±0.00**∗	**0.02±0.00**∗	**0.01±0.00**∗	**0.03±0.00**∗
NCES2$_{m=64}$	0.03±0.00	0.03±0.00	0.01±0.00	0.03±0.00
NCES2$_{m=128}$	0.03±0.00	0.03±0.00	0.02±0.00	0.04±0.00
NCES2$_{m=\{32,64\}}$	0.05±0.00	0.05±0.00	0.03±0.00	0.06±0.00
NCES2$_{m=\{32,128\}}$	0.06±0.00	0.06±0.00	0.03±0.00	0.06±0.00
NCES2$_{m=\{64,128\}}$	0.06±0.00	0.06±0.00	0.03±0.00	0.07±0.00
NCES2$_{m=\{32,64,128\}}$	0.09±0.00	0.09±0.00	0.05±0.00	0.10±0.00

NCES. To quantify the main differences between NCES2 and previous NCES approaches, we compare them on the test sets (the 100 unseen learning problems on each knowledge base). Some of these learning problems have solutions in $\mathcal{ALC}$ while others can only be solved in $\mathcal{ALCHIQ}^{(\mathcal{D})}$. NCES2$^{✂}$ and NCES2$^{\mathcal{ALC}}$ are ablations of NCES2. The first ablation corresponds to NCES2 without the improved data generator. The second ablation corresponds to NCES2 trained on the same data as NCES, i.e., data in $\mathcal{ALC}$ on which we apply the improved data generator. Both approaches use the ConEx embedding model, and the Set-Transformer architecture with 32 inducing points as the synthesizer. The results

Table 6.5: Comparison of NCES2 and NCES on test datasets

	F_1			
	NCES2	NCES2$^{✂}$	NCES2$^{\mathcal{ALC}}$	NCES
Carcinogenesis	**0.84 ± 0.33**∗	0.79 ± 0.37	0.71 ± 0.38	0.68 ± 0.41
Mutagenesis	**0.77 ± 0.39**∗	0.53 ± 0.44	0.53 ± 0.44	0.68 ± 0.42
Semantic Bible	**0.71 ± 0.34**∗	0.66 ± 0.37	0.64 ± 0.37	0.64 ± 0.36
Vicodi	**0.82 ± 0.32**∗	0.76 ± 0.35	0.53 ± 0.41	0.50 ± 0.44

given by the two approaches are reported in Table 6.5. From the table, we can observe that NCES2 significantly outperforms NCES on all datasets with an absolute difference of up to 0.32 F-measure on the Vicodi dataset. These large differences in performance show the superiority of NCES2 over NCES; in particular, they reveal the impact of the improved training data generator and the expressiveness of $\mathcal{ALCHIQ}^{(\mathcal{D})}$ as an ablation of any of these leads to a decrease in performance, see the results achieved by NCES2$^{✂}$ and NCES2$^{\mathcal{ALC}}$ in Table 6.5. Nevertheless, the two approaches have comparable prediction time. NCES2 should therefore be preferred over NCES on most class expression learning tasks.

Table 6.6 presents the predictive performance of the three best approaches NCES2, EvoLearner, and CELOE on the learning problems introduced in Section 6.1.1. NCES2 outperforms its competitors in F-measure on LP_2 and LP_4 as it computes exact solutions for these learning problems. CELOE fails to find suitable solutions and achieves 0.01 F-measure on both LP_2 and LP_4. EvoLearner computes approximate solutions with 0.83 and 0.93 F-measure for LP_2 and LP_4, respectively. All three approaches achieve comparable performance on LP_1 and LP_3.

The effectiveness of NCES2 is demonstrated by its ability to compute (i.e. synthesize) expressions it has never seen during training, e.g., LP_2 and LP_4. We hence believe that NCES2 should serve as a robust alternative to search-based approaches such as CELOE on especially large knowledge bases where the latter are prohibitively slow.

6.2.3 Real-world Problems

We applied NCES2 to real-world learning problems from 3 different datasets: Carcinogenesis (1 learning problem), Mutagenesis (1 learning problem), Family (7 learning problems). Learning problems on Carcinogenesis and Mutagenesis concern the description of chemical compounds that cause cancer and gene mutation, respectively. On the Family dataset, we aim to learn the concepts Uncle, Aunt, Cousin, Great-grandfather, Great-grandmother, Great-granddaughter, and Great-grandson from examples. Solutions to the given learning problems do not appear in the background knowledge nor in the training datasets.

In Table 6.7, we report the results obtained on the above learning problems. We conducted experiments in a 10-fold cross-validation setting. Each of the ten folds contains some part of the positive and negative examples. For each run, NCES2 solves a learning problem

Table 6.6: Solution per approach for learning problems LP_1, LP_2, LP_3, and LP_4 presented in Section 6.1.1. We consider the three best approaches NCES2, CELOE, and EvoLearner.

	Prediction	F_1
	CELOE	
LP_1	Man	0.99
LP_2	¬Bond	0.01
LP_3	Bond ⊔ (Atom ⊓ (¬Carbon-25))	0.99
LP_4	Flavour ⊔ (¬War)	0.01
	EvoLearner	
LP_1	Man	0.99
LP_2	Sulfur-74 ⊔ (∃ drosophila_slrl.{True})	83.33
LP_3	Atom ⊔ (∃ inBond.Atom)	0.99
LP_4	Intellectual-Construct	0.93
	NCES2	
LP_1	Man	0.99
LP_2	Fluorine-92 ⊔ Sulfur-74 ⊔ (∃ drosophila_rt.{False})	1.00
LP_3	(Atom ⊓ (Oxygen-45 ⊔ (¬Oxygen))) ⊔ (∃ inBond.(¬Carbon-10))	0.97
LP_4	Measurable-Trend ⊔ (∃ related.(Idea ⊔ Uprising))	1.00

Table 6.7: 10-fold cross-validation results on real-world problems. "–" denotes unknown learning problems.

	Train-F_1	*Test-F_1*	*Runtime (sec.)*
Family			
Aunt	0.80	0.81	1.14
Cousin	0.70	0.68	1.18
Great-granddaughter	1.00	1.00	0.96
Great-grandfather	0.94	0.95	0.94
Great-grandmother	0.94	0.93	1.32
Great-grandson	0.92	0.93	1.17
Uncle	0.89	0.89	0.95
Carcinogenesis			
–	0.71	0.70	1.91
Mutagenesis			
–	0.70	0.70	1.84

(100 predictions per problem) using 9 folds. The 10-th fold is used for evaluation. We report average results across the 10 folds. From the results, we can observe that NCES2 maintains a high predictive accuracy while being scalable[4] (≈1 second per problem on average). Learning problems on Carcinogenesis and Mutagenesis appear to be harder than those on Family. This explains the relatively low performance achieved by NCES2 on these problems.

[4]Note that for each run, we perform 100 predictions per problem and select the class expression with the highest F_1 score. Hence, the runtime per prediction is ≈0.01 second on average.

6.3 Conclusion

In this chapter, we presented NCES2, a new instance of synthesis-based approaches for class expression learning that operates within the description logic $\mathcal{ALCHIQ}^{(\mathcal{D})}$. NCES2 also features a novel data generation technique which constructs multiple sets of examples per learning problem in the training set, enhancing generalization to unseen learning problems at inference time. Moreover, it integrates an embedding model which is trained end-to-end with the neural synthesizer to alleviate dependence on pretrained embeddings for input examples. Experimental results suggest that both the new data generation technique and the increased expressiveness of the underlying description logic ($\mathcal{ALCHIQ}^{(\mathcal{D})}$) contribute the most to NCES2's superiority over earlier NCES instances. In the next chapter, we present our latest research contribution which introduces a reformulation of the classical class expression learning problem to make learning systems more robust to changes in input examples.

7 Robust Class Expression Synthesis via Iterative Sampling

Preamble. This chapter is based on Kouagou et al. [111] and addresses the fifth research question (see Section 1.2.4).

7.1 Methodology

We first give a modification of Definition 4.1 that encourages learning systems to compute solutions without using all the provided examples but rather the most informative ones. Second, we establish the connections between classical solutions and those from our new formulation. Finally, we describe our learning algorithm in detail and discuss potential limitations.

7.1.1 Generalized Learning Problem

Our proposed definition below is motivated by the fact that in real-world applications of CEL, one deals with small sets of examples (see an example in the introduction). In this context, a good learning system should be able to compute a solution that is as specific as possible but general enough to cover other relevant examples that are not given. In the next paragraphs, we denote the set of all solutions to the classical learning problem $\mathcal{CLP}(\mathcal{K}, T, E^+, E^-)$ by $\mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, E^+, E^-)$.

Definition 7.1 (Generalized Learning Problem ($\mathcal{GLP}$)). *Given a knowledge base $\mathcal{K}$, a target concept T, and sets of positive/negative examples $E^+ = \{e_1^+, e_2^+, \ldots, e_{n_1}^+\}$ and $E^- = \{e_1^-, e_2^-, \ldots, e_{n_2}^-\}$, the learning problem is to find non-empty subsets $\mathcal{E}^+ \subseteq E^+$, $\mathcal{E}^- subseteq E^-$ with the following properties*

1. $\mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, \mathcal{E}^+, \mathcal{E}^-) \neq \emptyset$
2. $\mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, \mathcal{E}^+, \mathcal{E}^-) \subseteq \mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, E^+, E^-)$
3. *There do not exist non-empty subsets $\mathcal{E}'^+ subseteq E^+$, $\mathcal{E}'^- \subseteq E^-$ such that $|\mathcal{E}'^+|+|\mathcal{E}'^-| < |\mathcal{E}^+| + |\mathcal{E}^-|$ and $\emptyset \neq \mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, \mathcal{E}'^+, \mathcal{E}'^-) \subseteq \mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, E^+, E^-)$,*

where $|.|$ denotes the cardinality of a set. Properties *1.* and *2.* in the above definition require that the set of all solutions (class expressions) to the learning problem $\mathcal{CLP}(\mathcal{K}, T, \mathcal{E}^+, \mathcal{E}^-)$ is non-empty and each of its elements is a solution to $\mathcal{CLP}(\mathcal{K}, T, E^+, E^-)$. Property *3.* further ensures that such subsets $\mathcal{E}^-$, $\mathcal{E}^+$ are minimal in size. We say that a solution $(\mathcal{E}^+, \mathcal{E}^-)$ to $\mathcal{GLP}$ is *ideal* if $\mathcal{E}^+ \neq E^+$ or $\mathcal{E}^- \neq E^-$. In such a case, $\mathcal{E}^+$ and $\mathcal{E}^-$ are called *core example sets*.

Theorem 7.2. *$\mathcal{GLP}$ has a solution if and only if $\mathcal{CLP}$ has one.*

To prove Theorem 7.2, we define $\Gamma(\mathcal{E}^+, \mathcal{E}^-, \mathcal{E}'^+, \mathcal{E}'^-)$ to be the following logical expression which we call the Γ condition:

$$\begin{aligned}\big[|\mathcal{E}'^+| + |\mathcal{E}'^-| < |\mathcal{E}^+| + |\mathcal{E}^-|\big] \wedge \\ \big[\mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, \mathcal{E}'^+, \mathcal{E}'^-) \neq \emptyset\big] \wedge \\ \big[\mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, \mathcal{E}'^+, \mathcal{E}'^-) \subseteq \mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, E^+, E^-)\big]\end{aligned}$$

for any non-empty sets $\mathcal{E}^+, \mathcal{E}^-, \mathcal{E}'^+$, and $\mathcal{E}'^-$. Here, $\wedge$ is the "logical and" operator.

Proof. First, assume that $\mathcal{GLP}$ has a solution. Then (by definition), there exist non-empty subsets $\mathcal{E}^+ \subseteq E^+$, $\mathcal{E}^- \subseteq E^-$ such that $\mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, \mathcal{E}^+, \mathcal{E}^-) \neq \emptyset$ and $\mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, \mathcal{E}^+, \mathcal{E}^-) \subseteq \mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, E^+, E^-)$. Let C be an arbitrary element in $\mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, \mathcal{E}^+, \mathcal{E}^-)$. Then, $C \in \mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, E^+, E^-)$ and therefore C is a solution to $\mathcal{CLP}$.

It remains to prove that if $\mathcal{CLP}$ has a solution, then $\mathcal{GLP}$ also has one. Assume that C_0 is a solution to $\mathcal{CLP}$, and let $\mathcal{E}_0^+ = E^+$ and $\mathcal{E}_0^- = E^-$. Then, $C_0 \in \mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, \mathcal{E}_0^+, \mathcal{E}_0^-)$ and $\mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, \mathcal{E}_0^+, \mathcal{E}_0^-) \subseteq \mathcal{S}_{\mathcal{CLP}}(\mathcal{K}, T, E^+, E^-)$ (i.e., properties *1.* and *2.* are satisfied by $(\mathcal{E}_0^+, \mathcal{E}_0^-)$). If there do not exist subsets $\mathcal{E}_1^+$, $\mathcal{E}_1^-$ such that $\emptyset \neq \mathcal{E}_1^+ \subseteq E^+$, $\emptyset \neq \mathcal{E}_1^- \subseteq E^-$, and $\Gamma(\mathcal{E}_0^+, \mathcal{E}_0^-, \mathcal{E}_1^+, \mathcal{E}_1^-)$ holds (property *3.*), then $(\mathcal{E}_0^+, \mathcal{E}_0^-)$ is a solution to $\mathcal{GLP}$. If such $\mathcal{E}_1^+, \mathcal{E}_1^-$ exist, define $S_n = (\mathcal{E}_n^+, \mathcal{E}_n^-)$ and $\Sigma_n = |\mathcal{E}_n^+| + |\mathcal{E}_n^-|$ for any integer $n \geq 1$ such that $\Gamma(\mathcal{E}_{n-1}^+, \mathcal{E}_{n-1}^-, \mathcal{E}_n^+, \mathcal{E}_n^-)$ holds. Let $S = \{S_n\}_{n \geq 1}$ and $\Sigma = \{\Sigma_n\}_{n \geq 1}$. Then, S and Σ are non-empty sets since $(\mathcal{E}_1^+, \mathcal{E}_1^-) \in S$ and $|\mathcal{E}_1^+| + |\mathcal{E}_1^-| \in \Sigma$. The mapping $f : S \to \Sigma : S_n \longmapsto \Sigma_n$ is clearly a bijection (see proof of Lemma). Moreover, Σ_n is a strictly decreasing sequence of integer values due to the fact that for any integers $n_1 > n_2 \geq 1$ such that S_{n_1} and S_{n_2} exist, we have $|\mathcal{E}_{n_1}^+| + |\mathcal{E}_{n_1}^-| < |\mathcal{E}_{n_2}^+| + |\mathcal{E}_{n_2}^-|$ (i.e., $\Sigma_{n_1} < \Sigma_{n_2}$) through the Γ condition. Given that subsets $\mathcal{E}_n^+$ and $\mathcal{E}_n^-$ are required not to be empty, we also have $\Sigma_n \geq 2$ for all $n \geq 1$ (at least one positive example and one negative example). The set Σ is therefore finite and admits a minimum Σ_{n^*}. Hence, $f^{-1}(\Sigma_{n^*}) \in S$ is a solution to $\mathcal{GLP}$ as it satisfies all the properties in Definition 7.1; this completes the proof. □

Lemma 7.3. *The mapping $f : S \to \Sigma : S_n \longmapsto \Sigma_n$ defined in the proof of Theorem 7.2 is a bijection.*

Proof. We first prove that f is a surjective function. Let $e \in \Sigma$. Then, there exists $n \geq 1$ such that $\Sigma_n = e$ (by definition of Σ). Consequently, $S_n \in S$ (note the same subscript n as Σ_n) and we have $f(S_n) = \Sigma_n$. Hence, f is surjective.

We now prove that f is one to one. Let $s_1, s_2 \in S$ such that $s_1 \neq s_2$. By definition of S, there exist $n_1 \geq 1$ and $n_2 \geq 1$ such that $s_1 = S_{n_1}$ and $s_2 = S_{n_2}$. Without loss of generality we can assume that $n_1 > n_2$. Then, $|\mathcal{E}^+_{n_1}| + |\mathcal{E}^-_{n_1}| < |\mathcal{E}^+_{n_2}| + |\mathcal{E}^-_{n_2}|$ following the Γ condition. In other words, $\Sigma_{n_1} < \Sigma_{n_2}$ and hence $\Sigma_{n_1} \neq \Sigma_{n_2}$. f is therefore one to one. □

Theorem 7.2 provides sufficient and necessary conditions for our formulated learning problem ($\mathcal{GLP}$) to have a solution. In particular, we have shown (see proof of the theorem) how ideal solutions to $\mathcal{GLP}$, i.e., $f^{-1}(\Sigma_{n^*})$ with $n > 0$, can be sought starting from an arbitrary solution to $\mathcal{CLP}$. Given the sequential nature and complexity of this search process—see for example the use of existential quantifiers which often require search over the complete space—more efficient search strategies are necessary. In the following subsection, we propose an iterative stochastic search method coupled with gradient-based optimization to construct an approach that approximates solutions to $\mathcal{GLP}$.

7.1.2 Learning Algorithm

Our approach uses the ConEx embedding model [45] to obtain embeddings for input examples, and the Set-Transformer architecture [118] to synthesize class expressions from embeddings. The choice of the ConEx embedding model is motivated by the fact that 1.) it can model both symmetric and anti-symmetric relations which are likely to be encountered on heterogeneous knowledge graphs, and 2.) it employs convolution operations for efficient embedding computation. The embedding model and the Set-Transformer instance are fused into a learner f_Θ and trained jointly. Figure 7.1 depicts the training procedure of the learner f_θ. At each iteration, our approach ROCES subsamples positive and negative examples for a given learning problem as inputs to the learner f_θ which then aims to predict the target class expression. For example, at the first iteration, "`Peng_0`" is selected as positive example, while "`Eagle_0`" and "`Bob`" are selected as negative examples. At the second iteration, different subsets[1] of examples are selected. The selection of specific positive or negative examples is based on a probability distribution which is skewed towards small subsets of examples; refer to Algorithm 14 for further details.

Description of Algorithm 14. In lines 1 and 2, we construct possible sizes S^+ and S^- for the subsets $\mathcal{E}^+$ and $\mathcal{E}^-$ (see Definition 7.1), respectively. Next, we define discrete probability functions[2] p^+ and p^- over S^+ and S^- (lines 3 and 4). In lines 5-8, we sample

[1] Positive and negative example sets may not be of equal size in a given iteration.

[2] The density functions p^+ and p^- are defined in such a way that higher probabilities are given to smaller values. In this way, we encourage the learner f_Θ to learn target expressions with small sets of examples. Of course, the search for the most suitable probability functions remains an interesting topic which we leave for future work.

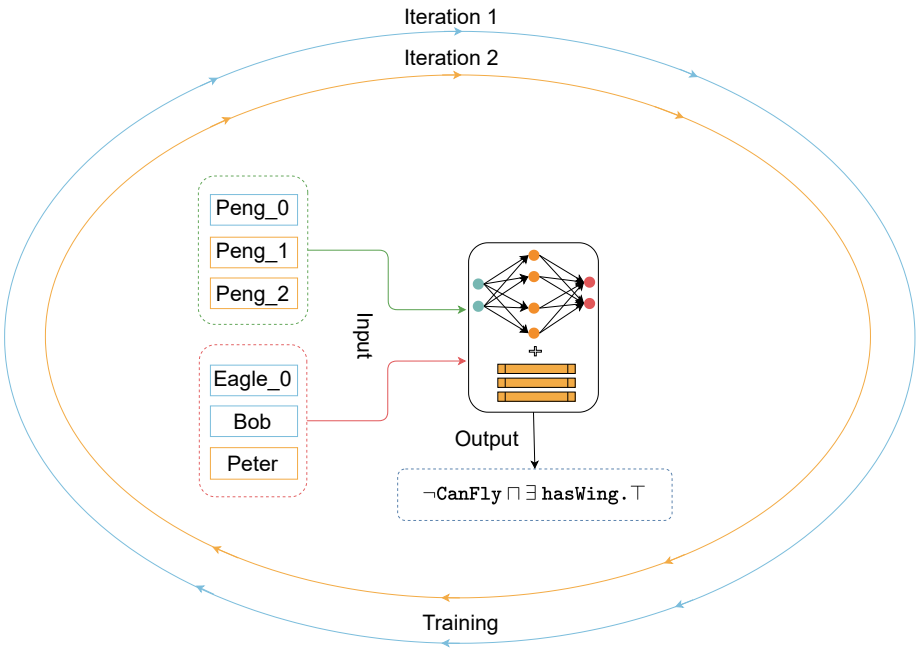


Figure 7.1: Overview of our proposed approach ROCES

candidate example set sizes k^+ and k^- following p^+ and p^-, and construct candidate subsets of examples $\mathcal{E}^+$ and $\mathcal{E}^-$ by uniformly sampling k^+ positive and k^- negative examples, respectively. Finally, the parameterized learner f_Θ synthesizes an expression $\hat{C}$ which we compare to the target C, compute the loss and back-propagate to update the parameters Θ (lines 9-11). Note that Algorithm 14 describes a single learning step with one training data point for the sake of simplicity. In practice (e.g. in our experiments), a batch of training data points is given and parameter updates are performed based on the input batch. Once the learner f_Θ is trained using Algorithm 14, it can be employed to solve learning problems with arbitrary example set sizes; we provide more details in Section 7.2. In the rest of the paper, we refer to the parameterized learner f_Θ trained using Algorithm 14 as our approach ROCES. While ROCES performs well on learning problems with arbitrary numbers of examples (as we will see in Section 7.2), some limitations regarding theoretical guaranties are worth noting.

Limitations of Algorithm 14. As mentioned earlier, our learning algorithm computes approximate solutions to $\mathcal{GLCP}$. The search for the exact solution can require up to $2^{|E^+|} \times 2^{|E^-|}$ checks (for $|E^+| = |E^-| = 100$, this is approximately 10^{60} checks). Hence, the optimal subsets of examples $(\mathcal{E}^+, \mathcal{E}^-)$ computed in Algorithm 14 might fail to satisfy property *3.* (minimality) in Definition 7.1 but they remain, in most cases, strictly smaller than the initial sets E^+ and E^- as suggested by the results in Table 7.3

Algorithm 14 Learning Step

Input: (C, E^+, E^-), f_Θ
hyperparameters: k, Opt (optimization algorithm)
Output: f_Θ

$S^+ \leftarrow [\min(k, |E^+|), 2k, 3k, \ldots, |E^+|]$
$S^- \leftarrow [\min(k, |E^-|), 2k, 3k, \ldots, |E^-|]$
 # Define probability functions over S^+ and S^-
$\forall\, x \in S^+,\ p^+(x) = \dfrac{1/[x]_{S^+}}{\sum_{s \in S^+}(1/[s]_{S^+})}$
$\forall\, x \in S^-,\ p^-(x) = \dfrac{1/[x]_{S^-}}{\sum_{s \in S^-}(1/[s]_{S^-})}$
Draw k^+ from S^+ following p^+
Draw k^- from S^- following p^-
$\mathcal{E}^+ \leftarrow \mathcal{U}(E^+, k^+)$ *# Uniform sampling*
$\mathcal{E}^- \leftarrow \mathcal{U}(E^-, k^-)$
$\hat{C} = f_\Theta(\mathcal{E}^+, \mathcal{E}^-)$ *# Synthesize a class expression*
$\mathcal{L} = \text{Loss}(\hat{C}, C)$ *# Compute the loss*
$\Theta \leftarrow Opt(\Theta, \nabla_\Theta \mathcal{L})$ *# Update the parameters Θ*
return f_Θ

7.2 Experiments

In this section, we evaluate our proposed approach. We start with experimental settings before presenting and discussing the obtained results.

7.2.1 Experimental Setup

Datasets

We used four benchmark datasets in our experiments: Semantic Bible[3], Vicodi [152], Carcinogenesis [224], and Mutagenesis [224]. Vicodi describes the European history, and Semantic Bible, the New Testament. Carcinogenesis and Mutagenesis describe chemical compounds and how they relate to each other. On the last two datasets, CEL can provide insights into hidden properties shared by different compounds and facilitate further investigations by domain experts. For instance, a class expression learner in [27] found that a chemical compound is carcinogenic if it can be described by the expression: $\neg(\exists$ `hasAtom.(Nitrogen-35` $\sqcup$ `Phosphorus-60` $\sqcup$ `Phosphorus-61` $\sqcup$ `Titanium-134`$)) \sqcap (\geq 3$ `hasStructure.(Halide` $\sqcap$ $\neg$`Halide10)` $\sqcup \exists$ `amesTestPositive.{True}` $\sqcap \geq 5$ `hasBond.(`$\neg$`Bond-7`$))$. In natural language, this would translate into: «A chemical compound is carcinogenic if and only if it does not contain a Nitrogen-35, Phosphorus-60, Phosphorus-61, or Titanium-134 atom and it has at least three Halide—excluding Halide10—structures or the ames test of the compound

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

is positive and there are at least five atom bonds which are not of bond type 7» (cf. [27], Section 6.1).

In our experiments, we use the training and test data constructed in NCES2 [109]. Consequently, there are 100 learning problems on the test set of each benchmark dataset; we refer to Table 6.1 for complete statistics.

Hyperparameter Configuration

Because of its efficiency, we used the random search approach [20] to find the best hyperparameter values for our approach, namely the learning rate, the training batch size N, the embedding dimension d for the embedding model ConEx, and the number of inducing points in the Set-Transformer model. As in Chapter 6, it is sufficient to search hyperparameter values on one dataset, e.g. Carcinogenesis, and use the same values on the rest of the datasets; we adopt the same approach in this work. We report hyperparameter settings in Table 7.1.

Table 7.1: Hyperparameter settings per dataset. Lr is the learning rate, d the embedding dimension in the embedding model, N the batch size, L the maximum length of expressions synthesized by ROCES, and gc is the gradient norm clipping value.

	Carcinogenesis	Mutagenesis	Semantic Bible	Vicodi
$Epochs$	400	400	400	400
Lr	0.001	0.001	0.001	0.001
d	50	50	50	50
N	512	512	512	512
L	48	48	48	48
gc	5	5	5	5

Hardware

We trained the learner f_Θ using Algorithm 14 (that is, ROCES) on a virtual machine equipped with 64 AMD EPYC 9334 32-Core Processors @3.91GHz, and a NVIDIA A100 80GB GPU. Post training, we used a server with 16 Intel Xeon E5-2695 CPUs @2.30GHz and 128GB RAM to conduct experiments on CEL where we compared our approach against CELOE, EvoLearner, and NCES2. This is because CELOE and EvoLearner do not support GPU computations, and we needed to ensure a fair comparison regarding runtimes.

7.2.2 Results and Discussion

Training Curves

ROCES was trained for 400 epochs on each dataset. The evaluation metrics used during training are the "hard accuracy" and the "soft accuracy" as defined in [113]. The hard accuracy measures how a predicted expression is similar to a target class expression in the training set in terms of their string representation. The soft accuracy is the *Jaccard index* between the set of the predicted tokens and the set of the target tokens; it does not take the order of the tokens into account. The hard accuracy is hence the most suited metric to train ROCES. We refer to [113] for more details about these metrics. In Figures 7.2,

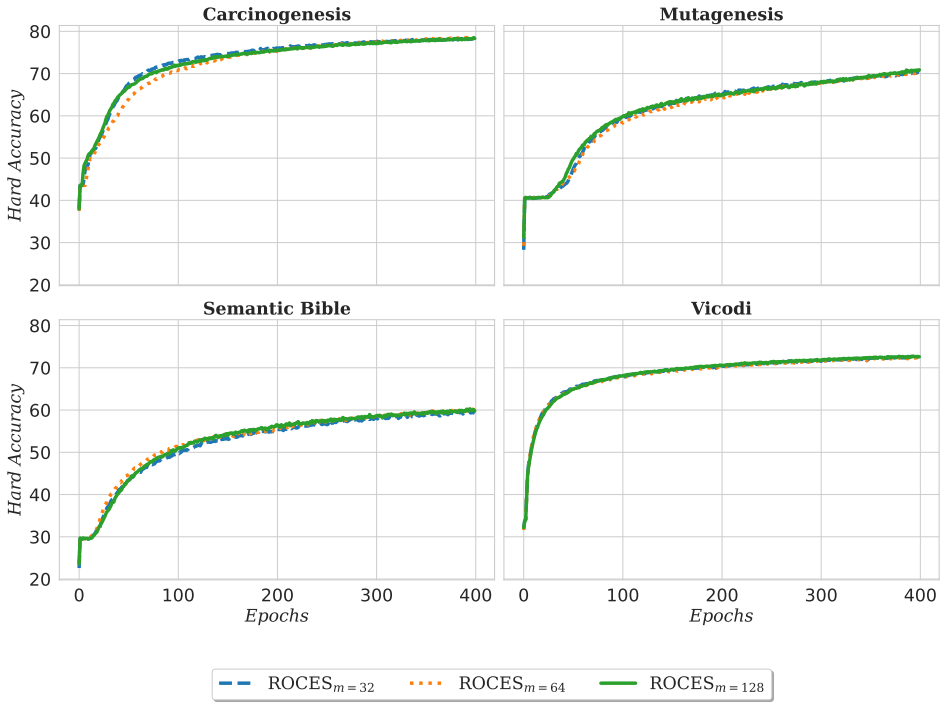


Figure 7.2: Hard accuracy curves of our proposed approach ROCES during training. The probability functions p^+, p^- defined in Algorithm 14 are used.

7.3, and 7.4 we show the training curves of ROCES. We can observe a rapid increase (in accuracy), decrease (in loss) in the early epochs and fast convergence on Carcinogenesis and Vicodi, which are the largest datasets. This observation aligns with the results presented in Table 7.2. This suggests that on large datasets, ROCES learns better mappings between sets of examples and class expressions that describe them.

Class Expression Learning on Test Datasets

ROCES remains the only approach to achieve both speed and high predictive performance on learning problems with various input example set sizes. Its superiority on large datasets

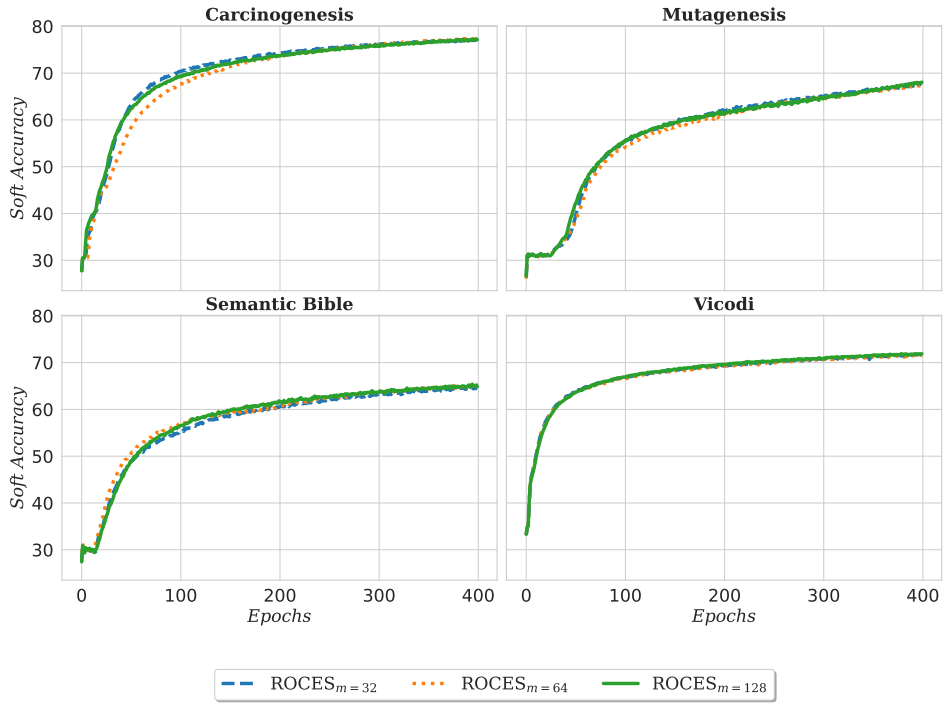


Figure 7.3: Soft accuracy curves during training

can be attributed to the fact that its deep neural network-based synthesizer performs better when it has enough training data to learn from. Meanwhile, search-based methods struggle to navigate the vast search space induced by large datasets. They perform better on small datasets (as seen in the case of Mutagenesis and Semantic Bible), where ROCES does not generalize well with the available training data. In Table 7.3, we report the frequency at which ROCES outperforms the best baseline (EvoLearner) without using the complete sets of examples. More precisely, we let ROCES explore 50 random pairs $(\mathcal{E}^+, \mathcal{E}^-)$ of subsets of examples and compute a solution for each. The results in the table suggest that with just 50 trials, ROCES outperforms EvoLearner on at least 63% (the highest being 91%) of the learning problems. The average F_1 score of the computed solutions is 90% and above on three datasets; the lowest performance being 81% and observed on Semantic Bible—see the row *Max*. Nonetheless, ROCES found the exact solution for 34% of the learning problems on this dataset. For example, it found the exact solution $\texttt{Mountain} \sqcup (\texttt{LandArea} \sqcap (\texttt{City} \sqcup (\exists\ \texttt{subregionOf}.\top)))$, while EvoLearner computed an approximate solution. We list below the solutions computed by all approaches.

1. ROCES: $\texttt{Mountain} \sqcup (\texttt{LandArea} \sqcap (\texttt{City} \sqcup (\exists\ \texttt{subregionOf}.\top)))$; F_1: 100%
2. ROCES$_U$: $\texttt{Mountain} \sqcup (\texttt{GeopoliticalArea} \sqcap (\texttt{City} \sqcup (\exists\ \texttt{subregionOf}.\top)))$; F_1: 100%
3. EvoLearner: $\texttt{City} \sqcup (\exists\ \texttt{location}.\top) \sqcup (\texttt{Mountain} \sqcap \texttt{GeographicArea}) \sqcup (\exists\ \texttt{subregionOf}.(\exists\ \texttt{subregionOf}.\texttt{GeographicArea}))$; F_1: 97.59%
4. CELOE: $\texttt{Mountain} \sqcup (\texttt{GeopoliticalArea} \sqcap \texttt{LandArea})$; F_1: 92.24%
5. NCES2: $\texttt{Mountain} \sqcup (\texttt{GeopoliticalArea} \sqcap (\texttt{City} \sqcup (\exists\ \texttt{subregionOf}.\top)))$; F_1: 100%.

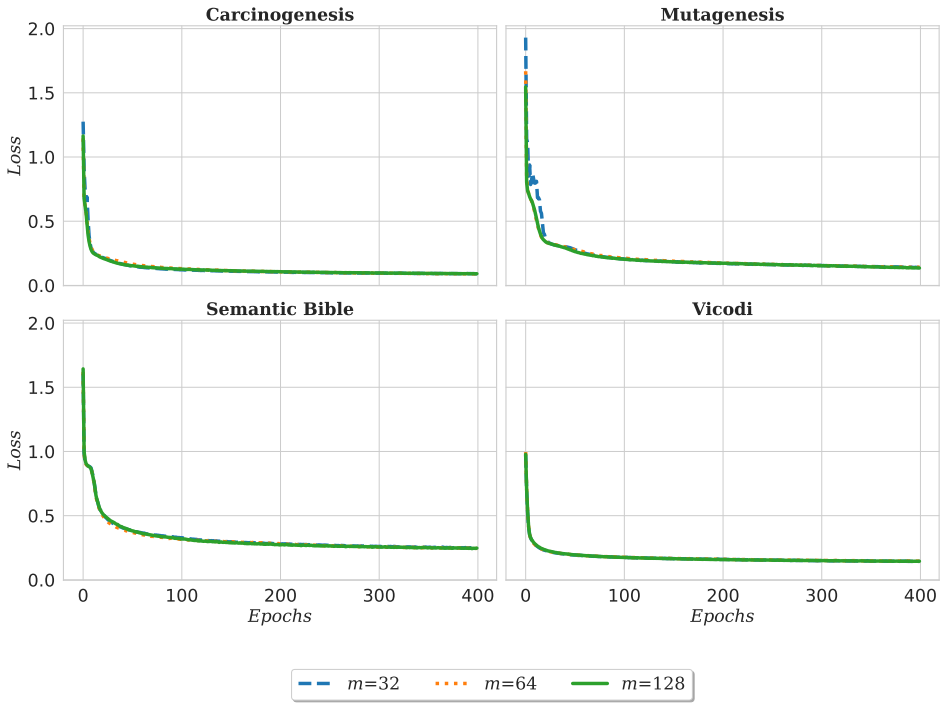


Figure 7.4: Loss (cross entropy) during training

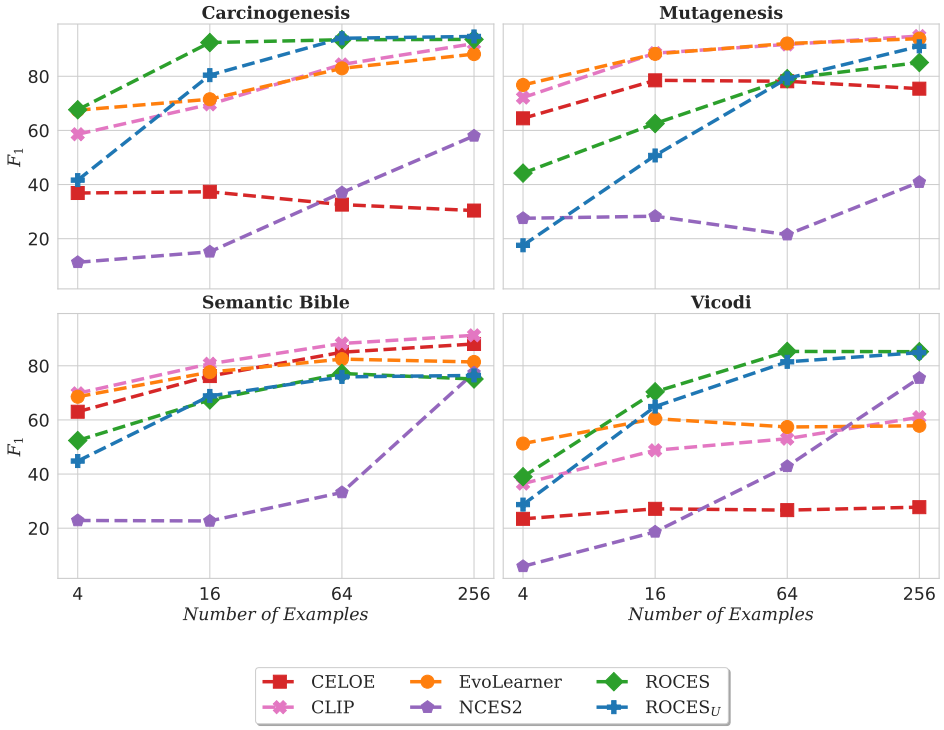


Figure 7.5: Average F_1 score of the computed solutions for different values of the input examples' set sizes

Table 7.2: Comparison of different approaches on learning problems with various input example set sizes (k). ROCES$_U$ is ROCES with uniform distributions p^+ and p^-. Bold (resp., underlined) values represent the best (resp., second best).

	Carcinogenesis			Mutagenesis		
k	16	64	Full	16	64	Full
			$F_1 \uparrow$			
CELOE	0.37	0.33	0.29	0.79	0.78	0.74
CLIP	0.70	0.84	**0.97**$*$	**0.89**$*$	<u>0.92</u>	**0.96**$*$
EvoLearner	0.72	0.83	0.89	<u>0.88</u>	**0.92**$*$	<u>0.95</u>
NCES2	0.15	0.37	0.91	0.28	0.22	0.85
ROCES (ours)	**0.92**$*$	<u>0.93</u>	0.94	0.63	0.79	0.90
ROCES$_U$ (ours)	<u>0.80</u>	**0.94**$*$	<u>0.94</u>	0.51	0.79	0.92
			Runtime (sec.) $\downarrow$			
CELOE	217.08	240.93	268.90	67.45	124.69	165.27
CLIP	201.32	200.29	33.00	197.93	205.42	107.19
EvoLearner	40.96	49.42	89.34	27.65	48.51	70.77
NCES2	0.01	0.01	0.09	0.01	0.01	0.09
ROCES (ours)	**0.01**	**0.01**	**0.07**	**0.01**	**0.01**	<u>0.06</u>
ROCES$_U$ (ours)	<u>0.01</u>	<u>0.01</u>	<u>0.07</u>	<u>0.01</u>	<u>0.01</u>	**0.05**

	Semantic Bible			Vicodi		
k	16	64	Full	16	64	Full
			$F_1 \uparrow$			
CELOE	0.76	<u>0.85</u>	<u>0.89</u>	0.27	0.27	0.23
CLIP	**0.81**$*$	**0.88**$*$	**0.92**$*$	0.49	0.53	0.69
EvoLearner	<u>0.78</u>	0.82	0.88	0.60	0.57	0.77
NCES2	0.23	0.33	0.77	0.19	0.43	**0.91**$*$
ROCES (ours)	0.67	0.77	0.75	**0.70**$*$	**0.85**$*$	0.84
ROCES$_U$ (ours)	0.69	0.76	0.76	<u>0.65</u>	<u>0.81</u>	<u>0.88</u>
			Runtime (sec.) $\downarrow$			
CELOE	95.38	163.80	172.04	233.18	300.01	300.01
CLIP	188.58	186.06	188.22	300.01	249.64	151.14
EvoLearner	12.36	17.93	18.44	170.78	248.55	236.92
NCES2	0.01	0.01	0.05	0.01	0.01	0.10
ROCES (ours)	**0.01**	**0.01**	**0.03**	**0.01**	**0.01**	**0.10**
ROCES$_U$ (ours)	<u>0.01</u>	<u>0.01</u>	<u>0.03</u>	<u>0.01</u>	<u>0.01</u>	<u>0.10</u>

Although NCES2 computed an exact solution with 100% F_1 score, it uses all available examples for this purpose, and achieves a lower performance when presented with incomplete examples. This suggests that ROCES is competitive in performance even when it does not use all available examples.

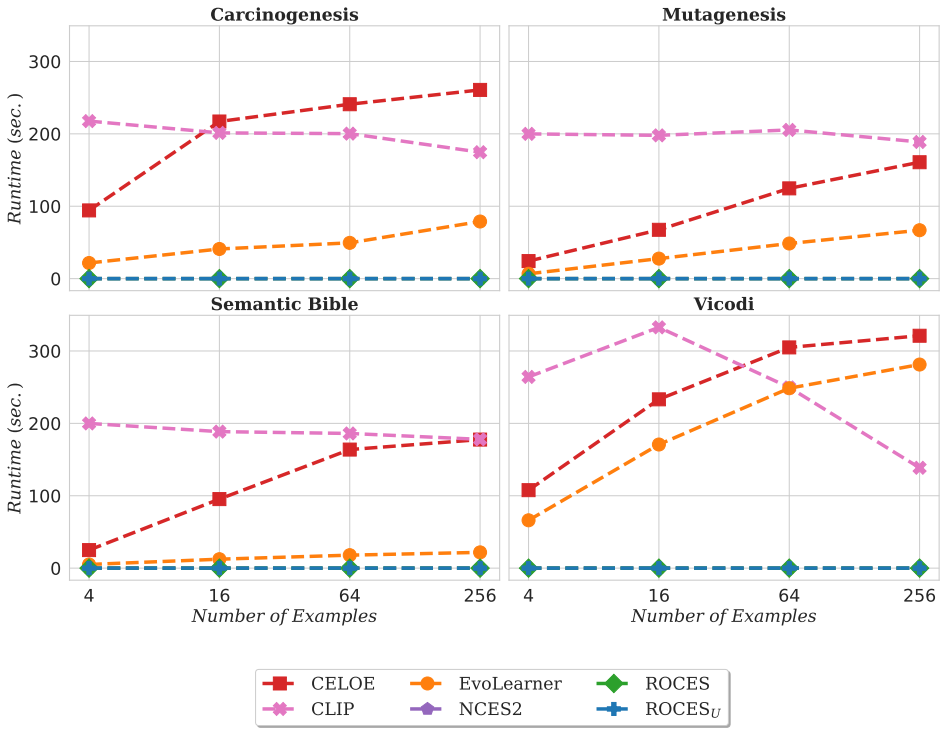


Figure 7.6: Average runtime per approach for different values of the input examples' set sizes

Few-shot Learning

Here, we let ROCES perform 50 attempts on each learning problem using only a subset of the provided examples. For each attempt, ROCES constructs a pair of example sets following Algorithm 14 (lines 1–8) and synthesizes a solution (see line 9 of Algorithm 14). From Figures 7.7 and 7.8, we can observe that most of the solutions computed by ROCES have an F-measure close to 100% (see the large area around 100). This is more noticeable when considering the best performance across the 50 trials (see the distribution for the *Max* aggregation). Again, the highest performance is observed on the largest datasets Carcinogenesis and Vicodi, but also on Mutagenesis (when ROCES is allowed multiple trials). On Semantic Bible, we observe lower F_1 scores especially on the *Min* aggregation (see the

Table 7.3: Frequency (*Freq.* in %) at which ROCES—with limited input examples—outperforms EvoLearner, and average quality (F_1) of the solutions computed by ROCES. *Min*, *Max*, and *Avg.* are aggregations accross the 50 trials.

	Carcinogenesis		Mutagenesis		Semantic Bible		Vicodi	
	Freq. (%)	F_1	*Freq.* (%)	F_1	*Freq.* (%)	F_1	*Freq.* (%)	F_1
Min	80	0.90	42	0.72	44	0.63	73	0.78
Max	91	0.95	81	0.93	63	0.81	85	0.90
Avg.	82	0.94	56	0.88	47	0.75	79	0.85

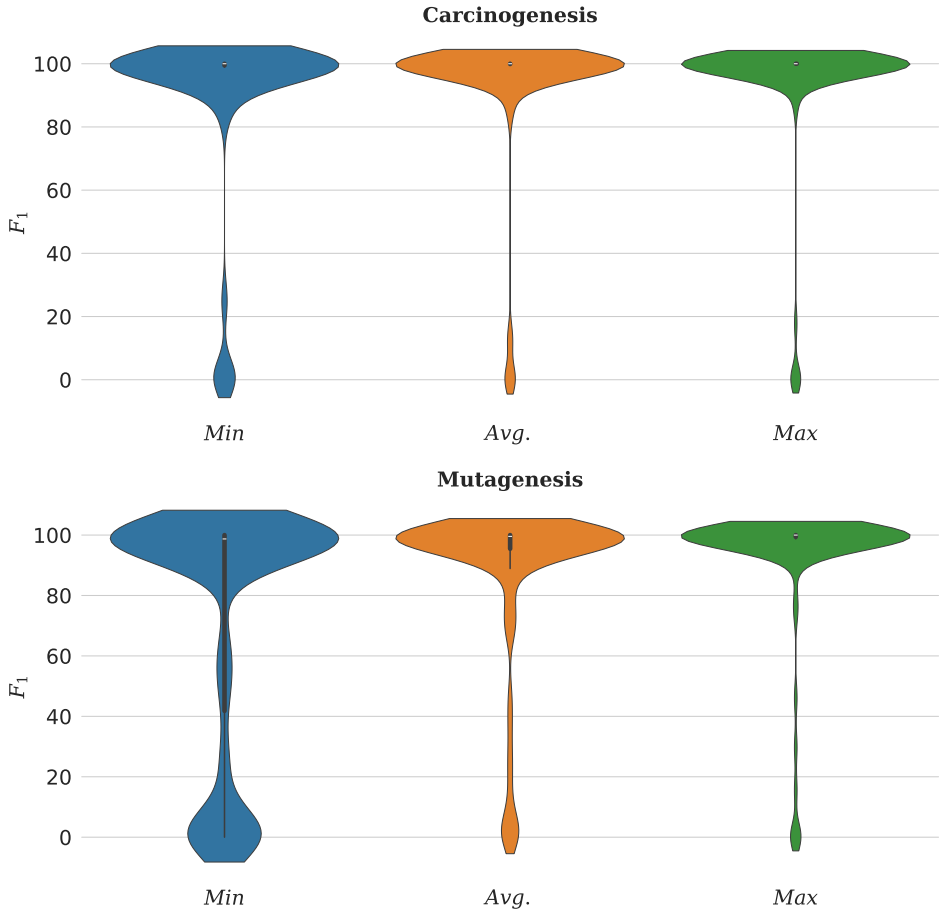


Figure 7.7: Distribution of the F-measure on Carcinogenesis and Mutagenesis. The aggregations (*Min*, *Avg.*, *Max*) are computed across 50 trials.

large area around 0 for the blue violin plot). This is again due to the fact that deep learning models perform better when there is enough high-quality training data, which is not the case for Semantic Bible (refer to Table 6.1 for dataset statistics). Nonetheless, when ROCES is allowed multiple trials, its performance significantly improves on this dataset.

Active Learning

In this evaluation setting, we measure how fast ROCES is able to retrieve a solution (or converge to its best possible prediction) when receiving feedback from an oracle. Given a learning problem whose example sets are unknown to ROCES, an oracle (in this case a description logic reasoner which knows the target class expression and its set of instances) provides the first sets of positive and negative examples. ROCES then makes an initial guess and sends its synthesized expression to the oracle which in turn evaluates the expression and provides at most k new positive and k new negative examples that are correctly classified by the predicted expression. The newly labelled examples are then added to the previous

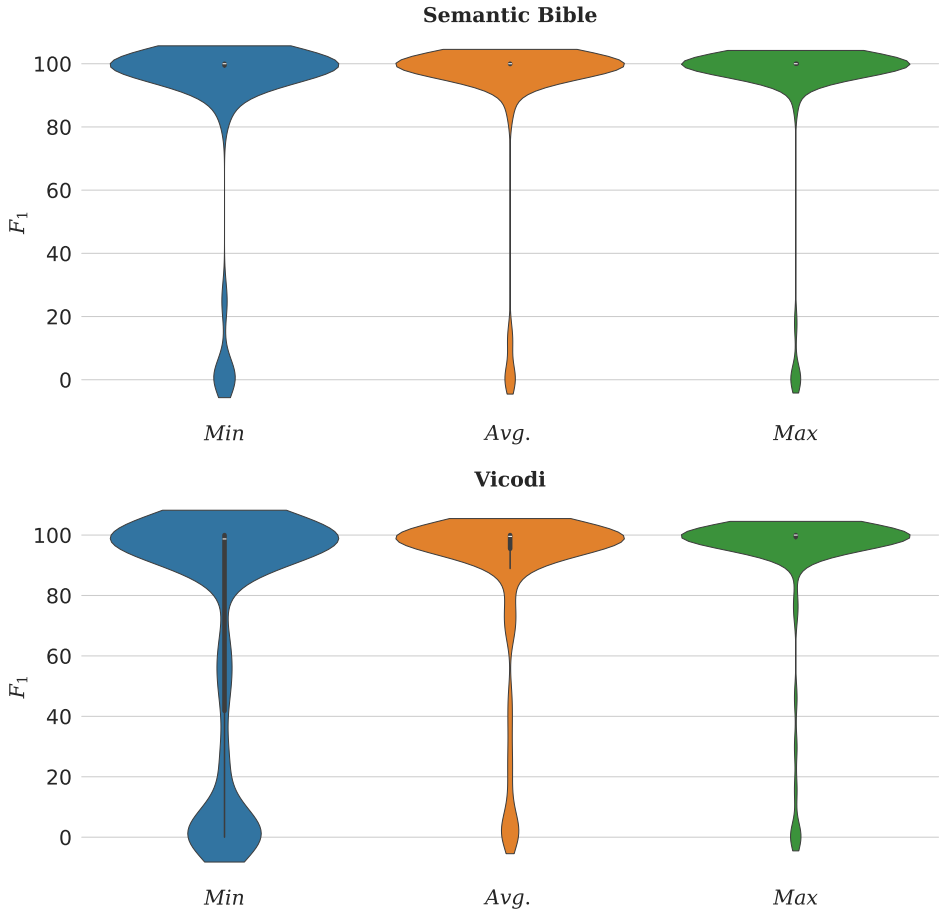


Figure 7.8: Distribution of the F-measure on Semantic Bible and Vicodi. The aggregations (*Min*, *Avg.*, *Max*) are computed across 50 trials.

sets of examples for subsequent predictions. This process continues until a solution is found or until the maximum number of iterations is reached. To validate the effectiveness of ROCES, we conducted several experiments on the test sets. Here, the quality of a solution is computed in terms of the number of positive/negative examples covered/ruled out (given by the F_1 score in Table 7.2). First, we compare ROCES against the state-of-the-art on learning problems involving small sets of examples: we varied the number of examples between 4 and 264 using powers of 4 (cf. Figures 7.5, 7.6, and Table 7.2). In the second experiment, we compare all approaches on learning problems with full sets of examples (see columns named "Full" in Table 7.2). Finally, we measure how many times our approach ROCES outperforms the best baseline without using all the provided examples.

In Table 7.2, we used the Wilcoxon Signed-Rank test to compare our approach against other state-of-the-art methods; the asterisk ($*$) indicates whether differences are significant. From the Table, we can observe that ROCES significantly outperforms the state-of-the-art (up to +10% F_1 score) on the largest datasets Carcinogenesis and Vicodi when only 16 or 64 input examples are used. Moreover, ROCES remains the most accurate approach on

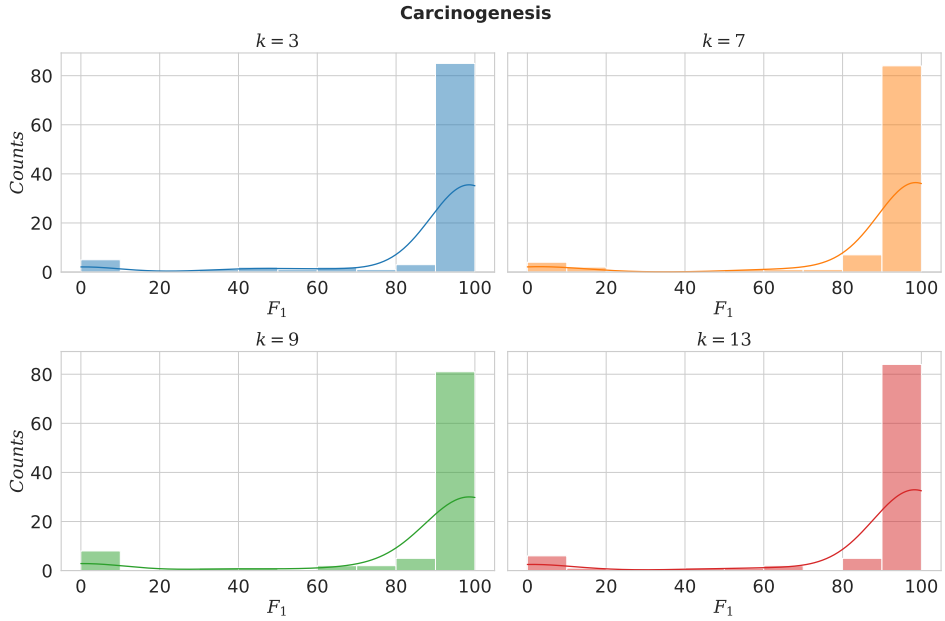


Figure 7.9: Distribution of the F-measure on Carcinogenesis in the active learning setting

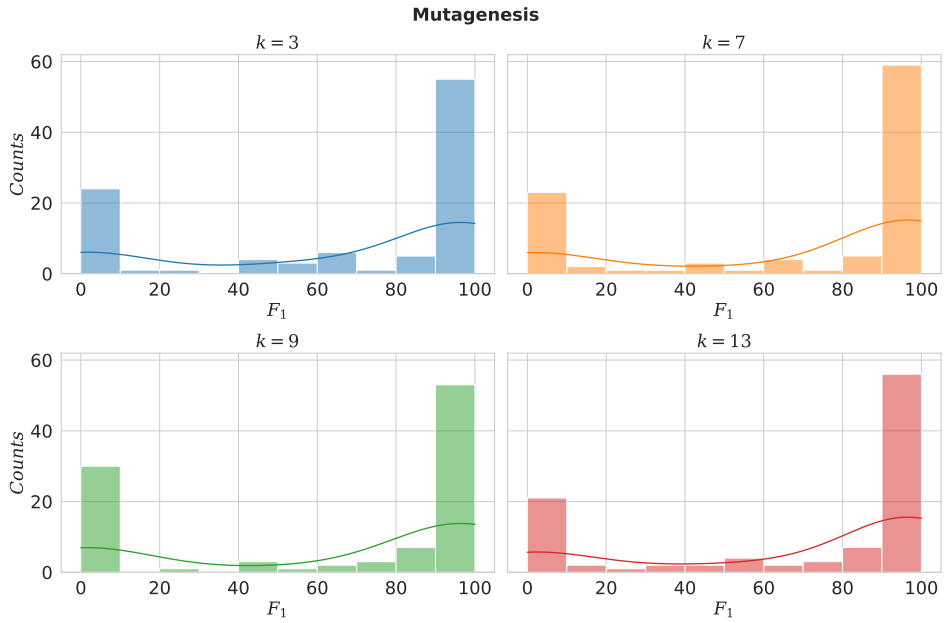


Figure 7.10: Distribution of the F-measure on Mutagenesis in the active learning setting

Carcinogenesis when the complete sets of examples are used, and ranks second on Vicodi behind NCES2.

On learning problems with limited input examples, NCES2 lags behind; it ranks last in 6 out of 8 cases when 16 or 64 examples are used. The same can be observed on Figure 7.5 where we plot the quality of the computed solutions for various input example set sizes. This is

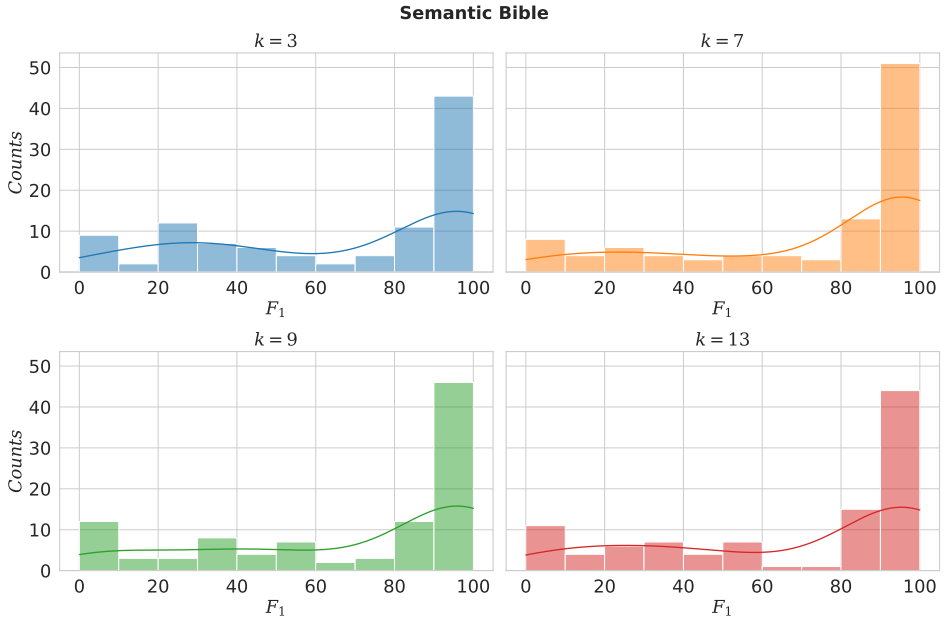


Figure 7.11: Distribution of the F-measure on Semantic Bible in the active learning setting

because NCES2 applies the naive approach of training on static input example sets predefined by the data generator. Search-based approaches (CELOE and EvoLearner) show a more stable performance across different input example set sizes as they use classification metrics at each step of the search process. However, they fall short when it comes to runtimes especially on large datasets, see e.g. Figure 7.6 and the lower part of Table 7.2. On average, they are over 1000× slower than ROCES and NCES2. Moreover, their prediction time increases with the number of examples used in a learning problem while deep learning-based approaches such as ROCES maintain a nearly constant prediction time (see Figure 7.6).

In Figures 7.9–7.12, we report experimental results for different values of k. In our experiments, the maximum number of iterations is set to 20, and initial sets of examples are of size at most 16. Results per learning problem are averaged across iterations. Following the steps described above, ROCES repeatedly interacts with an oracle to eventually improve the quality of its synthesized solutions. From the distribution of the F-measure in Figures 7.9, 7.10, 7.11, and 7.12, we can observe that ROCES is able to compute high-quality solutions even with small values of k. In fact, providing feedback with more than 7 newly labelled examples seems to have no positive impact on the quality of the computed solutions. This is probably due to the fact that ROCES is trained to perform well with small numbers of examples, see Algorithm 14.

7.2.3 Real-world Problems

We applied ROCES to a set of real-world learning problems from different datasets. We considered three datasets: Carcinogenesis (1 learning problem), Mutagenesis (1 learning

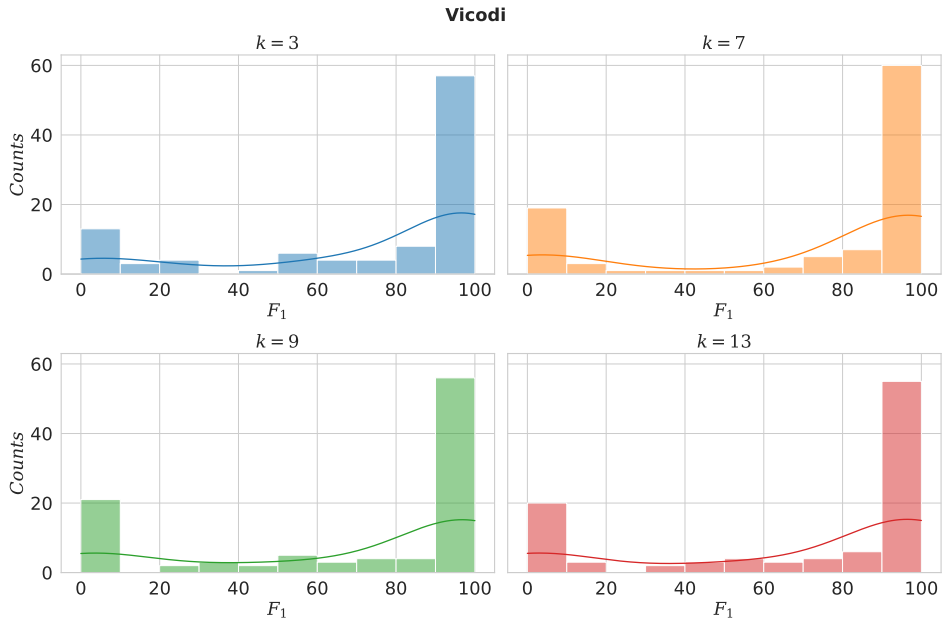


Figure 7.12: Distribution of the F-measure on Vicodi in the active learning setting

problem), and Family (7 learning problems). On Carcinogenesis and Mutagenesis, the goal is to describe chemical compounds that cause cancer and those that cause gene mutation, respectively. On the Family dataset, we aim to learn the concepts Uncle, Aunt, Cousin, Great-grandfather, Great-grandmother, Great-granddaughter, and Great-grandson. Note that each of the concepts corresponding to solutions of the given learning problems does not appear in the background knowledge nor in the training datasets.

Table 7.4: 10-fold cross-validation results on real-world problems. "–" denotes unknown learning problems.

	Train-F_1	*Test-F_1*	*Runtime (sec.)*
Family			
Aunt	0.80	0.81	1.12
Cousin	0.73	0.70	1.09
Great-granddaughter	1.00	1.00	0.92
Great-grandfather	0.94	0.93	0.92
Great-grandmother	0.94	0.95	1.31
Great-grandson	0.92	0.93	1.15
Uncle	0.88	0.89	0.91
Carcinogenesis			
–	0.71	0.70	1.77
Mutagenesis			
–	0.70	0.70	1.71

In Table 7.4, we report the results obtained on the above learning problems. We conducted experiments in a 10-fold cross-validation setting. Each of the ten folds contains some part

of the positive and negative examples. For each run, ROCES solves a learning problem (100 predictions per problem) using 9 folds. The 10-th fold is used for evaluation (see the column named $Test\text{-}F_1$ in Table 7.4). We report average results across the 10 folds. From the results, we can observe that ROCES maintains a high predictive accuracy while being scalable[4] ($\approx$1 second per problem). The carcinogenesis and mutagenesis problems are known to be challenging (in [27], a search-based approach achieves only 72% accuracy on Carcinogenesis), which also explains the relatively low performance achieved by ROCES on these problems.

7.3 Conclusion

In this chapter, we proposed a generalization of the traditional class expression learning problem and established the connections between the solutions of the classical and generalized problem. Our new reformulation forces learning algorithms to solve a given problem using cardinality-minimal sets of examples, thereby enhancing their ability to effectively tackle learning problems with arbitrary numbers of examples. We also proposed ROCES, a new learning algorithm for synthesis-based approaches to solve the generalized learning problem. Experimental results suggest that ROCES consistently outperforms previous synthesis-based approaches on learning problems with limited input examples while remaining highly scalable and competitive on full-size learning problems.

[4] Note that for each run, we perform 100 predictions per problem and select the class expression with the highest F_1 score. Hence, the runtime per prediction is $\approx$0.01 second on average.

8 Conclusion and Future Work

This chapter summarizes our research contributions, key findings, and future work directions. We start with search-based approaches for class expression learning before discussing our proposed family of neural class expression synthesizers. Finally, we conclude with potential future work directions and the broader impact of our research.

8.1 Summary

In this thesis, we focused on the development of class expression learning approaches at scale. A class expression is human-readable and interpretable, hence a white-box model. Most previous works for class expression learning investigated the use of description logic-specific refinement operators to construct and traverse partially ordered conceptual spaces in search of solutions to learning problems [62, 120, 122, 123, 177]. While the approaches developed in the aforementioned works are highly accurate on small datasets, they often require impractical runtimes on large-scale datasets due to the infinite and myopic nature of their search space, as illustrated in Figure 1.1. Moreover, approaches that do not use refinement operators and instead apply other search strategies—such as crossover and mutation operations [82], direct application of negation, disjunction and conjunction operators to preselected concepts [185]—also suffer similar scalability issues as suggested by our experiments.

8.1.1 Concept Learner with Integrated Length Prediction

Our first research contribution concerns the integration of concept length predictors into refinement operator-based approaches for class expression learning (see Chapter 4). The proposed approach CLIP [107] relies on the intuition that *if we have an accurate concept length predictor, then it can be used to approximate the length of solutions to learning problems.* To this end, we use neural networks, namely recurrent neural networks, convolutional neural networks, and multi-layer perceptrons, and develop an approach to automatically generate training data for them. The training data essentially consists of class expressions, their syntactic lengths as defined in Definition 2.2, and their respective sets of positive and negative examples. Hence, our overall pipeline for concept length prediction is trained in an unsupervised manner, i.e., our approach automatically creates data and labels it before training. Post-training, our concept length predictors not only significantly (p-value < 0.05

with the Wilcoxon Signed Rank test) outperform a random model, but their integration into the state of the art CELOE [122] also leads to improved runtimes and better solutions on large datasets. CLIP effectively narrows its search space by considering only concepts up to the predicted length, thereby avoiding large areas of the search space characterized by long concepts which are not necessarily candidate solutions to a learning problem and are hard to interpret. Nonetheless, as the size of the background knowledge base increases (e.g., when more atomic concepts are added), the search space of refinement operator-based approaches, including CLIP, grows considerably in size, resulting in long runtimes. This is due to the large number of refinements computed for each root node. Consequently, the pruning strategy of CLIP might not be sufficient on large datasets, especially if many learning problems are to be solved. We therefore propose a novel family of parallelizable, highly optimized algorithms for class expression learning on large datasets, see Section 8.1.2 below.

Key Findings. Concept lengths in $\mathcal{ALC}$ can be learned from the sets of instances described by those concepts. Moreover, incorporating reliable concept length predictors into search-based approaches for class expression learning often leads to improved runtimes and better quality solutions. However, as the size of the background knowledge base increases, the benefits of using concept length predictors to accelerate class expression learning diminish.

8.1.2 Neural Class Expression Synthesis

To address the scalability limitations inherent in search-based approaches for class expression learning, we developed a novel family of synthesis-based approaches targeting the description logic $\mathcal{ALC}$—neural class expression synthesizers (NCES) [113]—which directly translate input examples to class expressions without the need for an expensive search. Our work draws inspiration from the success of neural machine translation [35, 228], with large language models (LLMs), e.g., GPT-4 [1] achieving impressive results [105, 237, 248]. NCES use neural network architectures, e.g., Set-Transformer [118] and a vocabulary of tokens, constructed automatically from each input knowledge base, to synthesize class expressions. The vocabulary contains atomic concept and role names as well as description logic-specific tokens, e.g., $\forall$, $\sqcup$, $\neg$. As in the case of LLMs, NCES require pretraining to learn how to align tokens in a meaningful manner. To this end, we design a training data generation method akin to the one in our approach CLIP, but where actual class expressions are stored instead of their lengths. Moreover, we develop a tokenization technique to decompose each class expression into a sequence of tokens present in the predefined vocabulary. NCES are then trained to map the embeddings of positive and negative examples to a sequence of tokens representing a target class expression.

At inference time on unseen learning problems, NCES leverage the learned patterns to synthesize new class expressions based on input examples. One advantage of NCES is that they are not bound to a predefined set of class expressions as in the case of NERO [47]; they automatically adapt to new learning problems by selecting appropriate tokens and

their ordering to maximize the classification accuracy of input examples. Most importantly, NCES can solve multiple learning problems in parallel via batch processing, thanks to the parallelizable nature of deep neural networks and the availability of accelerated computing hardware such as GPUs. Our experimental results suggest that even on CPUs, NCES significantly (p-value < 0.05 with the Wilcoxon Signed Rank test) outperform search-based approaches for class expression learning while being highly competitive in predictive performance.

Despite their time efficiency and excellent predictive performance in the description logic $\mathcal{ALC}$, NCES naturally inherit certain vulnerabilities of neural networks, particularly concerning sensitivity to small changes in inputs: NCES do not perform consistently well when sets of positive and negative examples for a given learning problem are replaced by other relevant sets. For example, one might want to keep the set of positive examples unchanged but slightly modify the set of negative examples such that the original solution to the learning problem still holds. In such scenarios, NCES mostly fail to synthesize the desired solutions. Additionally, the dependence of NCES on pretrained embeddings of examples, and the limitation of their expressive power to the description logic $\mathcal{ALC}$ can also be seen as another hindrance. We address some of these limitations by proposing a new instance of synthesis-based approaches, with the main contributions summarized in Section 8.1.3.

Key Findings. Class expressions in $\mathcal{ALC}$ can be synthesized directly from sets of examples by using neural networks, e.g., LSTM, GRU, or Set-Transformer. Our proposed synthesis-based approaches for class expression learning (NCES) are several orders of magnitude faster than search-based approaches while being highly competitive in terms of quality of computed solutions. Nonetheless, NCES instances often struggle with generalization when input examples are swapped with other valid alternatives.

8.1.3 Neural Class Expression Synthesis in $\mathcal{ALCHIQ}^{(\mathcal{D})}$

We proposed NCES2 [109] as an extension of NCES to the description logic $\mathcal{ALCHIQ}^{(\mathcal{D})}$ which also comes with several improvements. Similarly to initial NCES instances, NCES2 relies on deep neural networks and a vocabulary of tokens to synthesis class expressions without a search process. Note that new special tokens are added to the vocabulary to support the expressive description logic $\mathcal{ALCHIQ}^{(\mathcal{D})}$, e.g., "$\geq$" and "$\leq$" for cardinality and number restrictions, and "$^-$" for inverse roles. Another addition is the integration of an embedding model into the learning process to alleviate the dependence on pre-existing embeddings for input examples. During training, the embedding model gets a batch of triples[1] as input and updates its trainable parameters such that its loss function (typically defined for link prediction on knowledge graphs [180, 207]) is minimized on the input batch. Simultaneously, the neural synthesizer (e.g., Set-Transformer) receives a batch of

[1] A triple represents a fact in the input knowledge base, e.g., the relationship between two individuals (which serve as examples in learning problems), or a class membership assertion.

class expressions and their corresponding sets of positive and negative examples (whose vector representations are provided by the embedding model) and updates its parameters by minimizing a token-level loss function, e.g., cross-entropy. Hence, the neural synthesizer and the embedding model are trained jointly to further improve the predictive performance of synthesis-based approaches for class expression learning.

NCES2 also employs a training data augmentation technique, where multiple sets of input examples are constructed for each learning problem in the training set. Because NCES assume that the maximum length of inputs is fixed, the sets of examples for some learning problems are downsampled during the data generation process. However, these omitted examples are also valid ones, which is not taken into account by early NCES instances. This causes NCES instances to perform poorly when such valid examples are used in a given learning problem at inference time. By constructing multiple samples per learning problem, NCES2 effectively improves the generalization capability of synthesis-based approaches. In Section 8.1.4 below, we summarize our last research contribution which further enhances the robustness of neural class expression synthesizers.

Key Findings. Class expressions in expressive description logics, e.g., $\mathcal{ALCHIQ}^{(\mathcal{D})}$ can also be synthesized directly from sets of examples by using neural networks. Moreover, an embedding model can be coupled with a neural synthesizer to enable end-to-end training and inference, thereby eliminating the requirement for pretrained embeddings of background knowledge bases. By constructing multiple sets of examples per learning problem during training, the predictive performance and generalization capability of synthesis-based approaches can further be improved.

8.1.4 Robust Class Expression Synthesis

Here, we proposed a generalization of the classical class expression learning problem and established connections between solutions of the new formulation and classical ones [111]. For example, we showed how solutions to the generalized learning problem ($\mathcal{GLP}$) can be sought starting from an arbitrary solution of the classical problem. The generalized learning problem forces learning systems to compute a solution to the classical problem by using the smallest possible subsets of examples, thereby improving their ability to solve learning problems with arbitrary numbers of examples. This reformulation is motivated by the fact that in real-world applications of class expression learning, e.g., ontology engineering [122], there can be as few as two examples in a learning problem (e.g., a knowledge engineer identifies few individuals that do not have enough type information and wants to enrich them with fitting class expressions) or several hundreds of examples (e.g., a knowledge engineer has identified a repeated pattern on many individuals and seeks clarification).

We also develop a learning algorithm for synthesis-based approaches to solve the generalized learning problem. In the training phase, our algorithm iteratively constructs subsets of input examples whose sizes are drawn from predefined probability distributions skewed

towards small values, and aims to synthesize correct solutions from those examples by minimizing the running loss using gradient-based optimization methods. The overall learning approach is referred to as ROCES. In our experiments, ROCES consistently outperforms previous synthesis-based approaches on learning problems with limited input examples while remaining highly scalable and competitive on full-size learning problems.

Key Findings. Using iterative sampling to construct sets of examples with various sizes during training improves the robustness of synthesis-based approaches to changes in the number of provided examples at inference time.

8.2 Future Work

In the near future, we aim to first improve the training data generation method of neural class expression synthesizers. As described in Chapters 4 and 5, we generate training data for neural class expression synthesis using a length-based refinement operator (see Algorithms 12 and 13 for details). Moreover, instance retrieval for each expression generated by this operator is managed by OWL reasoners [71, 193], which are not natively parallelizable. Consequently, generating sufficient training data via the above approach is computationally expensive and inefficient, especially on large datasets. Our experiments throughout this thesis suggest that the lack of sufficient training data often leads to poorly trained neural synthesizers which then underperform search-based methods. This demonstrates the need for a more efficient approach for training data generation or augmentation. Recently, there has been progress towards using SPARQL to answer queries corresponding to instance retrieval in description logics [101, 155]. By using this new approach, multiple queries can be sent to a SPARQL query endpoint, which allows to retrieve instances for multiple class expressions in parallel, thereby accelerating the training data generation process. Additionally, it could be interesting to investigate new ways to synthesize class expressions, e.g., using the next token prediction paradigm as currently done in large language models (LLMs), to further improve the predictive performance of our neural class expression synthesizers.

Second, we will explore potential use cases of neural class expression synthesizers beyond benchmark datasets. One direction could be a deployment on the web, where NCES instances can provide real-time answers to users' queries. For instance, users can ask for descriptions of collections of items, which can be easily translated into class expression learning problems: positive examples are items to be described, and negative examples can be any subset of the remaining items. For each query, our neural class expression synthesizers compute a class expression which can be verbalized into natural language text thanks to the verbalization module of Ontolearn[2]—a library for structured machine learning in Python in which NCES are already implemented. This item description setting is also valid for augmenting recommendation systems with class expression learners: sellers may

[2] `https://github.com/dice-group/Ontolearn`

be interested in knowing their costumers preferences, in which case they can provide a set of items bought in the past, i.e., positive examples. In this way, user preferences can be represented with learned class expressions, and both sellers and buyers get to know why a given item is recommended. A clear advantage of our neural synthesizers over traditional approaches for class expression learning is that the former support sophisticated computing hardware such as GPUs and can process multiple queries in parallel, which is essential for web-scale applications and for improving user experience.

Finally, the integration of neural class expression synthesizers into deep learning architectures to make the latter gray-box and enable explainable predictions is another promising research direction. Most of the existing deep learning architectures are black-box [74, 164, 176], which means it is extremely hard or impossible to explain how their outputs are derived. Current research in explainable AI has mainly focused on using decision trees, rules, and Bayesian networks to explain already existing black-box models [64, 74, 94, 249]. Other works exploit gradient-based and perturbation-based methods to explain the predictions of black-box models [30, 31, 189, 190, 232]. Simple white-box models such as decision trees often underperform deep neural networks [161, 184], and gradient- and perturbation-based methods for explaining black-box models are mostly post-hoc [4, 97, 190]. We will develop techniques to integrate neural class expression synthesizers into deep neural networks to enable ante-hoc explainability. To this end, we could first investigate ways to automatically annotate input data to obtain description logic knowledge bases (see [61] for a related work), which are required by NCES instances. As an example, in a dataset containing images of cats and dogs, the following assertions could be part of the constructed knowledge base: $\leq 7cm$ `hasEarLength(Animal_0)`, $\exists$ `hasEarShape.Pointy(Animal_1)`. Moreover, NCES instances should be able to use features from the deep learning model to be explained, e.g., outputs from the last hidden layer, to synthesize class expressions which faithfully describe the input data and the black-box model's understanding of it.

Bibliography

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Julián N Acosta, Guido J Falcone, Pranav Rajpurkar, and Eric J Topol. Multimodal biomedical ai. *Nature Medicine*, 28(9):1773–1784, 2022.

[3] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160, 2018.

[4] Sushant Agarwal, Shahin Jabbari, Chirag Agarwal, Sohini Upadhyay, Steven Wu, and Himabindu Lakkaraju. Towards the unification and robustness of perturbation and gradient based explanations. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 110–119. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/agarwal21c.html`.

[5] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD Conference*, pages 207–216. ACM Press, 1993.

[6] Arohan Ajit, Koustav Acharya, and Abhishek Samanta. A review of convolutional neural networks. In *2020 international conference on emerging trends in information technology and engineering (ic-ETITE)*, pages 1–5. IEEE, 2020.

[7] Lina Muhammad Al-Ghamdi. Towards adopting ai techniques for monitoring social media activities. *Sustainable Engineering and Innovation*, 3(1):15–22, 2021.

[8] Maximilian Alber, Sebastian Lapuschkin, Philipp Seegerer, Miriam Hägele, Kristof T Schütt, Grégoire Montavon, Wojciech Samek, Klaus-Robert Müller, Sven Dähne, and Pieter-Jan Kindermans. innvestigate neural networks! *Journal of machine learning research*, 20(93):1–8, 2019.

[9] Md Zahangir Alom, Tarek M Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C Van Essen, Abdul AS Awwal, and Vijayan K Asari. A state-of-the-art survey on deep learning theory and architectures. *electronics*, 8(3):292, 2019.

[10] Jafar Alzubi, Anand Nayyar, and Akshi Kumar. Machine learning from theory to algorithms: an overview. In *Journal of physics: conference series*, volume 1142, page 012012. IOP Publishing, 2018.

[11] Plamen P Angelov, Eduardo A Soares, Richard Jiang, Nicholas I Arnold, and Peter M Atkinson. Explainable artificial intelligence: an analytical review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 11(5):e1424, 2021.

[12] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information fusion*, 58: 82–115, 2020.

[13] Konstantina Athanasopoulou, Glykeria N Daneva, Panagiotis G Adamopoulos, and Andreas Scorilas. Artificial intelligence: the milestone in modern biomedical research. *BioMedInformatics*, 2(4):727–744, 2022.

[14] Taiwo Oladipupo Ayodele. Machine learning overview. *New Advances in Machine Learning*, 2(9-18):16, 2010.

[15] Liviu Badea and Shan-Hwei Nienhuys-Cheng. A refinement operator for description logics. In *ILP*, volume 1866 of *Lecture Notes in Computer Science*, pages 40–59. Springer, 2000.

[16] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[17] Ivana Balažević, Carl Allen, and Timothy M Hospedales. Hypernetwork knowledge graph embeddings. In *International Conference on Artificial Neural Networks*, pages 553–565. Springer, 2019.

[18] Jon Barwise. An introduction to first-order logic. In *Studies in Logic and the Foundations of Mathematics*, volume 90, pages 5–46. Elsevier, 1977.

[19] Eric B Baum. On the capabilities of multilayer perceptrons. *Journal of complexity*, 4 (3):193–215, 1988.

[20] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012.

[21] Mainak Biswas, Venkatanareshbabu Kuppili, Luca Saba, Damodar Reddy Edla, Harman S Suri, Elisa Cuadrado-Godia, John R Laird, Rui Tato Marinhoe, Joao M Sanches, Andrew Nicolaides, et al. State-of-the-art review on deep learning in medical imaging. *Frontiers in Bioscience-Landmark*, 24(3):380–406, 2019.

[22] Martin Boeker, Fábio França, Peter Bronsert, and Stefan Schulz. Tnm-o: ontology support for staging of malignant tumours. *Journal of biomedical semantics*, 7:1–11, 2016.

[23] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, pages 2787–2795, 2013.

[24] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. A semantic matching energy function for learning with multi-relational data - application to word-sense disambiguation. *Mach. Learn.*, 94(2):233–259, 2014.

[25] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1442–1451, 2017.

[26] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J. Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael S. Ryoo, Grecia Salazar, Pannag R. Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong T. Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-1: robotics transformer for real-world control at scale. In *Robotics: Science and Systems*, 2023.

[27] Lorenz Bühmann, Jens Lehmann, and Patrick Westphal. Dl-learner—a framework for inductive learning on the semantic web. *Journal of Web Semantics*, 39:15–24, 2016.

[28] Hans Kleine Büning and Theodor Lettmann. *Propositional logic: deduction and algorithms*, volume 48. Cambridge University Press, 1999.

[29] Erik Cambria and Bebo White. Jumping NLP curves: A review of natural language processing research [review article]. *IEEE Comput. Intell. Mag.*, 9(2):48–57, 2014.

[30] Zachariah Carmichael and Walter J Scheirer. Unfooling perturbation-based post hoc explainers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 6925–6934, 2023.

[31] Aditya Chattopadhay, Anirban Sarkar, Prantik Howlader, and Vineeth N Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE winter conference on applications of computer vision (WACV)*, pages 839–847. IEEE, 2018.

[32] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *NeurIPS*, pages 15084–15097, 2021.

[33] Muhao Chen and Carlo Zaniolo. Learning multi-faceted knowledge graph embeddings for natural language processing. In *IJCAI*, pages 5169–5170, 2017.

[34] Cristina Chesta, Olivier Siohan, and Chin-Hui Lee. Maximum a posteriori linear regression for hidden markov model adaptation. In *Eurospeech*, volume 99, pages 211–214, 1999.

[35] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *SSST@EMNLP*, pages 103–111. Association for Computational Linguistics, 2014.

[36] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, pages 1724–1734. ACL, 2014.

[37] William F Clocksin and Christopher S Mellish. *Programming in PROLOG*. Springer Science & Business Media, 2003.

[38] Alain Colmerauer. An introduction to prolog iii. *Communications of the ACM*, 33(7): 69–90, 1990.

[39] Mark Craven and Jude Shavlik. Extracting tree-structured representations of trained networks. *Advances in neural information processing systems*, 8, 1995.

[40] Balázs Csanád Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[41] Yuanfei Dai, Shiping Wang, Neal N Xiong, and Wenzhong Guo. A survey on knowledge graph embedding: Approaches, applications and benchmarks. *Electronics*, 9(5):750, 2020.

[42] Thomas Davenport and Ravi Kalakota. The potential for artificial intelligence in healthcare. *Future healthcare journal*, 6(2):94–98, 2019.

[43] Luc De Raedt and Luc Dehaspe. Clausal discovery. *Machine Learning*, 26:99–146, 1997.

[44] Caglar Demir. *Learning continuous representations for Knowledge Graphs*. PhD thesis, University of Paderborn, Germany, 2023.

[45] Caglar Demir and Axel-Cyrille Ngonga Ngomo. Convolutional complex knowledge graph embeddings. In *ESWC*, volume 12731 of *Lecture Notes in Computer Science*, pages 409–424. Springer, 2021.

[46] Caglar Demir and Axel-Cyrille Ngonga Ngomo. Clifford embeddings–a generalized approach for embedding in normed algebras. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 567–582. Springer, 2023.

[47] Caglar Demir and Axel-Cyrille Ngonga Ngomo. Learning permutation-invariant embeddings for description logic concepts. In *International Symposium on Intelligent Data Analysis*, pages 103–115. Springer, 2023.

[48] Caglar Demir and Axel-Cyrille Ngonga Ngomo. Neuro-symbolic class expression learning. In *IJCAI*, pages 3624–3632, 2023.

[49] Caglar Demir, Diego Moussallem, Stefan Heindorf, and Axel-Cyrille Ngonga Ngomo. Convolutional hypercomplex embeddings for link prediction. In *Asian Conference on Machine Learning*, pages 656–671. PMLR, 2021.

[50] Caglar Demir, Anna Himmelhuber, Yushan Liu, Alexander Bigerl, Diego Moussallem, and Axel-Cyrille Ngonga Ngomo. Rapid explainability for skill description learning. In Anastasia Dimou, Armin Haller, Anna Lisa Gentile, and Petar Ristoski, editors, *Proceedings of the ISWC 2022 Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 21st International Semantic Web Conference (ISWC 2022), Virtual Conference, Hangzhou, China, October 23-27, 2022*, volume 3254 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022. URL `https://ceur-ws.org/Vol-3254/paper403.pdf`.

[51] Caglar Demir, N'Dah Jean Kouagou, Arnab Sharma, and Axel-Cyrille Ngonga Ngomo. Inference over unseen entities, relations and literals on knowledge graphs. In *CompAI Workshop, ECAI*, 2024.

[52] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[53] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[54] Thomas G Dietterich, Richard H Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial intelligence*, 89(1-2): 31–71, 1997.

[55] Laura Dietz, Alexander Kotov, and Edgar Meij. Utilizing knowledge graphs for text-centric information retrieval. In *The 41st international ACM SIGIR conference on research & development in information retrieval*, pages 1387–1390, 2018.

[56] Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. A survey on ensemble learning. *Frontiers Comput. Sci.*, 14(2):241–258, 2020.

[57] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[58] Scott R Eliason. *Maximum likelihood estimation: Logic and practice*, volume 96. Sage, 1993.

[59] Waleed Abd Elkhalik. Ai-driven smart homes: Challenges and opportunities. *Journal of Intelligent Systems & Internet of Things*, 8(2), 2023.

[60] Erika Erickson, Japheth E Gado, Luisana Avilán, Felicia Bratti, Richard K Brizendine, Paul A Cox, Raj Gill, Rosie Graham, Dong-Jin Kim, Gerhard König, et al. Sourcing thermotolerant poly (ethylene terephthalate) hydrolase scaffolds from natural diversity. *Nature communications*, 13(1):7850, 2022.

[61] Zoe Falomir, Ernesto Jiménez-Ruiz, M Teresa Escrig, and Lledó Museros. Describing images using qualitative models and description logics. *Spatial Cognition & Computation*, 11(1):45–74, 2011.

[62] Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. DL-FOIL concept learning in description logics. In *ILP*, volume 5194 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2008.

[63] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017.

[64] Alex A Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD explorations newsletter*, 15(1):1–10, 2014.

[65] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. Fast rule mining in ontological knowledge bases with amie+. *The VLDB Journal*, 24(6): 707–730, 2015.

[66] Mikhail Galkin, Etienne G. Denis, Jiapeng Wu, and William L. Hamilton. Nodepiece: Compositional and parameter-efficient representations of large knowledge graphs. In *ICLR*. OpenReview.net, 2022.

[67] Mikhail Galkin, Xinyu Yuan, Hesham Mostafa, Jian Tang, and Zhaocheng Zhu. Towards foundation models for knowledge graph reasoning. In *The Twelfth International Conference on Learning Representations*, 2024.

[68] Marta Garnelo and Murray Shanahan. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 29:17–23, 2019.

[69] J-L Gauvain and Chin-Hui Lee. Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains. *IEEE transactions on speech and audio processing*, 2(2):291–298, 1994.

[70] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.

[71] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: an owl 2 reasoner. *Journal of automated reasoning*, 53:245–269, 2014.

[72] Ian Goodfellow. Deep learning, 2016.

[73] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of field robotics*, 37(3): 362–386, 2020.

[74] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, 2018.

[75] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[76] Xiao Guo, Zhenjiang Shen, Yajing Zhang, and Teng Wu. Review on the application of artificial intelligence in smart homes. *Smart Cities*, 2(3):402–420, 2019.

[77] Volker Haarslev, Kay Hidde, Ralf Möller, and Michael Wessel. The racerpro knowledge representation and reasoning system. *Semantic Web*, 3(3):267–277, 2012.

[78] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[79] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. *Advances in neural information processing systems*, 34:15908–15919, 2021.

[80] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, Zhaohui Yang, Yiman Zhang, and Dacheng Tao. A survey on vision transformer. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(1):87–110, 2023.

[81] Frank W Hartel, Sherri de Coronado, Robert Dionne, Gilberto Fragoso, and Jennifer Golbeck. Modeling a description logic vocabulary for cancer research. *Journal of biomedical informatics*, 38(2):114–129, 2005.

[82] Stefan Heindorf, Lukas Blübaum, Nick Düsterhus, Till Werner, Varun Nandkumar Golani, Caglar Demir, and Axel-Cyrille Ngonga Ngomo. Evolearner: Learning description logics with evolutionary algorithms. In *WWW*, pages 818–828. ACM, 2022.

[83] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[84] Pascal Hitzler and Md Kamruzzaman Sarker. Neuro-symbolic artificial intelligence: The state of the art. *Frontiers in Artificial Intelligence and Applications*, 2022.

[85] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation MIT-Press*, 9(8):1735–1780, 1997.

[86] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan F. Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4):71:1–71:37, 2022. doi: 10.1145/3447772. URL `https://doi.org/10.1145/3447772`.

[87] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[88] John J Hopfield. Hopfield network. *Scholarpedia*, 2(5):1977, 2007.

[89] Alfred Horn. On sentences which are true of direct unions of algebras. *Journal of Symbolic Logic*, 16(1):14–21, 1951. doi: 10.2307/2268661.

[90] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[91] Binbin Hu, Pan Zhou, Hongfang Yu, Yueyue Dai, Ming Wang, Shengbo Tan, and Ying Sun. Leanet: Lightweight u-shaped architecture for high-performance skin cancer image segmentation. *Computers in Biology and Medicine*, 169:107919, 2024.

[92] Sen Hu, Lei Zou, Jeffrey Xu Yu, Haixun Wang, and Dongyan Zhao. Answering natural language questions by subgraph matching over knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, 30(5):824–837, 2017.

[93] Huimin Huang, Lanfen Lin, Ruofeng Tong, Hongjie Hu, Qiaowei Zhang, Yutaro Iwamoto, Xianhua Han, Yen-Wei Chen, and Jian Wu. Unet 3+: A full-scale connected unet for medical image segmentation. In *ICASSP 2020-2020 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 1055–1059. IEEE, 2020.

[94] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51(1):141–154, 2011.

[95] Eleni Ilkou and Maria Koutraki. Symbolic vs sub-symbolic ai methods: Friends or enemies? In *CIKM (Workshops)*, volume 2699, 2020.

[96] Muhammad Imran, Ferda Ofli, Doina Caragea, and Antonio Torralba. Using ai and social media multimodal content for disaster response and management: Opportunities, challenges, and future directions, 2020.

[97] Weina Jin, Xiaoxiao Li, Mostafa Fatehi, and Ghassan Hamarneh. Generating post-hoc explanation from deep neural networks for multi-modal medical image analysis tasks. *MethodsX*, 10:102009, 2023.

[98] Ulf Johansson and Lars Niklasson. Evolving decision trees using oracle guides. In *2009 IEEE Symposium on Computational Intelligence and Data Mining*, pages 238–244. IEEE, 2009.

[99] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

[100] Louis Mozart Kamdem Teyou, Caglar Demir, and Axel-Cyrille Ngonga Ngomo. Embedding knowledge graphs in degenerate clifford algebras. In *ECAI 2024*, pages 1293–1300. IOS Press, 2024.

[101] Nikolaos Karalis, Alexander Bigerl, Caglar Demir, Liss Heidrich, and Axel-Cyrille Ngonga Ngomo. Evaluating negation with multi-way joins accelerates class expression learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 199–216. Springer, 2024.

[102] Salman H. Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM Comput. Surv.*, 54(10s):200:1–200:41, 2022.

[103] Wahab Khan, Ali Daud, Khairullah Khan, Shakoor Muhammad, and Rafiul Haq. Exploring the frontiers of deep learning and natural language processing: A comprehensive overview of key challenges and emerging trends. *Natural Language Processing Journal*, page 100026, 2023.

[104] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[105] Tom Kocmi and Christian Federmann. Large language models are state-of-the-art evaluators of translation quality. In Mary Nurminen, Judith Brenner, Maarit Koponen, Sirkku Latomaa, Mikhail Mikhailov, Frederike Schierl, Tharindu Ranasinghe, Eva Vanmassenhove, Sergi Alvarez Vidal, Nora Aranberri, Mara Nunziatini, Carla Parra Escartín, Mikel Forcada, Maja Popovic, Carolina Scarton, and Helena Moniz, editors, *Proceedings of the 24th Annual Conference of the European Association for Machine Translation*, pages 193–203, Tampere, Finland, June 2023. European Association for Machine Translation. URL `https://aclanthology.org/2023.eamt-1.19`.

[106] N'Dah Jean Kouagou, P.G. Dlamini, and S.M. Simelane. On the multi-domain compact finite difference relaxation method for high dimensional chaos: The nine-dimensional lorenz system. *Alexandria Engineering Journal*, 59(4):2617–2625, 2020. ISSN 1110-0168. doi: https://doi.org/10.1016/j.aej.2020.04.025. URL `https://www.sciencedirect.com/science/article/pii/S1110016820301769`. New trends of numerical and analytical methods for engineering problems.

[107] N'Dah Jean Kouagou, Stefan Heindorf, Caglar Demir, and Axel-Cyrille Ngonga Ngomo. Learning concept lengths accelerates concept learning in ALC. In *ESWC*, volume 13261 of *Lecture Notes in Computer Science*, pages 236–252. Springer, 2022.

[108] N'Dah Jean Kouagou, Stefan Heindorf, Caglar Demir, and Axel-Cyrille Ngonga Ngomo. Neural class expression synthesis. In *NeSy*, volume 3432 of *CEUR Workshop Proceedings*, pages 430–431. CEUR-WS.org, 2023.

[109] N'Dah Jean Kouagou, Stefan Heindorf, Caglar Demir, and Axel-Cyrille Ngonga Ngomo. Neural class expression synthesis in *ALCHIQ(D)*. In *ECML/PKDD (4)*, volume 14172 of *Lecture Notes in Computer Science*, pages 196–212. Springer, 2023.

[110] N'Dah Jean Kouagou, Caglar Demir, Hamada M. Zahera, Adrian Wilke, Stefan Heindorf, Jiayi Li, and Axel-Cyrille Ngonga Ngomo. Universal knowledge graph embeddings. In *Companion Proceedings of the ACM on Web Conference 2024*, WWW '24, page 1793–1797, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400701726. doi: 10.1145/3589335.3651978. URL https://doi.org/10.1145/3589335.3651978.

[111] N'Dah Jean Kouagou, Stefan Heindorf, Caglar Demir, and Axel-Cyrille Ngonga Ngomo. Roces: Robust class expression synthesis in description logics via iterative sampling. In Kate Larson, editor, *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 4335–4343. International Joint Conferences on Artificial Intelligence Organization, 8 2024. doi: 10.24963/ijcai.2024/479. URL https://doi.org/10.24963/ijcai.2024/479. Main Track.

[112] N'Dah Jean Kouagou, Arif Yilmaz, Michel Dumontier, and Axel-Cyrille Ngonga Ngomo. Discovering the unknown: Improving rule mining via embedding-based link prediction. *Knowledge Based Systems*, 00(0):0000–0000, 2024. ISSN 0000-0000.

[113] N'Dah Jean Kouagou, Stefan Heindorf, Caglar Demir, and Axel-Cyrille Ngonga Ngomo. Neural class expression synthesis. In *European Semantic Web Conference*, pages 209–226. Springer, 2023.

[114] R Krishnan, G Sivakumar, and P Bhattacharya. Extracting decision trees from trained neural networks. *Pattern recognition*, 32(12), 1999.

[115] Sanjay Krishnan and Eugene Wu. Palm: Machine learning explanations for iterative debugging. In *Proceedings of the 2Nd workshop on human-in-the-loop data analytics*, pages 1–6, 2017.

[116] Evelina Lamma, Fabrizio Riguzzi, LM Pereira, et al. Learning three-valued logic programs. *SFERA Archive of Research Products of the University of Ferrara*, 1999.

[117] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

[118] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019.

[119] J. Lehmann. Dl-learner: learning concepts in description logics. *The Journal of Machine Learning Research*, 2009.

[120] J. Lehmann. *Learning OWL class expressions*, volume 22. IOS Press, 2010.

[121] J. Lehmann and P. Hitzler. Concept learning in description logics using refinement operators. *Machine Learning*, 78, 2010.

[122] J. Lehmann, S. Auer, L. Bühmann, and S. Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, 2011.

[123] Jens Lehmann and Pascal Hitzler. Foundations of refinement operators for description logics. In *ILP*, volume 4894 of *Lecture Notes in Computer Science*, pages 161–174. Springer, 2007.

[124] Jens Lehmann and Johanna Völker. *Perspectives on ontology learning*, volume 18. IOS Press, 2014.

[125] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Soeren Kammel, J Zico Kolter, Dirk Langer, Oliver Pink, Vaughan Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE intelligent vehicles symposium (IV)*, pages 163–168. IEEE, 2011.

[126] Dongsheng Li, Chao Chen, Qin Lv, Li Shang, Stephen Chu, and Hongyuan Zha. Ermma: Expected risk minimization for matrix approximation-based recommender systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[127] Li Li, Miloš Doroslovački, and Murray H Loew. Approximating the gradient of cross-entropy loss function. *IEEE access*, 8:111626–111635, 2020.

[128] Wenzhe Li, Hao Luo, Zichuan Lin, Chongjie Zhang, Zongqing Lu, and Deheng Ye. A survey on transformers in reinforcement learning. *Trans. Mach. Learn. Res.*, 2023, 2023.

[129] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, pages 2181–2187. AAAI Press, 2015.

[130] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. Explainable ai: A review of machine learning interpretability methods. *Entropy*, 23(1):18, 2020.

[131] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[132] Zhenghao Liu, Chenyan Xiong, Maosong Sun, and Zhiyuan Liu. Entity-duet neural ranking: Understanding the role of knowledge graph semantics in neural information retrieval. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2395–2405, 2018.

[133] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *EMNLP*, pages 1412–1421. The Association for Computational Linguistics, 2015.

[134] Jingzhe Ma, Shaobo Duan, Ye Zhang, Jing Wang, Zongmin Wang, Runzhi Li, Yongli Li, Lianzhong Zhang, and Huimin Ma. Efficient deep learning architecture for detection and recognition of thyroid nodules. *Computational Intelligence and Neuroscience*, 2020(1):1242781, 2020.

[135] Yue Ma and Felix Distel. Learning formal definitions for snomed CT from text. In *AIME*, volume 7885 of *Lecture Notes in Computer Science*, pages 73–77. Springer, 2013.

[136] Tambiama Madiega. Artificial intelligence act. *European Parliament: European Parliamentary Research Service*, 2021.

[137] Samuel Manoharan et al. An improved safety algorithm for artificial intelligence enabled processors in self driving cars. *Journal of artificial intelligence*, 1(02):95–104, 2019.

[138] Rainer Manthey and François Bry. Satchmo: A theorem prover implemented in prolog. In Ewing Lusk and Ross Overbeek, editors, *9th International Conference on Automated Deduction*, pages 415–434, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg. ISBN 978-3-540-39216-3.

[139] Anqi Mao, Mehryar Mohri, and Yutao Zhong. Cross-entropy loss functions: Theoretical analysis and applications. In *International conference on Machine learning*, pages 23803–23828. PMLR, 2023.

[140] Elio Masciari, Areeba Umair, and Muhammad Habib Ullah. A systematic literature review on ai based recommendation systems and their ethical considerations. *IEEE Access*, 2024.

[141] Nicholas Matsumoto, Jay Moran, Hyunjun Choi, Miguel E Hernandez, Mythreye Venkatesan, Paul Wang, and Jason H Moore. Kragen: a knowledge graph-enhanced rag framework for biomedical problem solving using large language models. *Bioinformatics*, 40(6), 2024.

[142] Larry R Medsker, Lakhmi Jain, et al. Recurrent neural networks. *Design and Applications*, 5(64-67):2, 2001.

[143] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *IJCAI*, pages 3137–3143, 2019.

[144] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2013.

[145] Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

[146] Raymond J Mooney and Mary Elaine Califf. Induction of first-order decision lists: Results on learning the past tense of english verbs. *Journal of Artificial Intelligence Research*, 3:1–24, 1995.

[147] James Moore and Allen Newell. How can merlin understand? *L. Gregg, Knowledge and Cognition*, 1973.

[148] S. Muggleton. Inductive logic programming. *New generation computing*, 1991.

[149] Stephen Muggleton. Inverse entailment and progol. *New generation computing*, 13: 245–286, 1995.

[150] Stephen H Muggleton, Cao Feng, et al. *Efficient induction of logic programs*. Turing Institute London, UK, 1990.

[151] Khan Muhammad, Amin Ullah, Jaime Lloret, Javier Del Ser, and Victor Hugo C de Albuquerque. Deep learning for safe autonomous driving: Current challenges and future directions. *IEEE Transactions on Intelligent Transportation Systems*, 22(7): 4316–4336, 2020.

[152] Gábor Nagypál. History ontology building: The technical view. *Humanities, Computers and Cultural Heritage*, page 207, 2005.

[153] D. Nardi, Ronald J. Brachman, et al. An introduction to description logics. *Description logic handbook*, 1, 2003.

[154] Daniele Nardi, Ronald J Brachman, et al. An introduction to description logics. *Description logic handbook*, 1:40, 2003.

[155] Axel-Cyrille Ngonga Ngomo, Caglar Demir, N'Dah Jean Kouagou, Stefan Heindorf, Nikolaos Karalis, and Alexander Bigerl. Class expression learning with multiple representations. In *Compendium of Neurosymbolic Artificial Intelligence*, volume 369 of *Frontiers in Artificial Intelligence and Applications*, pages 272–286. IOS Press, 2023.

[156] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Icml*, 2011.

[157] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing YAGO: scalable machine learning for linked data. In *WWW*, pages 271–280. ACM, 2012.

[158] Feiping Nie, Zhanxuan Hu, and Xuelong Li. An investigation for loss functions widely used in machine learning. *Communications in Information and Systems*, 18(1):37–52, 2018.

[159] Shan-Hwei Nienhuys-Cheng and Roland de Wolf. *What is inductive logic programming?* Springer, 1997.

[160] Ding Ning, Varvara Vetrova, Karin Bryan, Yun Sing Koh, Andreas Voskou, N'Dah Jean Kouagou, and Arnab Sharma. Diving deep: Forecasting sea surface temperatures and anomalies. In *ECML PKDD Discovery Track*, 2024. URL `https://divingdeepecml.github.io/`.

[161] Yuzuru Okajima and Kunihiko Sadamasa. Deep neural networks constrained by decision rules. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2496–2505, 2019.

[162] Adrian Paschke and Michale Schroeder. Inductive logic programming for bioinformatics in prova. *VLDB DBM*, 2007.

[163] Michael Pazzani and Dennis Kibler. The utility of knowledge in inductive learning. *Machine learning*, 9:57–94, 1992.

[164] Jeremy Petch, Shuang Di, and Walter Nelson. Opening the black box: the promise and limitations of explainable machine learning in cardiology. *Canadian Journal of Cardiology*, 38(2):204–213, 2022.

[165] Giulio Petrucci, Chiara Ghidini, and Marco Rospocher. Ontology learning in the deep. In *EKAW*, volume 10024 of *Lecture Notes in Computer Science*, pages 480–495, 2016.

[166] Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *Ussr computational mathematics and mathematical physics*, 4(5):1–17, 1964.

[167] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.

[168] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. In *CVPR*, pages 7077–7087. Computer Vision Foundation / IEEE, 2021.

[169] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 652–660, 2017.

[170] J. Ross Quinlan. Learning logical definitions from relations. *Machine learning*, 5: 239–266, 1990.

[171] J Ross Quinlan and R Mike Cameron-Jones. Foil: A midterm report. In *Machine Learning: ECML-93: European Conference on Machine Learning Vienna, Austria, April 5–7, 1993 Proceedings 6*, pages 1–20. Springer, 1993.

[172] Keshavagari Smithin Reddy, Ramya Polaki, V Sulochana, Gundala Pallavi, et al. Comparative analysis of deep learning models for early skin cancer detection using 3d total body photography. In *2024 4th International Conference on Sustainable Expert Systems (ICSES)*, pages 1275–1281. IEEE, 2024.

[173] Ridho Reinanda, Edgar Meij, Maarten de Rijke, et al. Knowledge graphs: An information retrieval perspective. *Foundations and Trends® in Information Retrieval*, 14(4): 289–444, 2020.

[174] Raymond Reiter. On closed world data bases. In *Readings in artificial intelligence*, pages 119–140. Elsevier, 1981.

[175] Andrew Philip Rennison, Peter Westh, and Marie Sofie Møller. Protein-plastic interactions: the driving forces behind the high affinity of a carbohydrate-binding module for polyethylene terephthalate. *Science of the Total Environment*, 870:161948, 2023.

[176] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[177] G. Rizzo, N. Fanizzi, and C. d'Amato. Class expression induction as concept space exploration: From dl-foil to dl-focl. *Future Generation Computer Systems*, 2020.

[178] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[179] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[180] Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(2):1–49, 2021.

[181] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5): 206–215, 2019.

[182] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

[183] Omer Sagi and Lior Rokach. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1249, 2018.

[184] Abhaya Kumar Sahoo, Chittaranjan Pradhan, and Himansu Das. Performance evaluation of different machine learning methods and deep-learning based convolutional neural network for health decision making. *Nature inspired computing for data science*, pages 201–212, 2020.

[185] Md. Kamruzzaman Sarker and Pascal Hitzler. Efficient concept induction for description logics. In *AAAI*, pages 3036–3043. AAAI Press, 2019.

[186] DR Sarvamangala and Raghavendra V Kulkarni. Convolutional neural networks in medical image understanding: a survey. *Evolutionary intelligence*, 15(1):1–22, 2022.

[187] Stefan Schulz, Kornél Markó, and Boontawee Suntisrivaraporn. Formal representation of complex snomed ct expressions. In *BMC medical informatics and decision making*, volume 8, pages 1–6. Springer, 2008.

[188] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[189] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[190] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: visual explanations from deep networks via gradient-based localization. *International journal of computer vision*, 128:336–359, 2020.

[191] Jude W Shavlik. Combining symbolic and neural learning. *Machine Learning*, 14: 321–331, 1994.

[192] Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M Summers. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285–1298, 2016.

[193] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[194] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *NIPS*, pages 4077–4087, 2017.

[195] Ashwin Srinivasan. The aleph manual. *University of Oxford, Dept. Computer Science*, 2001.

[196] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[197] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2019.

[198] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[199] Shohei Taniguchi, Keno Harada, Gouki Minegishi, Yuta Oshima, Seong Cheol Jeong, Go Nagahara, Tomoshi Iiyama, Masahiro Suzuki, Yusuke Iwasawa, and Yutaka Matsuo. Adopt: Modified adam can converge with any $beta\_2$ with the optimal rate. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[200] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[201] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[202] Tijmen Tieleman. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26, 2012.

[203] Pallavi Tiwari, Bhaskar Pant, Mahmoud M Elarabawy, Mohammed Abd-Elnaby, Noor Mohd, Gaurav Dhiman, and Subhash Sharma. Cnn based multiclass brain tumor detection using medical imaging. *Computational Intelligence and Neuroscience*, 2022 (1):1830010, 2022.

[204] Gabriele Tolomei, Fabrizio Silvestri, Andrew Haines, and Mounia Lalmas. Interpretable predictions of tree-based ensembles via actionable feature tweaking. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 465–474, 2017.

[205] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[206] An C Tran, Jens Dietrich, Hans W Guesgen, and Stephen Marsl. Parallel symmetric class expression learning. *Journal of Machine Learning Research*, 18(64):1–34, 2017.

[207] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR, 2016.

[208] Dmitry Tsarkov and Ian Horrocks. Fact++ description logic reasoner: System description. In *International joint conference on automated reasoning*, pages 292–297. Springer, 2006.

[209] Lewis Tunstall, Leandro Von Werra, and Thomas Wolf. *Natural language processing with transformers*. " O'Reilly Media, Inc.", 2022.

[210] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[211] Giulia Vilone and Luca Longo. Notions of explainability and evaluation approaches for explainable artificial intelligence. *Information Fusion*, 76:89–106, 2021.

[212] Warren J Von Eschenbach. Transparency and the black box problem: Why we do not trust ai. *Philosophy & Technology*, 34(4):1607–1622, 2021.

[213] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. HAT: hardware-aware transformers for efficient natural language processing. In *ACL*, pages 7675–7688. Association for Computational Linguistics, 2020.

[214] Peifeng Wang, Jialong Han, Chenliang Li, and Rong Pan. Logic attention based neighborhood aggregation for inductive knowledge graph embedding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7152–7159, 2019.

[215] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 2017.

[216] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, pages 1–26, 2020.

[217] Shaochen Wang, Zhangli Zhou, and Zhen Kan. When transformer meets robotic grasping: Exploits context for efficient grasp detection. *IEEE Robotics Autom. Lett.*, 7 (3):8170–8177, 2022.

[218] Wu Wang, Junho Lee, Fouzi Harrou, and Ying Sun. Early detection of parkinson's disease using deep learning and machine learning. *IEEE Access*, 8:147635–147646, 2020.

[219] Z. Wang, J. Li, Z. LIU, and J. TANG. Text-enhanced representation learning for knowledge graph. In *Proc. of IJCAI*, 2016.

[220] Lisheng Wei, Kun Ding, and Huosheng Hu. Automatic skin cancer detection in dermoscopy images based on ensemble lightweight deep learning network. *IEEE Access*, 8:99633–99647, 2020.

[221] Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. *ArXiv*, abs/1711.09576, 2017. URL `https://api.semanticscholar.org/CorpusID:40079179`.

[222] Lilian Weng. Attention? attention! *lilianweng.github.io*, 2018. URL `https://lilianweng.github.io/posts/2018-06-24-attention/`.

[223] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[224] Patrick Westphal, Lorenz Bühmann, Simon Bin, Hajira Jabeen, and Jens Lehmann. Sml-bench–a benchmarking framework for structured machine learning. *Semantic Web*, 10(2):231–245, 2019.

[225] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *EMNLP (Demos)*, pages 38–45. Association for Computational Linguistics, 2020.

[226] Wilson Wong, Wei Liu, and Mohammed Bennamoun. Ontology learning from text: A look back and into the future. *ACM computing surveys (CSUR)*, 44(4):1–36, 2012.

[227] Jianxin Wu. Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, 5(23):495, 2017.

[228] Y. Wu, M. Schuster, Z. Chen, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[229] Ruobing Xie, Zhiyuan Liu, Jia Jia, Huanbo Luan, and Maosong Sun. Representation learning of knowledge graphs with entity descriptions. In *AAAI*, pages 2659–2665. AAAI Press, 2016.

[230] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Product knowledge graph embedding for e-commerce. In *Proceedings of the 13th international conference on web search and data mining*, pages 672–680, 2020.

[231] Peng Xu, Xiatian Zhu, and David A. Clifton. Multimodal learning with transformers: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(10):12113–12132, 2023.

[232] Shawn Xu, Subhashini Venugopalan, and Mukund Sundararajan. Attribution in scale and space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9680–9689, 2020.

[233] Zhentao Xu, Mark Jerome Cruz, Matthew Guevara, Tie Wang, Manasi Deshpande, Xiaofeng Wang, and Zheng Li. Retrieval-augmented generation with knowledge graphs for customer service question answering. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2905–2909, 2024.

[234] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9:611–629, 2018.

[235] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR (Poster)*, 2015.

[236] Guangyu Robert Yang and Xiao-Jing Wang. Artificial neural networks for neuroscientists: a primer. *Neuron*, 107(6):1048–1070, 2020.

[237] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *ACM Transactions on Knowledge Discovery from Data*, 18(6):1–32, 2024.

[238] Ibrar Yaqoob, Latif U Khan, SM Ahsan Kazmi, Muhammad Imran, Nadra Guizani, and Choong Seon Hong. Autonomous driving cars in smart cities: Recent advances, requirements, and challenges. *IEEE Network*, 34(1):174–181, 2019.

[239] Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. Qa-gnn: Reasoning with language models and knowledge graphs for question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 535–546, 2021.

[240] A Yoganandhan, SD Subhash, J Hebinson Jothi, and V Mohanavel. Fundamentals and development of self-driving cars. *Materials today: proceedings*, 33:3303–3310, 2020.

[241] Dongran Yu, Bo Yang, Dayou Liu, Hui Wang, and Shirui Pan. A survey on neural-symbolic learning systems. *Neural Networks*, 2023.

[242] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J. Smola. Deep sets. In *NIPS*, pages 3391–3401, 2017.

[243] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. Collaborative knowledge base embedding for recommender systems. In *KDD*, pages 353–362. ACM, 2016.

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

[245] Qian Zhang, Jie Lu, and Yaochu Jin. Artificial intelligence in recommender systems. *Complex & Intelligent Systems*, 7:439–457, 2021.

[246] Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion knowledge graph embeddings. In *Advances in Neural Information Processing Systems*, pages 2731–2741, 2019.

[247] Haitao Zhao, Shaoyuan Sun, and Bo Jin. Sequential fault diagnosis based on LSTM neural network. *IEEE Access*, 6:12929–12939, 2018.

[248] Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. Multilingual machine translation with large language models: Empirical results and analysis. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2765–2781, 2024.

[249] Xiubin Zhu, Dan Wang, Witold Pedrycz, and Zhiwu Li. Fuzzy rule-based local surrogate models for black-box model explanation. *IEEE Transactions on Fuzzy Systems*, 31(6):2056–2064, 2022.

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