LEARNING TO LEARN
CONCEPT DESCRIPTIONS

Giulio Petrucci

Advisor
Dr. Marco Rospocher
Fondazione Bruno Kessler-IRST

Co-Advisor
Dr. Chiara Ghidini
Fondazione Bruno Kessler-IRST

Committee
Prof. Dr. Philipp Cimiano
Bielefeld University
Prof. Dr. Simone Paolo Ponzetto
Universität Mannheim
Prof. Dr. Roberto Basili
Università degli Studi di Roma “Tor Vergata”

January 2018
“L’ape è un insetto con sei zampe.”
— Giulio Petrucci, 1985 ca.

To my family.
This Ph.D. thesis has been developed within the ICT International Doctoral School of University of Trento in partnership with Fondazione Bruno Kessler (FBK). In particular, the author carried out his Ph.D. research activities at the Data and Knowledge Management (DKM) and the Process and Data Intelligence (PDI) groups in FBK, within the FBK Ph.D. International Programme.

This research has been partially carried out with computational resources granted by the Microsoft Azure for Research Award programme.
Abstract

The goal of automatically encoding natural language text into some formal representation has been pursued in the field of Knowledge Engineering to support the construction of Formal Ontologies. Many State-of-the-Art methods have been proposed for the automatic extraction of lightweight Ontologies and to populate them. Only few have tackled the challenge of extracting expressive axioms that formalize the possibly complex semantics of ontological concepts.

In this thesis, we address the problem of encoding a natural language sentence expressing the description of a concept into a corresponding Description Logic axiom. In our approach, the encoding happens through a syntactic transformation, so that all the extralogical symbols in the formula are words actually occurring in the input sentence. We followed the recent advances in the field of Deep Learning in order to design suitable Neural Network architectures capable to learn by examples how to perform this transformation.

Since no pre-existing dataset was available to adequately train Neural Networks for this task, we designed a data generation pipeline to produce datasets to train and evaluate the architectures proposed in this thesis. These datasets provide therefore a first reference corpus for the task of learning concept description axioms from text via Machine Learning techniques, and are now available for the Knowledge Engineering community to fill the pre-existing lack of data.

During our evaluation, we assessed some key characteristics of the ap-
proach we propose. First, we evaluated the capability of the trained models to generalize over the syntactic structures used in the expression of concept descriptions, together with the tolerance to unknown words. The importance of these characteristics is due to the fact that Machine Learning systems are trained on a statistical sample of the problem space, and they have to learn to generalize over this sample in order to process new inputs. In particular, in our scenario, even an extremely large training set is not able to include all the possible ways a human can express the definition of a concept. At the same time, part of the human vocabulary is likely to fall out of the training set. Thus, testing these generalization capabilities and the tolerance to unknown words is crucial to evaluate the effectiveness of the model. Second, we evaluated the improvement in the performance of the model when it is incrementally trained with additional training examples. This is also a pivotal characteristic of our approach, since Machine Learning-based systems are typically supposed to continuously evolve and improve, on the long term, through iterative repetitions of training set enlargements and training process runs. Therefore, a valuable model has to show the ability to improve its performance when new training examples are added to the training set.

To the best of our knowledge, this work represents the first assessment of an approach to the problem of encoding expressive concept descriptions from text that is entirely Machine Learning-based and is trained in a end-to-end fashion starting from raw text. In detail, this thesis proposes the first two Neural Network architectures in literature to solve the problem together with their evaluation with respect to the above pivotal characteristics, and a first dataset generation pipeline together with concrete datasets.
Keywords:

Ontology Learning, Neural Networks, Natural Language Processing, Semantic Web
Acknowledgments

Alla fine di questa incredibile avventura—tornare a scuola a 35 anni non è cosa da poco—mi trovo ancora una volta, per mia enorme fortuna, a dovere e volere esprimere un debito di gratitudine verso tutte quelle persone che hanno condiviso con me, in modi e tempi differenti, parte di questo viaggio.

Il primo ringraziamento va ai miei due advisor, Dr. Marco Rospocher e Dr. Chiara Ghidini, per avermi costantemente aiutato in questi anni con il loro supporto, i loro ottimi consigli e le loro preziose critiche. Assieme a loro, desidero ringraziare i revisori di questa tesi, Prof. Phillipp Cimiano e Prof. Simone Paolo Ponzetto, per le loro osservazioni, grazie alle quali ho potuto migliorare il mio lavoro.

Durante questi anni, ho avuto la possibilità di incontrare brillanti ricercatori, ognuno dei quali ha dato un contributo alla mia crescita professionale e umana. Ringrazio Stefano Teso, la Prof. Raffaella Bernardi, Audun Vennesland, Mirco Ravanelli e Sara Tonelli per il loro tempo, il loro incoraggiamento e per il loro essere stati continua fonte di ispirazione. Desidero ringraziare anche Fabio Remondino, Domenica Dininno ed Eleonora Grilli per avermi trascinato in un progetto nato quasi per caso, da una chiacchierata sulla mia città.

Un enorme ringraziamento va ai colleghi delle unità DKM e PDI della Fondazione Bruno Kessler: Luciano Serafini, Radim Nedbal, Loris Bozzato, Gaetano Calabrese, Alessandro Daniele. In particolare, ringrazio i miei co-autori Mauro Dragoni e Chiara Di Francescomarino per il loro costante supporto e per avermi permesso di flettere i muscoli e lavorare a interessanti progetti e pubblicazioni che hanno integrato il mio percorso di formazione. Un ringraziamento anche alla mia tesista Erinda Jaupaj per aver messo tutta la sua passione, le sue energie e le sue competenze nel suo lavoro—e tutta la sua pazienza nel sopportare il sottoscritto.
Ringrazio le persone che sono state responsabili della prima tappa del mio percorso in Google, Enrique Alfonseca e Aliaksei Severyn. Grazie a Christian Schuler, Mihai Dorin Istin e Marius Vlad per avermi dato, adesso, l’opportunità di far parte della loro squadra e per essere stati così pazienti e disponibili con me in questi mesi così intensi. Grazie, infine, a Daniil Mirylenka, aiuto determinante per questa nuova avventura, sia dal punto di vista professionale che umano.

Grazie a Rizza Kaye Cases, Sajan Raj Ojha, Rajeev Piyare, Emanuele Sansone e Stefano Leucci per essere stati esempi di impegno e dedizione; grazie anche per avermi ricordato che, di tanto in tanto, c’è solo bisogno di sedersi a tavola. Grazie a Mario De Nisi per essere stato il primo volto amico della mia nuova città. Grazie a Marco Fossati per le discussioni continue a mensa, sotto a un palco, davanti a una birra. Grazie a Francesco Corcoglioniti per i preziosi consigli e per avermi cavato le castagne dal fuoco più di una volta. Grazie a Eleonora Mencarini per le sue parole, forse senza volerlo, profonde e ispirate. Grazie ad Anna Feltracco per la sua saggezza, le sue risate e per essere sempre stata capace di tirarmi su di morale. Grazie a Greta Adamo per le discussioni eclettiche che mi hanno ricongiunto a uno dei grandi amori degli anni passati: la Metafisica. Grazie ad Annalisa Armani e Alessandra Frongia (e Riccardo de Pretis, ovviamente) per avermi aiutato a sentirmi così bene qui-e-ora. Grazie ad Alessio Coletta per aver condiviso con me l’equivalente di un mese—e forse più—in una caverna. Grazie al compagno di trincea Ivan Donadello, insostituibile controparte nelle circostanze favorevoli come in quelle avverse. Grazie a Riccardo De Masellis per essere stato un continuo supporto e una continua fonte di ispirazione, non solo dal punto di vista professionale. Mi avevano detto che un dottorato è molto più che modelli, formule, numeri, esperimenti e articoli. Ora posso dirlo: è proprio così. Grazie per essere stati miei amici.

Tante altre persone, al di fuori del contesto lavorativo, mi hanno aiutato
in questa avventura, da distanze e prospettive diverse. Il primo ringraziamento, anche se non-personale, va alla città di Trento alla quale, una volta lasciata la mia amatissima Ascoli Piceno, ho assegnato il compito più difficile: essere casa. E ci è riuscita, nel migliore dei modi possibile: il cielo azzurro, l’aria limpida, la cornice delle montagne e il mormorio dell’Adige resteranno per sempre parte di me. Parlando di Trento, un enorme ringraziamento va alla Dottoressa (sia nel senso di PhD che di medico) Raffaela Baiocchi per essere stata una amica incredibile e una impareggiabile rete di sicurezza. Grazie a Mattia Maistri, Gabriele Bartoletti, Adriano Mesaroli, James Zanella, Giacomo Bracci, Daniele Della Bona e al Prof. Stefano Zambelli per aver condiviso con me i loro ispirati punti di vista in ambito economico e sociale; grazie anche per gli sforzi che avete messo in campo per realizzare le nostre piccole grandi imprese. Grazie a Zeudi Migliara per i giorni e le notti in biblioteca. Grazie a Maria Grazia Corbelli per i viaggi in macchina Trento-Ascoli e Ascoli-Trento, sempre troppo brevi per le nostre chiacchierate. Grazie alle meravigliose persone, brillanti ricercatori e fantastici ingegneri che ho incontrato a Zurigo e che mi hanno aiutato nei miei primi passi in questa nuova avventura: grazie a Marcos Calvo, Joan Pastor, Anna Potapenko, Guido Davide Dall’Olio, Vittorio Massaro e Tiziana Mancosu. Grazie a Stefano Giannarelli, per l’amicizia profonda, il confronto intenso e le telefonate (mai al di sotto ai 60 minuti) capaci di rimandarmi a una prospettiva sempre nuova sulla realtà. Grazie a Bruno Splendiani per quei giorni a Barcellona, grazie per avermi aiutato a concentrarmi su ciò che è davvero necessario, per il continuo incoraggiamento, per il confronto così profondo e ispirato. Grazie a tutti i miei amici che da Ascoli non hanno mai smesso di supportarmi quando ero a Trento e che hanno sempre trovato il modo di sedersi a tavola con me quando ero a casa. Grazie di cuore a Barbara De Vecchis a cui chiedo ancora pubblicamente scusa per averla lasciata a piedi quasi 5 anni fa. Grazie a Tiziana e Stefano, Valeriano e Stefania, Sara e Massi e ad Alessandra;
grazie a Mario, Mauro ed Evelina, grazie e congratulazioni ai neo-sposi Serafino e Valeria. Grazie ad Amanda Felicetti per le uscite folli e i concerti, grazie a Elisa Marchegiani per non avermi mai perso di vista. Grazie anche a Lindo, Fabrizio e Daniela, Tiziano e Stefania: con voi l’appuntamento è a Febbraio, giusto? Grazie alla mia famiglia allargata: zii, zie, cugini e cugine, e soprattutto a mia nonna Santa per ricordarmi, ogni volta, quanto sia bello tornare a casa. Grazie dal profondo del cuore a Eleonora Angelini per aiutarmi a rileggere il reale con un intelletto e una sensibilità impareggiabili; grazie in particolare per essermi stata così vicina nei faticosi giorni in cui scrivevo la prima versione di queste pagine. Grazie ad Antonio Olmi O.P. per aiutarmi costantemente a riconciliare le mie attitudini, la mia sensibilità e i miei studi in una lettura della realtà che sia la più profonda e completa possibile. Grazie a Madre Maria Sophia Githai O.S.B. per la sua costante vicinanza e per avermi dato ospitalità e ristoro ogni volta che ne avevo bisogno. Grazie a don Paolo Sabatini per il suo paziente aiuto senza il quale, forse, non sarei stato capace di riconoscere il momento favorevole. Infine, grazie a don Bruno Tomasi, capace di inciampare su di me con un tempismo che lascia senza fiato. Un ringraziamento speciale a Gianluca e Valentina: non credo sia possibile esprimere con le parole quello che vorrei dire. Ci provo lo stesso: grazie per esserci sempre, in ogni momento, in ogni situazione. Grazie anche—perdonami perché, sinceramente, non me lo aspettavo nemmeno io—al Caporale Maggiore Alpino Paracadutista Ranger Fabio Comini (9 Maggio 1989 - 21 Maggio 2015): tu non sai quante volte, in questi anni e specie nei momenti difficili, il mio pensiero è tornato ad aggrapparsi al ricordo di quella notte del Febbraio 2014: semplicemente grazie, amico mio.

Ci sono quattro persone a cui devo un ringraziamento particolare. Senza di loro, probabilmente, questa avventura non sarebbe mai iniziata. Grazie al Dr. Andrea Carbini, al Dr. Mariano Pierantozzi, al Dr. Michele Catasta e alla Alma Soror Prof. Lucia Aquilanti. Grazie per essere stati una continua fonte
di ispirazione, grazie per essere stati così spesso le lenti attraverso le quali ho provato a mettere a fuoco la mia esperienza, grazie per avermi aiutato a tenere vivo il mio amore per la scienza e la tecnologia, grazie per avermi ridato fiducia in me stesso quando ero in procinto di iniziare questa avventura e per avermi mostrato che, sì, tutto era possibile e a portata di mano, anche quando non riuscivo nemmeno a pensarlo.

Mamma, Babbo, Alessandro: grazie dal più profondo del mio cuore per ogni parola e ogni silenzio, ogni lacrima e ogni risata. Nulla di tutto questo sarebbe stato possibile senza di voi. Nulla di tutto questo e forse anche di meno.

E infine, ringrazio Dio perché mi ha dato tutte queste persone a cui e per cui essere grato. E perché

“mi ha dato una intelligenza, una volontà, una ragione: ebbene, queste devo adoperarle, tenerle in esercizio, farle funzionare. Se non si adoperano, si arrugginiscono e si finisce per essere nullità, dei terra terra, dei lombrichi che strisciano, senza un’idea buona, geniale, ardita; degli ignavi, a Dio spiacenti e a’ nemici sui.”

Giulio

---

1Dal diario del beato Alberto Marvelli, 23 Agosto 1946. Le ultime parole sono una citazione dal Canto III dell’Inferno di Dante Alighieri.
## Contents

<table>
<thead>
<tr>
<th>Abstract</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 The Context</td>
<td>3</td>
</tr>
<tr>
<td>1.2 The Problem</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Contributions</td>
<td>5</td>
</tr>
<tr>
<td>1.3.1 Suitable Network Architectures</td>
<td>6</td>
</tr>
<tr>
<td>1.3.2 Bootstrap Datasets and Reference Set</td>
<td>8</td>
</tr>
<tr>
<td>1.4 Thesis Outline</td>
<td>9</td>
</tr>
<tr>
<td>2 Background</td>
<td>13</td>
</tr>
<tr>
<td>2.1 An overview on Ontologies</td>
<td>13</td>
</tr>
<tr>
<td>2.1.1 Description Logic Languages</td>
<td>18</td>
</tr>
<tr>
<td>2.1.2 Ontology Engineering Methodologies</td>
<td>25</td>
</tr>
<tr>
<td>2.1.3 Ontology Learning from Text</td>
<td>27</td>
</tr>
<tr>
<td>2.2 An overview on Neural Networks</td>
<td>34</td>
</tr>
<tr>
<td>2.2.1 Nomenclature and Basic Notation</td>
<td>37</td>
</tr>
<tr>
<td>2.2.2 Neural Networks: Basic Concepts</td>
<td>38</td>
</tr>
<tr>
<td>2.2.3 The Learning Process of Neural Networks</td>
<td>43</td>
</tr>
<tr>
<td>2.2.4 Neural Networks and Natural Language Processing</td>
<td>49</td>
</tr>
<tr>
<td>2.2.5 The New Wave of Neural Networks</td>
<td>55</td>
</tr>
<tr>
<td>3 From Definitions to Formulae</td>
<td>57</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----</td>
</tr>
<tr>
<td>3.1 The Problem</td>
<td>57</td>
</tr>
<tr>
<td>3.1.1 First Challenge: Choosing the Source Language</td>
<td>58</td>
</tr>
<tr>
<td>3.1.2 Second Challenge: Choosing the Target Language</td>
<td>60</td>
</tr>
<tr>
<td>3.2 The Approach</td>
<td>61</td>
</tr>
<tr>
<td>3.2.1 First Challenge: the Dataset</td>
<td>64</td>
</tr>
<tr>
<td>3.2.2 Second Challenge: the Architecture</td>
<td>65</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4 Dataset</th>
<th>67</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Motivations and Desiderata</td>
<td>67</td>
</tr>
<tr>
<td>4.2 Data Generation Process</td>
<td>70</td>
</tr>
<tr>
<td>4.3 Discussion</td>
<td>74</td>
</tr>
<tr>
<td>4.4 Collecting Manually Built Examples</td>
<td>76</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5 Tagging and Transduction</th>
<th>79</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 The Intuition</td>
<td>80</td>
</tr>
<tr>
<td>5.2 Tagging</td>
<td>82</td>
</tr>
<tr>
<td>5.3 Transduction</td>
<td>85</td>
</tr>
<tr>
<td>5.4 Evaluation</td>
<td>87</td>
</tr>
<tr>
<td>5.4.1 Research Questions</td>
<td>87</td>
</tr>
<tr>
<td>5.4.2 Evaluation Metrics</td>
<td>89</td>
</tr>
<tr>
<td>5.4.3 Datasets</td>
<td>91</td>
</tr>
<tr>
<td>5.4.4 Experimental Settings and Results</td>
<td>96</td>
</tr>
<tr>
<td>5.5 Discussion</td>
<td>99</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6 Translation</th>
<th>105</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 The Intuition</td>
<td>107</td>
</tr>
<tr>
<td>6.2 Architecture Description</td>
<td>108</td>
</tr>
<tr>
<td>6.2.1 The Decoder</td>
<td>109</td>
</tr>
<tr>
<td>6.2.2 The Attention Module</td>
<td>111</td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Syntax and semantics for $\mathcal{AL}$ and some extensions.</td>
<td>25</td>
</tr>
<tr>
<td>4.1</td>
<td>Manually curated dataset and splits.</td>
<td>77</td>
</tr>
<tr>
<td>5.1</td>
<td>Patterns for Concept placeholders filling.</td>
<td>94</td>
</tr>
<tr>
<td>5.2</td>
<td>Network hyperparameters.</td>
<td>97</td>
</tr>
<tr>
<td>5.3</td>
<td>Training parameters.</td>
<td>97</td>
</tr>
<tr>
<td>5.4</td>
<td>Accuracy on the test set.</td>
<td>97</td>
</tr>
<tr>
<td>6.1</td>
<td>Sentence and formula encoding.</td>
<td>123</td>
</tr>
<tr>
<td>6.2</td>
<td>Network settings and hyperparameters for the closed vocabulary evaluation.</td>
<td>124</td>
</tr>
<tr>
<td>6.3</td>
<td>Results for the closed vocabulary evaluation.</td>
<td>124</td>
</tr>
<tr>
<td>6.4</td>
<td>Results for the open vocabulary evaluation.</td>
<td>126</td>
</tr>
<tr>
<td>6.5</td>
<td>Evaluation with manual training set extension of 75 examples for different bootstrap datasets. The models are tested against 425 examples. CF indicates the number correctly translated sentences.</td>
<td>127</td>
</tr>
<tr>
<td>6.6</td>
<td>Comparison of the performances of the proposed approach against two baselines</td>
<td>129</td>
</tr>
</tbody>
</table>
List of Figures

2.1 TBox with terminological axioms about parental relationships. ........................................... 22
2.2 The Ontology Learning Layer Cake and its application to a textual description. ........................................... 29
2.3 Biological neuron and its connections. The picture is taken from Wikipedia. ........................................... 39
2.4 An artificial neuron with three input signals (plus the bias signal). ........................................... 40
2.5 The sigmoid and the hyperbolic tangent functions. ........................................... 41
2.6 A layered architecture with 3 inputs, 4 hidden neurons (in red) and 2 output neurons (in blue). ........................................... 42
2.7 Single neuron architecture with error signal (in red) flowing back on the j-th weight. ........................................... 44
2.8 Error backpropagation in a multi-layer architecture focused on the j-th neuron of the l-th layer. ........................................... 46
2.9 Gated Recursive Unit ........................................... 54
4.1 The data generation pipeline ........................................... 72
4.2 Sampling over the universe of language used for definitions with grammar generated examples ........................................... 75
5.1 Sentence into Formula through Tagging and Transduction. ........................................... 81
5.2 Network architecture for the Tagging task ........................................... 84
5.3 Network architecture for the Transduction task ........................................... 86
5.4 Data generation pipeline for the Tagging and Transduction tasks. ................................................. 93

6.1 Translation of a sentence into a formula by mean of syntactic transformation. ............................................. 108

6.2 Full network architecture, decoding the \( j \)-th symbol. Arrows represent variables, scalars or vectors, flowing between the different modules. Dashed arrows represent variables flowing from the previous timestep. ............................................. 110
Chapter 1

Introduction

In Computer Science, Formal Ontologies (see [34]) are computational artifacts that allow to represent knowledge about a portion of reality in a formal and unambiguous way. A Formal Ontology (hereafter Ontology) is intended to describe a domain of interest in term of involved concepts, individuals, and relations among them. The well-known definition from [79] describes an Ontology as “a formal representation of a shared conceptualization.” Being based on formal languages, Ontologies provide also inference capabilities so that a software agent can perform reasoning activities about the described domain, gaining new knowledge about it.

When a company, an institution, or an organization decides to develop a new Ontology to obtain a machine-readable representation of some domain of interest, the objectives pursued with this development must be carefully assessed. Indeed, as clearly remarked in [81], authoring, managing and maintaining an Ontology

“does neither come by coincidence nor does it come for free.”

The process of encoding human knowledge into an Ontology (see [81]) typically happens in different stages, each of which involves several human actors, with different skills, in different roles, and devoted to different tasks. These actors have to mutually interact during this process. Knowledge Engineers
have typically to interview Domain Experts in order to gain the meaningful knowledge to be encoded. They have to specify the scope and boundaries of the domain of interest and to organize the acquired knowledge in an informal manner. After that, they have to represent such knowledge in a formal way, so that concepts, individuals, and relations actually holding in the domain of interest are effectively captured. As a consequence, this process can be costly, time-consuming, and error-prone.

In addition to interviews to Domain Experts, some knowledge on the domain of interest is often available to Knowledge Engineers in the form of documents like glossaries, technical specifications, and so on. The capability of leveraging such textual sources by automatically extracting formal knowledge from them could significantly speed up the Ontology engineering process and dramatically reduce its cost. This advantage could be even more significant in a scenario, like the one defined by the World Wide Web, where the mass of available documents is constantly increasing in term of volume and variety. Starting from this observation, the Knowledge Engineering community put a significant effort in developing methodologies, techniques and tools in order to automatically—or semi-automatically—extract formal knowledge from unstructured sources, such as text corpora.

Several approaches have been proposed along the years to exploit textual resources for extracting knowledge. These approaches can be classified into two main groups. The first group is focused on populating existing Ontologies with new knowledge about individuals and is referred to as Ontology Population. The second group comprises all the approaches with the goal of creating a new Ontology and is referred to as Ontology Learning (see [17]). The work presented in this thesis falls in this second field. More in detail, we focus on the extraction of complex concept descriptions from natural language text.

In Section 1.1 we set the context of our investigation, briefly overviewing
the research field of Ontology Learning from text. In Section 1.2 we state the main goal of our investigation. In Section 1.3 we describe the main contributions of our work, while in Section 1.4 we describe the structure of this thesis.

1.1 The Context

Different approaches towards Ontology Learning can rely on different techniques. Moreover, they can aim at extracting different types of formal knowledge. Some approaches (see [66, 45, 23, 91]) are tailored to extract the textual surface realization of the main concepts involved in the domain description. Other approaches have the goal of collecting the different textual forms of the same concept together, or to identify intensional concepts definitions (see [66, 83, 18, 61, 74]). Increasing expressiveness of the target representation, hierarchical (see [40, 76, 27, 48, 84]) or conceptual (see [97, 73, 13, 1, 39, 96]) relations among concepts can be extracted as well. The highest degree of expressiveness that we can target is the one where complex concept descriptions are extracted from text (see [88, 67, 37, 82, 68, 50]).

The investigation we carried out in this thesis started with a deep exploration of the State-of-the-Art approaches in Ontology Learning from text. We noticed how the most notable examples in this field have been proposed in the first years of the 2000s, while more recently the whole community was experiencing some kind of a shortcoming. This exploration revealed, even if ten years have passed, a situation which can still be described quoting the authors of [88]:

“state-of-the-art in lexical Ontology learning is able to generate ontologies that are largely informal or lightweight ontologies in the sense that they are limited in their expressiveness”

Focusing in particular on the task of extracting concepts descriptions from
1.2. THE PROBLEM

The main research goal of our investigation was to design and evaluate a system capable to translate the description of a concept expressed in natural language into a corresponding concept class description in some Description Logic language (see [5]). The translation is performed through a syntactic transformation of the text, like in the other State-of-the-Art works presented before. The application of the proper transformation rules is not intended to
be manually encoded but *learnt by examples*, exploiting Machine Learning techniques.

Addressing this goal, we followed recent advances in Recurrent Neural Networks in order to find a suitable network architecture to accomplish the task we wanted to tackle. Moreover, we put a significant effort in obtaining the datasets needed to train and evaluate the models.

### 1.3 Contributions

The research goal as described in Section 1.2 is vast and generic, and encompasses the work that can be achieved during a Doctoral Course. In order to effectively address our goal in the Ph.D., we had to first define precise scope and boundaries to the problem in terms of source and target language.

We had to design a proper Neural Network architecture that is capable to handle that particular portion of natural language that is used to express concept definitions. Together, we had to find proper datasets to train and evaluate the models under investigation.

Regarding the evaluation process, we had to figure out both *what* to evaluate and *how* to evaluate it. First, we evaluated the models with respect to their generalization capabilities. Indeed, even an extremely large training set will not be able to cover all the ways a human express the definition of a concept. Likewise, some portion of the human vocabulary will fall outside the training data. For this reason, the network must learn how to generalize over the training data so that new syntactic realizations and unknown words can be tolerated.

We also evaluated the capacity of the model to improve its performances when new training examples are added to the training set. Indeed, we envision the long-term life cycle of our system as a continuous process of acquisition of new knowledge through the collection of new training examples and
iterative repetition of the training process over the enlarged training set.

This considerable effort has been necessary since no other approach presented in the literature tackled the same problem in a similar way and there is no publicly available and accepted benchmark. To the best of our knowledge, indeed, no other approach has been proposed to solve this task that is entirely Machine Learning-based, that targets the same Description Logic language, and that is trained in an end-to-end fashion. With respect to the last point, we remark that we use just raw text as input, without any additional features—as an example, additional features could be the result of Natural Language Processing toolkits performing Part-of-Speech tagging, syntactic parsing, and so on, which are largely used in rule-based approaches. From this standpoint, our work can be a trailblazer work for the whole community, showing that a purely Machine Learning-based approach to the problem can be profitable.

In the remainder of this Section, we elaborate more on the two main contributions of this thesis: the network architectures that we investigated (in 1.3.1) and the dataset that we used to train and evaluate these architectures (in 1.3.2).

1.3.1 Suitable Network Architectures

The first main contribution of our work is in the Neural Network architectures we designed and evaluated during our investigation. From the architectural point of view, our investigation can be represented as a two stage process that closely followed the advances of Neural Networks application to Natural Language Processing tasks. At each stage, different network architectures were evaluated in terms of their generalization capabilities over the input language.

In the first stage of our investigation, we used at-the-time State-of-the-art in Neural Network architectures for Natural Language Processing to design
an architecture made of two different networks, each of which was trained for a different task. Intuitively, one was in charge of extracting a minimal formula template, in terms of involved concepts and roles, while the other one was in charge of mapping text fragments of the sentence representing concepts or roles to the proper position within that template. This first model is presented in Chapter 5. The evaluation process allowed us to assess the capabilities of Recurrent Neural Networks to manage the syntactic complexity of concept descriptions expressed in natural language and to tolerate unknown words. At the same time, the evaluation helped us to highlight some structural limitations of the model with respect to the syntactic complexity of the input sentence and to emphasize the cost of having an architecture composed of two networks (two set of hyperparameters to defined and tuned, two datasets to be collected, two training processes to be run).

In the second stage of our work, we took advantage of further advances in the field or Deep Learning to overcome the limitations of the architecture presented in the first stage. The last result of this work is a single Neural Network that exploits a Pointer Network (see [85]) as the pivot component of the output layer of a Recurrent Encoder-Decoder configuration (see [16]). As well as for the two-networks solution, we evaluated the generalization capabilities of this single-network model both in terms of syntactic structures and vocabulary. Moreover, we evaluated the capability of the model to improve its performance by adding new annotated examples to the training set. From a theoretical point of view, such a network provide a more general solution to the problem, being capable to transform an arbitrary natural language concept description into the corresponding Description Logic formula through syntactic transformation. This second model is described in details in Chapter 6.
1.3.2 Bootstrap Datasets and Reference Set

A second contribution of our work is given by the datasets we used to train and evaluate the models we proposed. Like for any Machine Learning approach, properly training and evaluating a Neural Network requires a collection of examples. Such collection must be a significant sample of the universe of the input language, so that the network can capture its regularities and learn how to generalize over it, in order to correctly process unseen sentences. Moreover, it must contains some unknown words, namely words that the model learns to process relying only on the rest of the sentence acting as a context. To the best of our knowledge, there is no publicly available dataset with these characteristics for the task we were aiming to tackle. As a consequence, we had to build our own datasets to evaluate the model in terms of its capabilities to generalize over different syntactic structures and to tolerate unknown words. We designed a general data generation process that has been implemented in different manners during the two experimental stages of our work to provide training and evaluation data, for each model, in a proper format.

We acknowledge that such datasets are per se a limited approximation of the portion of natural language that is actually used by human agents to express concept descriptions. Though, this is a common situation when a task is proposed for the first time. Moreover, the datasets we used have been proven suitable to assess the validity of the models we investigated with respect to the rationale behind our evaluation process. Moreover, on the long run, the life cycle of a Machine Learning system goes through several training steps. Indeed, as new knowledge is available in the form of further training examples, the training set is usually extended with these examples and the system is trained again. In this long term perspective, the dataset produced so far can be seen as a bootstrap dataset to be continuously augmented with
new training examples, in order to improve the performances of the network. In our evaluation process, we also assessed the capacity of the network to actually improve its performances when new examples are added to the training set, using manually annotated real world examples. We gained evidence that our datasets could serve as valid bootstrap datasets in a scenario where new examples are continuously added to the training set.

The rationale behind the generation process of the bootstrap datasets is presented in Chapter 4. For each of the two architectures we trained and evaluated, different implementations of this process were put in place. They are described in Section 5.4.3 and Section 6.3.3. Together with these datasets, we devise also a manually curated dataset comprising 500 sentences and the corresponding formulae. This manually curated dataset has been used in our evaluation to assess the actual capability of the model under investigation of improving its performance over real world data by adding manually annotated training examples to the automatically generated bootstrap datasets. The manually curated dataset is described in Section 4.4.

1.4 Thesis Outline

The remainder of the present thesis is structured as follows. In Chapter 2 we provide all the necessary background knowledge on Ontologies (Section 2.1)—and in particular on the state of the art in Ontology Learning (in 2.1.3)—and on Neural Networks (Section 2.2). In Chapter 3 we will describe how we envisioned the problem of extracting complex concept description from text and the approach we proposed to tackle the problem. We outline the main challenges and set precise boundaries to our investigation in terms of source and target language. In Chapter 4 we will present the rationale we followed to obtain the examples used to train and validate the different models under investigation. The two different Neural Network architectures we trained and
evaluated are described in Chapter 5 and Chapter 6 in terms of the structure, the research questions we tried to assess during the evaluation process, the different experimental settings, and the obtained results. In Chapter 7 we discuss other approaches in the literature tackling the problem of extracting expressive knowledge from natural language text, remarking the differences among such approaches and the one presented in this work. Finally, Chapter 8 concludes the thesis and outlines some directions for possible future investigation efforts.
CHAPTER 1. INTRODUCTION

1.4. THESIS OUTLINE
Chapter 2

Background

The objective of the work carried out in this thesis is to design, train and evaluate neural architectures for the task of translating the definition of a concept expressed in natural language into a corresponding Description Logic formula. The task as we envisioned it is described in detail in Chapter 3, together with the main challenges it carries, thus setting the scope and boundaries of our investigation. In the present Chapter, we provide the necessary background on Formal Ontologies (henceforth Ontologies) and Neural Networks needed to fully understand the work of this thesis.

2.1 An overview on Ontologies

In Computer Science, we call Ontology an artifact capable to formally encode some knowledge about some piece of reality, providing a machine-understandable representation of all the involved entities, their properties, and their mutual relations. Such representations are intended to provide a formal way to represent knowledge in an unambiguous manner. Ontologies also support the sharing of the represented knowledge among software agents as well as between machines and humans. In addition, Ontologies allow the application of logical inference on the facts that are already known in order to gain new knowledge on the domain of interest. A widely accepted definition
of an Ontology is the one given by the authors of [79], which states that an Ontology is “a formal, explicit specification of a shared conceptualization.”

Let us assume a scenario in which an agent—human or artificial—has some knowledge about the world. Let us also assume that this agent, called AgentA, wants to explicitly represent this knowledge using statements in natural language as follows:

**A1.** Socrates is human;

**A2.** every human is mortal;

**A3.** every human has a birth date.

For any human reader with a basic knowledge of the English language, both in terms of grammar and vocabulary, it is easy to just read statement A1 and figure out Socrates as a particular human being. Then, using the knowledge of what a human being is, the reader could immediately ascribe a proper birth date, possibly a death date, and so on to Socrates. On the opposite, an artificial agent is not supposed to have a precise representation of the concept of human, so that the word “human” in A1 does not have proper semantics. Adding statements A2 and A3 we are defining the semantics of the term “human” as identifying the set of all the entities that are also mortal and have a birth date. We could possibly have to define the semantics of “human” and “birth date” as well but for now we assume that AgentA has knowledge of such concepts.

This small example illustrates one of the prominent goals that Ontologies fulfill: provide a semantics to the symbols describing the piece of reality an intelligent agent wants to deal with. The semantics of these symbols is formally specified through axioms, namely formulæ expressed in some logic language acting as the statement A1, A2, and A3 in our example.

When authoring an Ontology, Knowledge Engineers have to choose a proper formal language that allow the specification of a formal semantics.
Moreover, this formal language must provide Knowledge Engineers with all such primitives that are necessary to actually capture the characteristics of the portion of the world they want to model. The choice of the formal language has crucial implications also from the point of view of the inference activities. Starting from the same knowledge owned by AgentA, any human agent could easily deduce that Socrates is mortal—from A1 and A2—and that he has a birth date—from A2 and A3. The activity of inferring new knowledge starting from a given set of axioms is called reasoning. The formal language we choose for encoding our Ontology must provide AgentA with a proper support for such reasoning activities, so that it can infer the following new axioms:

**A4.** Socrates is mortal

**A5.** Socrates has a birth date

Summarizing, the formalism used for the representation of ontological knowledge, must satisfy two main requirements. First, it must provide primitives and constructs to properly describe the domain of interest. Second, it must provide an inference mechanism that allows to infer new explicit knowledge. Such requirements create, from the computational perspective, a trade-off between expressiveness, needed to describe complex facts about the world, and simplicity, necessary to ensure that all the inference problems are decidable and computed in a reasonable amount of time.

Among the different formal languages that can be used to represent ontological knowledge, Description Logics (see §) are particularly popular. These language are suitable to build descriptions of domains for knowledge-based applications in terms of the involved, concepts, instances and relations among them. They are provided with a formal set-theoretic semantics. Focusing on the knowledge owned by AgentA, terms like “human”, “mortal” and “birth date” denote concepts that can be considered as sets of items: the set
of all the human beings, the set of all the entities that are subject to death, and the set of all the available birth dates, respectively. Statement A1 tells that the individual named Socrates belongs to the set of all humans, or that it is an \textit{instance} of such concept. Statement A2 asserts that the set of all human being is a subset of the mortal entities. Typically, subset/superset relations are called \textit{taxonomical relations} and the term \textit{taxonomy} (or, in alternative, \textit{hierarchy}) is used to denote a collection of such relations. Finally, A3 describes a conceptual relation, or \textit{role}, between the elements of the set of all humans and the elements of the set of all birth dates, according to which, for each individual in the set of all humans, there is a corresponding item in the set of birth dates. We call statements involving only concepts, like A2 and A3, \textit{terminological axioms} and the set of all such axioms is called \textbf{TBox}. Conversely, all the statements such as A1 involving individuals are called \textit{assertions} and are collected into the so called \textbf{ABox}. Terminological axioms and assertions are generally referred to as \textit{axioms}. Description Logic languages and their salient characteristics will be presented in Section \ref{onol}.\textsuperscript{16}

Back to our driving example, once chosen a suitable formal language, it would be easy to actually encode statements A1, A2, and A3. Typically, however, the knowledge about a certain domain to be encoded in an Ontology is wider and more complex that the one that we represented in statements A1, A2 and A3. An overview on the Ontology Engineering process and methodologies will be given in \ref{ontoleng}.

One way to speed up the process of authoring an Ontology could be to exploit some process to automatically extract a formal representation of the knowledge available in textual sources. Though, the intrinsic nature of natural language which is informal, ambiguous, and polysemous, makes this process not straightforward. To overcome this limitation, much effort has been put in this direction and many different approaches have been proposed along the years, tackling the problem of extracting formal knowledge.
CHAPTER 2. BACKGROUND

2.1. AN OVERVIEW ON ONTOLOGIES

from text. All such experiences have been globally referred to as **Ontology Learning** or **Ontology Population** from text, depending if the target of the acquisition process is the TBox or the ABox, respectively. Our work falls in the area of Ontology Learning from text. An overview on the subject is provided in 2.1.3

Before proceeding further, we want to introduce another pivotal characteristic of Ontologies by extending our driving example. Let us assume a second agent, say **AgentB** having some knowledge about reality that can be encoded with the following statements:

**B1.** hemlock\(^1\) is mortal;

**B2.** everything that is mortal, is also dangerous.

Also in this case, any human being with a minimal knowledge of the grammar and the vocabulary of the English language could easily understand that **AgentA** uses the term “mortal” in the sense of “what is subject to death”, while **AgentB** uses the same term to intend that something is “lethal”. **AgentB** lacks such background knowledge, so that, reading statement A2, it could infer that every human is dangerous—and this is, luckily, not the case. The semantics that the two agents give to the symbol “mortal” is different. We say that the two agents commits to different conceptualizations. In a scenario in which these agents have to interoperate, like when **AgentB** reads statement A2, this difference can lead to incoherent conclusions about the world.

Another main advantage in using Ontologies is that they provide an explicit semantics of terms, so that different agents modeling their knowledge using the same Ontology will commit to the same conceptualization. In this perspective, encoding the knowledge of **AgentB** we could have used a different term, say “lethal”. Ontologies can allow different agents to exchange

\(^1\)Hemlock, or *Conium maculatum*, see [https://en.wikipedia.org/wiki/Conium_maculatum](https://en.wikipedia.org/wiki/Conium_maculatum) (last accessed on January 31, 2018), is a poisonous herb.
their knowledge in an unambiguous way, establishing a common semantics. This scenario is called Semantic Interoperability and is another pivotal characteristics of Ontology-based applications. Semantic Interoperability is not a prominent issue to the extent of our work, so it will not be further developed in the remainder of this section.

2.1.1 Description Logic Languages

Description Logic languages (or Description Logics) are a family of formal languages used to represent application domains in term of relevant concepts and entities, their properties, and relations among them. Description Logic languages have been developed to overcome a main limitation of earlier formalism for knowledge representation, such as semantic networks and frames (see \[12\]). Indeed, the former are provided with logic-based formal semantics, while the latter were not.

When we encode the knowledge to describe the domain of an application, we have to distinguish two different kinds of knowledge. The first kind, is the one that settles the **terminology** of the knowledge base, defining all the **concepts** and the **roles**. From a set-theoretic perspective, concepts are interpreted as set of individuals, while roles are interpreted as sets of pairs of individuals, namely relations. As an example, we can define the concept of **Country** to collect all the countries of the world, the concept of **City** to collect all the cities in the world, and the concept of **Metropolis** to denote all the cities with more than ten millions of inhabitants. We could define the role **capitalOf** as the set of pairs composed of a country and its capital city. Concepts such as **Country**, **City**, and **Metropolis**, and roles such as **capitalOf** are called **atomic**, since they are defined in terms of themselves. We can combine such atomic concepts and roles in order to create **complex** concepts or roles, capable to capture a more complex knowledge of the domain. So, we could define the concept of **MetroCapital**
as the set of all the cities that are capital of a state and have more than ten millions of inhabitants. \textit{MetroCapital} is defined in terms of the role \texttt{capitalOf} and the concept \texttt{Metropolis}. All the definitions of concepts and roles are called \textbf{terminological axioms} and collected together into the so called \textbf{TBox}.

The other kind of knowledge we might want to encode is made of \textbf{assertions}, which are particular axioms describing the state of affairs of particular individuals. We could define some individuals like \texttt{Brasilia} and \texttt{Rome} and assert that they are instances of the \texttt{City} concept, and individuals like \texttt{Brasil} and \texttt{Italy} and assert they are instances of the \texttt{Country} concept. Moreover, we can assert that \texttt{Brasilia} is an instance of the \texttt{MetroCapital} concept and the pairs \texttt{(Brasilia, Brasil)} and \texttt{(Rome, Italy)} are in the \texttt{capitalOf} relation. All the assertion of a knowledge base are collected into its \textbf{ABox}.

We use the expression \textbf{Knowledge Base} to denote a set of axioms that describes some domain of interest. A Knowledge Base is typically made of a TBox and an ABox. The complexity and the expressiveness that we can achieve in encoding a Description Logic knowledge base depends on the particular Description Logic language we are exploiting. Below, we illustrate the main Description Logic languages, starting from the basic ones and moving to more complex ones.

\textbf{Attributive Languages}

Attributive Languages (\textit{AL}-languages) are a family of Description Logics introduced in \cite{71} as a minimal language that has some practical interest. More complex Description Logics start from \textit{AL}-languages and extend them with different primitive constructors. Below, we present the syntax and the semantics of \textit{AL}-languages and some of their extensions that are meaningful to the work of this thesis.

Regarding the notation, we will use letters \texttt{A} and \texttt{B} to denote atomic
concepts, letters $C$ and $D$ to denote complex concepts, letters $R$ and $S$ for roles and lowercase letters $a$, $b$, and $c$ for individuals. Other symbols, all of them recapitulated in Table 2.1, will be clarified as they are used. Concept descriptions in $\mathcal{AL}$-languages are formed according to the following syntax rules:

\[
\begin{align*}
C, D &\rightarrow A \quad \text{(atomic concept)} \quad (2.1) \\
\top & \quad \text{(universal concept)} \quad (2.2) \\
\bot & \quad \text{(bottom concept)} \quad (2.3) \\
\neg A & \quad \text{(atomic negation)} \quad (2.4) \\
C \sqcap D & \quad \text{(intersection)} \quad (2.5) \\
\forall R.C & \quad \text{(value restriction)} \quad (2.6) \\
\exists R.\top & \quad \text{(limited existential quantification).} \quad (2.7)
\end{align*}
\]

The formal semantics of an attributive language is defined by introducing an interpretation, which provides a way to compute the set theoretic meaning of a formula. An interpretation is a pair $\mathcal{I} = \langle \Delta^\mathcal{I}, (\cdot)^\mathcal{I} \rangle$. The first element $\Delta^\mathcal{I}$ is a non empty set called interpretation domain, comprising all the entities that exist. The second element $(\cdot)^\mathcal{I}$ is a function, called interpretation function, that grounds the symbols used for individuals, atomic concepts and atomic roles to elements in $\Delta^\mathcal{I}$, according to the following rules:

\[
\begin{align*}
a^\mathcal{I} &\in \Delta^\mathcal{I} \quad (2.8) \\
A^\mathcal{I} &\subseteq \Delta^\mathcal{I} \quad (2.9) \\
R^\mathcal{I} &\subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}. \quad (2.10)
\end{align*}
\]

Individuals are interpreted as elements of the domain, atomic concepts are interpreted as subsets of the domain, and atomic roles are interpreted as binary relations over the domain. We can extend such interpretation function
by means of the following inductive definitions:

\[ \top^\mathcal{I} = \Delta^\mathcal{I} \]  
\[ \bot^\mathcal{I} = \emptyset^\mathcal{I} \]  
\[ (\neg A)^\mathcal{I} = \Delta^\mathcal{I} \setminus A^\mathcal{I} \]  
\[ (C \cap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I} \]  
\[ (\forall R.C)^\mathcal{I} = \{ a \in \Delta^\mathcal{I} \mid \forall b . (a,b) \in R^\mathcal{I} \rightarrow b \in C^\mathcal{I} \} \]  
\[ (\exists R.\top)^\mathcal{I} = \{ a \in \Delta^\mathcal{I} \mid \exists b . (a,b) \in R^\mathcal{I} \} . \]

Finally, we say that two concepts \( C \) and \( D \) are equivalent if \( C^\mathcal{I} = D^\mathcal{I} \) for all the interpretations \( \mathcal{I} \).

Terminological axioms can be classified in two types: inclusion axioms and equality axioms. Inclusion axioms, or inclusions, express a relation of inclusion between concepts (resp. roles), stating that one concept (resp. role) is a sub-concept (resp. sub-role) of another. These axioms are of the form:

\[ C \sqsubseteq D \] (concept inclusion)  
\[ R \sqsubseteq S \] (role inclusion).

We say that an interpretation \( \mathcal{I} \) satisfies an inclusion \( C \sqsubseteq D \) (resp. \( R \sqsubseteq S \)) if \( C^\mathcal{I} \subseteq D^\mathcal{I} \) (resp. \( R^\mathcal{I} \subseteq S^\mathcal{I} \)).

Equality axioms, or equalities, define relations of identity between concepts (resp. roles) stating that one concept (resp. role) is the same of another. Equality axioms have the form:

\[ C \equiv D \] (concept equality)  
\[ R \equiv S \] (role equality).

We say that an interpretation \( \mathcal{I} \) satisfies an equality \( C \equiv D \) (resp. \( R \equiv S \)) if \( C^\mathcal{I} = D^\mathcal{I} \) (resp. \( R^\mathcal{I} = S^\mathcal{I} \)).
To clarify what we defined so far, let us present a simple running example. According to (2.1), we introduce the atomic concept \textit{Human} to denote all the human beings, and the atomic concept \textit{Female} to denote all the female beings. We also introduce the atomic role \textit{hasChild}. Recalling from (2.10) that atomic roles are interpreted as binary relations, we use \textit{hasChild} to denote all the pairs of individuals, where the second one is a child of the first one. Starting from these atomic concepts and roles, we can define complex concepts and creating an expressive TBox with terminological axioms about parental relations. Such TBox is reported in Figure 2.1.

\[
\begin{align*}
\text{Male} & \equiv \neg \text{Female} \\
\text{Woman} & \equiv \text{Human} \sqcap \text{Female} \\
\text{Man} & \equiv \text{Human} \sqcap \text{Male} \\
\text{Parent} & \equiv \text{Human} \sqcap \exists \text{hasChild}. \top \sqcap \forall \text{hasChild.}\text{Human} \\
\text{NonParent} & \equiv \text{Human} \sqcap \forall \text{hasChild.}\bot
\end{align*}
\]

Figure 2.1: TBox with terminological axioms about parental relationships.

To define the concept \textit{Male} as the set of all individuals that are not female, we use the atomic negation, as in (2.4), writing $\neg \text{Female}$. We use the limited existential quantification defined in (2.7) and write $\exists \text{hasChild}. \top$ to indicate set of all the individuals that have a child. We can use value restriction defined in (2.6) to denote the concepts of all the entities having only human children as $\forall \text{hasChild.}\text{Human}$. We can build the complex definition of \textit{Parent} as the set of all individuals that are human beings, have children and all their children are human. Similarly, we can define the concept of \textit{NonParent} as the set of all the human beings not having any children. This second clause can be expressed using another value restriction like $\forall \text{hasChild.}\bot$.

Finally, let $\mathcal{T}$ be a set of axioms, we say that an interpretation $\mathcal{I}$ satisfies $\mathcal{T}$ if and only if it satisfies every axiom in $\mathcal{T}$. We call interpretation $\mathcal{I}$ a model for $\mathcal{T}$, writing $\mathcal{I} \models \mathcal{T}$. 
Extending Attributive Languages

We can add further constructs to \( \mathcal{AL} \), ending up having more expressive languages. Traditionally, in Description Logic, each extension is denoted with a new letter. In the following, we describe those extensions that are relevant to the work illustrated in the thesis hereafter.

\( \mathcal{ALU} \). The **union of concept**, \( U \), is written \( C \sqcup D \) and interpreted as:

\[
(C \sqcup D)^I = C^I \cup D^I
\]  

(2.21)

The resulting Description Logic language is denoted with \( \mathcal{ALU} \).

\( \mathcal{ALE} \). Allowing arbitrary and not only atomic concepts in the right-hand side of the limited existential quantification is possible through the **full existential quantification** operator, denoted with the letter \( \mathcal{E} \), written as \( \exists R.C \) and interpreted as:

\[
(\exists R.C)^I = \{ a \in \Delta^I \mid \exists b . (a,b) \in R^I \land b \in C^I \}. 
\]  

(2.22)

Adding this operator to \( \mathcal{AL} \) generates the \( \mathcal{ALE} \) language.

\( \mathcal{ALC} \). The **negation**, denoted with \( \mathcal{C} \) for complement, has been initially defined in \( \mathcal{AL} \) only for atomic concept, and can be easily extended to an arbitrary concept. It is defined with \( \neg C \) and interpreted as:

\[
\neg C^I = \Delta^I \setminus C^I.
\]  

(2.23)

Note that the semantics defined so far enforces the following equivalences:

\[
C \sqcup D \equiv \neg(\neg C \sqcap \neg D) \]  

(2.24)

\[
\exists R.C \equiv \forall R.\neg C, \]  

(2.25)
meaning that the concept union and the full existential quantification can be expressed using the negation. Therefore, adding \( C \) to \( \mathcal{AL} \) implicitly means also the addition of \( U \) and \( E \). Adding the construct of concept negation to \( \mathcal{AL} \) originates the Description Logic language \( \mathcal{ALC} \), one of the most popular Description Logic languages.

\( \mathcal{ALN} \). The last extensions are about the number restriction in roles. We can have unqualified number restrictions, \( \mathcal{N} \), denoted with \( \geq nR \) and with the following semantics:

\[
\{ a \in \Delta^I \mid |\{ b \in \Delta^I \mid (a,b) \in R^I \}| \geq n \}.
\] (2.26)

The semantics for all the comparison operators \( >, =, <, \) and \( \leq \) can be defined in an analogous way. The notation \( |\cdot| \) indicates the number of elements in a finite set. Adding \( \mathcal{N} \) to \( \mathcal{AL} \) originates \( \mathcal{ALN} \).

\( \mathcal{ALQ} \). Similarly for the existential restriction, we can have a qualified number restriction, \( \mathcal{Q} \), writing \( \geq nR.C \) with the following semantics:

\[
\{ a \in \Delta^I \mid |\{ b \in \Delta^I \mid (a,b) \in R^I \land b \in C^I \}| \geq n \}.
\] (2.27)

Adding \( \mathcal{Q} \) to \( \mathcal{AL} \) originates \( \mathcal{ALQ} \). Note that adding \( \mathcal{Q} \) implies the implicit addition of \( \mathcal{N} \), as the unqualified numbered restriction can be written as \( \geq nR.\top \).

A recap of all operators introduced so far is contained in Table 2.1. Once defined all such extensions, we can denote any language based on \( \mathcal{AL} \) just composing the letters that indicate the sets of primitives that we are using. The reference language for the scope of this work is \( \mathcal{ALCQ} \). This particular Description Logic language has all the primitives of the \( \mathcal{AL} \)-language plus complex concept negation, concept union, qualified existential and numbered.
restrictions. More complex languages with more expressive constructs can be built, but they fall beyond the scope of the current work, as will be clarified in Chapter 3. A detailed description of the various extensions of $AL$-languages can be found in [6].

<table>
<thead>
<tr>
<th>primitive</th>
<th>syntax</th>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal concept</td>
<td>⊤</td>
<td>$\Delta I$</td>
</tr>
<tr>
<td>Bottom concept</td>
<td>⊥</td>
<td>$\emptyset I$</td>
</tr>
<tr>
<td>Atomic concept</td>
<td>$A$</td>
<td>$A^I$</td>
</tr>
<tr>
<td>Concept negation ($C$)</td>
<td>$\neg C$</td>
<td>$\Delta I \setminus C I$</td>
</tr>
<tr>
<td>Concept intersection</td>
<td>$C \cap D$</td>
<td>$C I \cap D I$</td>
</tr>
<tr>
<td>Concept union ($U$)</td>
<td>$C \cup D$</td>
<td>$C I \cup D I$</td>
</tr>
<tr>
<td>Atomic role</td>
<td>$R$</td>
<td>$R^I$</td>
</tr>
<tr>
<td>Value restriction</td>
<td>$\forall R.C$</td>
<td>$a \in \Delta I \mid \forall b . (a,b) \in R^I \rightarrow b \in C^I$</td>
</tr>
<tr>
<td>Limited existential quantification ($E$)</td>
<td>$\exists R.T$</td>
<td>$a \in \Delta I \mid \exists b . (a,b) \in R^I$</td>
</tr>
<tr>
<td>Full existential quantification ($E$)</td>
<td>$\exists R.C$</td>
<td>$a \in \Delta I \mid \exists b . (a,b) \in R^I \land b \in C^I$</td>
</tr>
<tr>
<td>Unqualified numbered restriction ($N$)</td>
<td>$\geq nR$</td>
<td>$a \in \Delta I \mid {b \in \Delta I \mid (a,b) \in R^I} \geq n$</td>
</tr>
<tr>
<td>Qualified numbered restriction ($Q$)</td>
<td>$\geq nR.C$</td>
<td>$a \in \Delta I \mid {b \in \Delta I \mid (a,b) \in R^I \land b \in C^I} \geq n$</td>
</tr>
</tbody>
</table>

Table 2.1: Syntax and semantics for $AL$ and some extensions.

### 2.1.2 Ontology Engineering Methodologies

As described in [81], the whole process of encoding an Ontology and rolling it out in a productive ecosystem is a complex process. Indeed, the objective of the development must be carefully assessed since the whole process can require a significant effort. Along the years, many different methodologies have been proposed in order to support Knowledge Engineers and the other involved stakeholders in the several activities of the Ontology Engineering process—see [81,29,80,35] just to name a few notable examples. These methodologies have to be considered more as general guidelines than as procedures to be strictly followed. They also differentiate depending on the particular scenario or context—e.g., if the Ontology development happens in a heterogeneous and distributed context or within a close group of people, if it insists only on a single piece of a system domain to be modeled or on a wide organization-level domain, and so on. Regardless of the differences, we
can still identify some traits that are common to different methodologies and more specifically to the different phases that are intended to happen through the process.

First, the scope and boundaries of the Ontology must be defined. In this phase, the domain that the Ontology will cover must be effectively specified, together with the intended use of the Ontology.

Then, a first conceptualization with intermediate and often informal representations such as text, diagrams, specification documents, and so on, is performed. The typical output of this second phase can be considered as a semi-formal description of the Ontology, sketching the planned area of the ontology application and listing valuable sources from which to gather the knowledge that will have to be encoded.

Subsequently, the formalization phase will turn such semi-formal description into a proper Formal Ontology. This process requires Knowledge Engineers to interview Domain Experts and possibly scan through existing document bases, in order to gain a meaningful terminology and description of the domain. During the encoding process, other specific fine-grained guidelines can be followed (see [2]).

After the formalization phase, the Ontology must be evaluated under several perspectives: if it fits the syntactic and/or semantics constraints, if it can actually be exploited by all the other actors in the environment of the organization, and so on. If the evaluation is negative, a new refinement phase is needed, and so on, until the evaluation is positive.

At this point, the Ontology can be rolled-out into production. From this moment on, the Ontology will likely evolve, according to the actual needs of the involved actors.

Note that the phases described above are not intended to happen in a waterfall scheme but, especially for the last three, they are likely to happen iteratively.
Even from this brief discussion, it is easy to understand that the whole process can require a significant effort, in particular because of the involvement of several human actors in the different stages. In order to reduce this cost, the scientific community has been pursuing the ambitious goal of automating large part of this process. One way that has been explored is the possibility of leveraging already existing text corpora describing the domain of interest and automatically turn them into Formal Ontologies. This field has been named Ontology Learning from Text and it will be discussed in 2.1.3.

2.1.3 Ontology Learning from Text

The task of encoding human knowledge into a formal representation (such as an Ontology) is a crucial step in the development of Semantic Web based applications. Well established methodologies (see [81]) heavily rely on a number of manual activities, performed by Knowledge Engineers, tailored to collect that portion of knowledge which is significant with respect to the domain at hand. Examples of these activities are interviews to Domain Experts and/or the selection of knowledge from existing, often textual, sources such as technical specifications, glossaries, and encyclopaedic entries. Such knowledge is then manually encoded into a formal representation. Manually eliciting and encoding knowledge can be extremely costly and time-consuming, especially for applications where knowledge sources are continuously increasing in volume and variety. This issue is well known in the literature and contributes to the so-called Knowledge Acquisition Bottleneck (see [62]).

Conversely, a solution to this problem already envisioned in literature is the development of some system capable to process such unstructured textual sources and automatically—or semi-automatically—encode them into some formal language. This could significantly relieve part of the burden from human operators. Trying to pursue this ambitious goal, the Knowledge
Engineering community put a significant effort in tackling such problem from different angles and perspectives. Many approaches have been proposed along the years to extract structured knowledge from unstructured text corpora. All such methods can be grouped into two main categories. The first one collects all the approaches aiming to formalize knowledge about concepts and relations among concepts, while the second comprises all the approaches with the goal of extracting knowledge about individuals, their mutual relations, and their classification within concepts. We refer to the latter category as **Ontology Population** and to the former as **Ontology Learning**. Hereafter we focus only on Ontology Learning as it is the one relevant to the scope of this thesis.

Ontology Learning methods build upon well-established techniques from several disciplines like Natural Language Processing, Machine Learning, Information such methods have been organized into the so called **Ontology Learning Layer Cake** (or just Layer Cake), which is a conceptual sketch of a generic Ontology Learning layered architecture, described in [17]. In this architecture, each layer is associated with a task. Layers are sorted in ascending order, according to the complexity of the task they implements, with the most basic at the bottom and the most complex at the top. Ideally, in this formalization, the output of each layer is the input for the one on the top of it. We want to remark how this architecture is *not* meant to constrain the Ontology Learning process in a strict pipeline. Indeed, it has to be intended some sort of driving criterion to organize the many efforts put by community in developing Ontology Learning approaches.

The whole architecture is depicted in Fig. 2.2, where a small example text corpus is reported together with the corresponding output for each layer. Below, the tasks associated to each layer are described and reference to notable examples are given.
“Many people work in a hospital. They can be doctors, nurses or janitors. In a hospital, patients are people with some disease. Each doctor can cure one or more patient. Each illness have some particular symptoms. Patients are hospitalized in different departments, depending on their disease. Dr. Robert Kelso, is a doctor and he works at the Sacred Heart Hospital.”

<table>
<thead>
<tr>
<th>Task</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axiom Extraction</td>
<td>Doctor ( \equiv \geq 0 \text{cure.Patien}t )</td>
</tr>
<tr>
<td>Relation Extraction</td>
<td>cure(Doctor, Disease)</td>
</tr>
<tr>
<td>Hierarchy Extraction</td>
<td>is_a(Doctor, Person)</td>
</tr>
<tr>
<td>Concept Extraction</td>
<td>Disease</td>
</tr>
<tr>
<td>Synonym Extraction</td>
<td>{disease, illness}</td>
</tr>
<tr>
<td>Term Extraction</td>
<td>disease, illness, doctor</td>
</tr>
</tbody>
</table>

Figure 2.2: The Ontology Learning Layer Cake and its application to a textual description.

**Term Extraction**

The goal of this task is to identify all the relevant terms in the text corpus. Frequency based criteria, metrics from Information Retrieval like Term Frequency (TF), Term Frequency-Inverse Document Frequency (TF-IDF) or more task-specific ones like Terminology Identification Measure-Domain Relevant Measure (TIM-DRM), have been used in several approaches such as the ones described in [66, 26, 84, 45, 42, 23, 18], together with the C-value/NC-value metric from Computational Linguistics, which considers also the term context. Linguistic features are generally exploited in approaches such as the ones described in [18, 42, 66, 74]. In [97], all these criteria are combined into a more complex approach in which the extracted terms are connected as nodes in a graph structure called Conceptual Model. The graph topology is built exploiting syntactic patterns. Terms are then ranked and filtered on the basis of metrics from Graph Theory (such as Betweenness, Centrality, Degree, HITS, PageRank) or Information Retrieval (such as TF, TF-IDF). Rankings are then combined according to several voting schemes (Majority,
Intersection, and so on) in order to select only those terms which are actually relevant. The solution presented in [45] combines heuristics with the usage of domain knowledge, while in [91] a machine learning based approach has been shown capable to scale across different domains with a reduced training effort.

**Synonym Extraction.**

The goal of this task is to identify when different terms have the same meaning. In order to achieve such goal, the well-known distributional hypothesis, claiming that terms sharing syntactic contexts are semantically similar, is largely exploited. Examples are works in [18, 61, 74]. In particular, in [74], a rich similarity metric is defined, taking into account both the syntactic and the semantic context, together with statistical co-occurrences of terms. Other Text Mining clustering techniques are used in the approach presented in [26], while external resources such as WordNet\(^2\) are used in [66].

**Concept Learning.**

At this layer, concepts must be induced in an intensional way. Clustering algorithms and Latent Semantic Indexing (LSI) are used in [26] in order to suggest concepts to the Ontology engineer, while in [74] sets of terms considered synonyms are casted into new concepts named after the more occurring terms. WordNet Domains and other resources are used in [83] to associate the newly extracted keyphrases with existing concepts. Context similarity is used in [18, 61] to induces concept definitions over sets of terms.

**Concept Hierarchy.**

The goal of this task is to learn taxonomic relations over extracted concepts. Recall that a taxonomic relation between two concepts is a relation in

which we state that one concept is a sub-concept of the other. Well-known lexico-syntactic patterns from [40] are exploited in [97] and, together with WordNet, in [18] to solve this problem. The approach presented in [76] and further developed in [77] uses syntactic patterns as features in order to train a binary classifier able to predict if two nouns are in a taxonomic relation. Instead, in [74], a taxonomy is induced using a hierarchical clustering process based on Kohonen Self Organizing Maps with a distance metric based on term co-occurrence and a variation of TF-IDF. In [61] a complex context similarity measure is used to insert new concepts in the right position of an existing hierarchy by asserting their equivalence or sub-concept relation with an existing concept. Starting from a graph in which nodes represent terms and edges represent relations between concepts, some other recent works use a task-specific algorithm (graph pruning, edge weighting, and so on) in order to turn this graph into a taxonomic tree. In [48, 84] a graph with concepts as vertices is built using pattern-driven web search operations, while in [27] a similarity metric over a vector space model is used to evaluate term relatedness, and organize concepts in a graph structure, according to their mutual similarity.

**Relation Learning.**

The output of this layer is the set of all relations among concepts and individuals, eventually organized in a hierarchical order. The work in [97] uses the same approach followed for terms extraction, namely using linguistic patterns and metrics from Graph Theory, to perform relation extraction. Linguistic patterns are used also in [42] in order to detect verb-based relations between pairs of noun phrases. In [61], context similarity is taken as an evidence of a generic conceptual relation: if a certain degree of context similarity between two concepts is detected but does not originate an equivalence or a subclass relation, a generic `related_to` relation is assumed to hold. Statistical signif-
2.1. AN OVERVIEW ON ONTOLOGIES

The significance of co-occurrence is used in [73] in order to predict a relation between two terms, while, in [20], patterns expressed in an *ad hoc* formalism are used to detect instances of known relations. Syntactic features are used in [18] in order to detect general relations, while a combination of a set of patterns together with WordNet is used for mereological relations. Works presented in [13, 1, 39, 96] use iterative approaches. First, a set of lexico-syntactic patterns is hand-crafted. Then, these patterns are used to match textual surface realizations of such relations. The matched text fragments are finally used to extract new patterns to extend the initial catalogue. Parse tree feature spaces are used in [14, 19, 95] in order to train tree kernel based This task classifiers which can predict occurrences of predefined relations. Hand-crafted rules are used in [8] in order to label a set of examples which will be used to train a classifier. This semi-supervised approach has been extended in [24] with Part-of-Speech tag patterns acting as syntactic constrains and a large dictionary of relations acting as lexical constraints, in order to improve the quality of the extraction. A pairwise vector space clustering is used in [73] in order to detect recurring patterns and identify relation instances. A clustering technique is used also in [56] in order to detect generic relations for given type signatures. Named entities are clustered according to their similarity in [38] in order to detect relation occurrences. Finally, several ontological resources like Freebase[^3], YAGO[^4] and Wikipedia[^5] infoboxes are exploited in [55, 92, 58, 57] as sources of evidence in order to match textual relation instances from which different types of features are extracted to train classifiers or cluster instances.

[^3]: The Freebase project is currently dismissed. The data dumps are available at [https://developers.google.com/freebase/](https://developers.google.com/freebase/) (last accessed on January 31, 2018).
Axiom Learning.

The last layer addresses the problem of axioms learning. The goal of this task is to extract terminological axioms that go beyond plain taxonomical relations. Several approaches have been proposed along the years, ranging from the extraction of axioms in $\mathcal{EL}^{++}$ to axioms in $\mathcal{SHOIN}^D$, which is the Description Logic language implemented by the full OWL-DL language.

From a methodological point of view, we observe that many of these approaches strongly rely on hand-crafted rules, manually edited grammars, or heuristics. Examples are [87, 88, 67, 52, 70, 68, 50]. In these works, text is processed with pre-trained Natural Language Processing standard toolkits in order to extract symbolic features. This new text representation is subsequently manipulated through a set of rules in order to obtain a corresponding formal representation. In these approaches, the World Wide Web is largely exploited as a source of evidence to validate if some axiom actually holds. Recently, the authors of [64] proposed a fully Machine Learning-based approach. They exploited a minimal set of linguistic and semantic feature to train several classifiers to extract concept formalizations for SNOMED CT from documents about the biomedical domain.

Note that many of these approaches implicitly collapse different layers of the layer cake, accepting raw text as inputs and performing a linguistic and, possibly, semantic analysis instead of a knowledge extraction strictly coherent with the Layer Cake. During our work, we also developed an approach that turns raw text into a Description Logic formula through syntactic transformation, as stated in [87]. The problem, together with the approach we followed to tackle it, will be described in detail in Chapter 3. From a technical standpoint, our approach is fully Machine Learning-based and exploits deep Neural Network architectures, for which some background concepts will be given in the next Section.
2.2 An overview on Neural Networks

Machine Learning is the field of Computer Science that collects those methods aiming to give a software agent the ability to solve a task without explicitly programming it. Instead, the agent is trained over a set of examples to mimic the desired behaviour. Such training is performed through the application of some induction algorithm over a set of examples, which are expected to represent a meaningful sample of the expected behaviour of the agent. As an example, consider an agent that, given a picture representing a cat or a dog, can tell which one is actually represented. Such agent will accept as input some numerical representation of the picture and will produce as an output one label, CAT or DOG. Referring to this simple example, we can outline all the major challenges that one has to face trying to exploit a Machine Learning approach. This particular task is called a classification task, since we want our model to predict which class an input image belongs to. Other tasks that are commonly solved with Machine Learning approaches are also regression, namely to predict the value of a continuous variable, and clustering, namely grouping together inputs that can be considered similar. To the extent of this thesis, we have been dealing with classification problems.

First, we must find a meaningful and effective numerical representation of the input. This representation is typically a vector. Each component of such vector is called a feature of the input. In the scenario of our example, a suitable input representation can be found constraining the dimension of the input image to say $128 \times 128$ pixels. Then we can represent each pixel with three numbers between 0 and 255, each of which represents one of the RGB component for the given pixel. In this way, we will have a vector of $3 \times 128 = 49152$ features to represent the input image.

Then, we have to collect a meaningful sample of images of cats and dogs. Such examples must be manually annotated, so that for each image we know
which is the correct label that the agent has to choose. Saying meaningful, we mean that it must be large enough to cover different types of cats and dogs, of different colors, pictured from different angles, of different sizes, with different backgrounds, and so on. As a counter-example, think of a collection of such examples where we have only black cats and not even a single black dog. In this way, the induction algorithm will produce an agent that will be biased, in the sense that it will more likely say CAT whatever black pet is actually in the input image. The set of examples that we use to train the agent is called training set. Note that in practice, this term is used in a slightly different way. Indeed, not all the training examples are used for actually training the agents, as some of them are used to test the performance during and after the training. Such example are used to verify that the network is actually learning the underlying phenomenon and not just memorizing the training examples. This memorization behavior is known as overfitting.

Once we have a proper training set and have settled a proper input representation, we have to choose a proper model to train. A Machine Learning model is typically a class of functions, namely a function with several parameters with values to be defined—as an example, the formula $f(x) = mx$ represents the class of all the lines crossing the origin of the cartesian plane. The type of function and the number of parameters have to be decided at design time. During the training phase, pairs made of inputs and corresponding expected outputs from the training set are submitted to the agent. It computes its own predictions of the output labels, and adjusts the value of its parameters so that the prediction error is reduced. This training procedure is called supervised learning, meaning that the model is provided with some information about the expected output for the training examples. When no labelled data are available during the training, we are in an unsupervised learning scenario, and the model is trained to find some structure in its input. To the extent of this thesis, we have been dealing with supervised
2.2. AN OVERVIEW ON NEURAL NETWORKS  CHAPTER 2. BACKGROUND

learning problems.

Training data for a supervised learning scenario can be collected in different ways. The most straightforward way, is to collect input data and manually label them with the corresponding answer expected by the agent. In our running example, we can collect several pictures of dogs and cats and ask some human annotators to tell us if a given pictures depicts a DOG or a CAT. This way is typically the most accurate but expensive, since it involves human annotators that, for more complex tasks, can require an ad hoc training. To reduce the cost of the training data collection process, one can think of synthetically generate such data together with their annotation. In our case, one could use a 3D render of some dogs and cats to generate pictures of them from different angles, with different colors, and so on. The resulting artificial data set will be an approximation of the real world phenomenon under observation. Such approach is used when a new model is proposed for some new task for which no training set is available (see [33, 93, 90, 89, 63, 11, 32, 65] just to name a few). In this way, indeed, before embarking into the process of producing a manually annotated dataset, one can test if some model is actually suitable for the task, even if approximately. In between of these two approaches, we can follow a distant supervision approach, where input data are aligned to the proper label through some hand-crafted rules, some heuristics or other Machine Learning systems. Back to our example, suppose that we have another system that computes some degree of similarity between images. In this way, we could manually annotate few images and then consider as cats and dogs other images that are considered similar to the annotated ones from the system above, increasing the size of the training set. As a trade off, the distant supervision can introduce some noise among the training example, since the other system that we use to label the new examples can be inaccurate and it is likely to model some phenomena which is different from the one under observation.
CHAPTER 2. BACKGROUND  2.2. AN OVERVIEW ON NEURAL NETWORKS

The simple example presented so far helped us to focus on some prominent issues that everyone has to face when dealing with a Machine Learning-based approach to solve any task: the collection of the training data as a meaningful sample of the problem space, the proper input representation, the training process. In particular, the example described a supervised learning problem for a classification task, which is also the scenario in which the present work falls into. Moreover, among the different available Machine Learning models, we used Neural Networks, a particular class of Machine Learning methods in which the model is built through the connection of different units, each of which is intended to resemble a biological neuron. In this way, we say that Neural Networks are inspired by neuronal biological systems. In this section, after having settled some basic notation (2.2.1), we will ground the basic knowlege on Neural Networks (2.2.2), focusing in particular on the family of Recurrent Neural Networks which have been extensively used in the field of Natural Language Processing and are particularly suitable to model human language (2.2.4). Finally, we will try to recap and motivate the new wave of Neural Networks, which is often referred to as Deep Learning (2.2.5).

2.2.1 Nomenclature and Basic Notation

We indicate vectors with bold lowercase letters, e.g. \( \mathbf{x} \). Writing explicitly a vector in its components, we use square brackets, like in \( \mathbf{x} = [x_1, ..., x_n] \). To denote the \( k \)-th element of the vector \( \mathbf{x} \) we use the index within square brackets, like in \( \mathbf{x}[k] \). The element-wise dot product between two vectors is indicated using the \( \odot \) symbol. To indicate the concatenation between two vectors, we will use the \( \oplus \) operator, so that given \( \mathbf{a} = [a_1, ..., a_n] \) and \( \mathbf{b} = [b_1, ..., b_m] \), we write: \( \mathbf{a} \oplus \mathbf{b} = [a_1, ..., a_n, b_1, ..., b_m] \). We use bold uppercase letters, e.g. \( \mathbf{W} \) to indicate matrices. The transposition operation for vectors and matrices is indicated with the \( .^T \) superscript, as in \( \mathbf{x}^T \). Uppercase letters are used to represents sets other than the set of real numbers, denoted with
2.2. AN OVERVIEW ON NEURAL NETWORKS  

Given a finite set $A$, we indicate the number of its element with $|A|$. We use the greek letter $\theta$ to indicate a set of trainable parameters. We indicate sequences of objects—to distinguish them from vectors defined on some vector space—with comma separated items, like in $s = s_i, \ldots, s_n$ or $H = h_1, \ldots, h_n$. For the concatenation of two sequences we use the $\oplus$ symbol as well. The position of a symbol within a sequence is called timestep and is indicated with a subscript, so that $x_i$ represents the vector $f$ at the $i$-th timestep. Typically we will use the letter $i$ to indicate the generic timestep of an input sequence, while $j$ will be use for the generic timestep of the output sequence.

2.2.2 Neural Networks: Basic Concepts

Neural Networks are particular computing systems inspired by the biological nervous system of animals. The nervous system is a globally complex system, made of several locally simple interconnected cells, the neurons, with electric signals flowing through this network. Each neuron accepts different electric signals coming from other neurons. The signal produced by a neuron propagates through a nerve fiber called axon, with several terminals. A neural cell has several extensions capable to accept the signal carried by an axon, called dendrites. The connection between a terminal of an axon and a dendrite is realized by a specialized structure called synapse. Through such connection system, a single neuron can receive many signals coming from other neurons. All such signals reach the nucleus of the neuron and, if their combined effect reaches some critical level, the neuron fires a new electric signal that propagates to other neurons in the network. A neuron with its connections is depicted in Figure 2.3.

Inspired from the biological neuron, the authors of [53] proposed their model for an artificial neuron (just neuron henceforth), a mathematical function intended to model the biological neuron. For a more detailed de-
Figure 2.3: Biological neuron and its connections. The picture is taken from Wikipedia.

scription of the biological neuron and its relation with the artificial one, see Chapter 1 of [25]. Further on, we will use the word “neuron” to denote an artificial neuron. Concretely, a neuron is a function accepting a vector of real numbers as input, each of which represents a signal accepted by a dendrite, and producing a single real number as output, representing the signal propagated by the axon. First, the input vector $\mathbf{x}$ is dot-multiplied with another vector $\mathbf{w}$, representing the weights of the input signals, and the summed with a constant value $b$, called the bias signal. The result of this first step, $z$, is typically called potential, and is described by

$$z = \mathbf{w}^T \mathbf{x} + b,$$

where, said in the number of inputs of the neuron, both $\mathbf{x}$ and $\mathbf{w}$ are vectors defined in $\mathbb{R}^m$. Note that the bias signal can be seen as the weight of an input signal which is constantly set to 1.

The final output of the neuron, $y$, also called activation, is produced processing the potential with another function, $f(\cdot)$, called the activation
function. We can write the neuron formula as

\[ y = f(z) = f(w^T x + b). \]  

(2.29)

The process from the inputs to the activation of a neuron is depicted in Figure 2.4. Note that the calculation of the potential and the application of the activation function are represented in separate stages, for the sake of clarity. Further on, they will always be collapsed together, as in the dashed ellipse in Figure 2.4.

![Figure 2.4: An artificial neuron with three input signals (plus the bias signal).](image)

The first model of the artificial neuron used just a threshold as its activation function: like for biological neurons, such units can be on or off depending on their potential. Lately, many other activation functions have been proposed and used. Examples of common activation functions are the sigmoid function and the hyperbolic tangent function. They have been exploited also in this thesis and are graphically represented in Figure 2.5.

The sigmoid first one is defined as:

\[ \sigma(x) = \frac{1 + e^x}{e^x}. \]  

(2.30)

It ensures that the output falls in the [0, 1] range, so that it can be considered as the probability of the neuron parametrized with \( w \) and \( b \) to be on.
This probabilistic interpretation is lost with the hyperbolic tangent function, defined as follows:

\[
tanh(x) = \frac{1 + e^{-2x}}{1 - e^{-2x}}.
\]

(2.31)

This function constrains its output within the \([-1, 1]\) range. Nonetheless, such function is widely used because of better empirical performances, possibly due to having both positive and negative values for the activation.

A Neural Network is a network resulting from the composition of several neurons, according to some topology defined at design time. The whole Neural Network can be seen as a class of functions accepting a vector of real numbers as input and producing another vector of real numbers as output, similarly to the single neuron, but on a wider scale. Those neurons accepting as input the actual input of the network, are called input neurons, while those producing the final output of the network are called output neurons. Each neuron accepts, as its input, the output signals of another set of neurons. Exceptions are the input neurons, which accept the actual input of the network. Following these connections, we can represent a Neural Network as a possibly cyclical graph structure. When neurons can be grouped together with respect to their depth in the graph, they form groups called layers.
It is straightforward to extend the equation of a neuron presented in (2.29) to a whole layer. Said in the number of inputs of the layer, namely the number of neurons of the previous layer or the direct inputs of the network, and out the number of outputs of the layer, namely the number of its neurons, we can write:

\[ y = f(W^T x + b) \]  

(2.32)

where \( x \) is a vector in \( \mathbb{R}^{in} \) representing the input of the layer, \( W \) is a matrix in \( \mathbb{R}^{in \times out} \) in which the \( i \)-th row is the weight vector of the \( i \)-th neuron in the layer, and \( b \) a vector in \( \mathbb{R}^{out} \) where the \( i \)-th component is the bias signal of the \( i \)-th neuron in the layer. Note that the activation function \( f(\cdot) \) is intended to be applied element-wise to the activation value which is, in this case, a vector in \( \mathbb{R}^{out} \).

![Figure 2.6: A layered architecture with 3 inputs, 4 hidden neurons (in red) and 2 output neurons (in blue).](image)

Typically, network architectures are described in terms of their layers,
which are stacked one on top of the other. An example of a layered architecture is depicted in Figure 2.6. The network topology is determined at design time. The number of layers, the dimensions of each layer, the type of activation functions used for each layer (or even for each neuron), and all the other settings that define such topology fall, together with other values that will be introduced later, in the set of the so-called hyper-parameters of the network. Following the network topology, we can chain the equations of each layer, obtaining a single one. The result of this chaining operation is a function depending on a set of parameters, \( \theta \), comprising all the weights and bias signals from all the layers in the network—plus, possibly other parameters. We can write:

\[
y = f(x; \theta),
\]

(2.33)

where \( x \) is the network input and \( y \) the network output. More precisely, we obtain a class of functions, one for each possible combination of values of the parameters. So, while the class of function is defined by the hyper-parameters, the actual function is defined by the particular set of values that is chosen for the parameters. Such values are determined at training time, namely, during the learning process. Note that function (2.33) computes the final output of the network starting from the direct input. This computation is called the **feedforward** process of the network.

### 2.2.3 The Learning Process of Neural Networks

In a supervised learning scenario, like the one that we have been dealing with in our work, a Neural Networks is trained trying to minimize the value of some **loss function**. This function is intended to measure the difference between the actual behavior of the network and the expected one. The choice if the proper loss function is a crucial one when designing a neural network training process. Training examples are submitted to the network and the
value of the weights are adjusted so that the loss is minimized. An extremely popular training procedure is the so called error backpropagation algorithm (see [69]). In the error backpropagation, the inverse direction of the feedforward process is followed in order to compute the gradients of the loss function with respect to each parameter. Such gradients are then used by some optimization method, chosen at design time, in order to adjust the value of each parameter so that its contribution to the global error decreases. We will describe in detail the algorithm starting from a single neuron, like the situation depicted in Figure 2.7, with a number $J$ of incoming input signals, generally indicated with $x_j$. In our first scenario, a single training example is submitted to the network.

Let us assume that an input vector $\mathbf{x} = [x_1, ..., x_J]$ is submitted to the neuron, which generates the output value $y$. Said $\tilde{y}$ the expected value, the loss function will depend on the actual output, the expected one, and the current value of the parameter set, so that we can write:

$$L \equiv L_\theta(\tilde{y}, y), \quad (2.34)$$
where $\theta$ is the set of all the parameters of the network to be adjusted during the training phase. In the current example, we have $\theta = w_1, \ldots, w_J, b$. The main idea is that a fraction of the gradient of the loss with respect to the weight is used to update the weight itself, according to

$$w_j = w_j - \eta \frac{\partial L}{\partial w_j}$$  \hspace{1cm} (2.35)

where the quantity $\eta$ is the so-called learning rate, which determines how much of the gradient of the error is actually used to update the value of the weight. Note that the update value is signed with minus since we aim to minimize the loss.

To compute the updated value of the weight $w_j$, we first compute the derivative of the error with respect to such weight by applying the chain rule for derivatives, ending up in

$$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_j}.$$  \hspace{1cm} (2.36)

We define the error signal, $\delta$, as the amount:

$$\delta = \frac{\partial L}{\partial y} \frac{\partial y}{\partial z}$$  \hspace{1cm} (2.37)

which can be seen as the signal flowing backward through the network and carrying a measure of difference between the actual output and the expected one. Plugging equation (2.37) into equation (2.36) and explicitly writing the dot product in the computation of (2.28), we can rewrite (2.36) as:

$$\frac{\partial L}{\partial w_j} = \delta \frac{\partial z}{\partial w_j} = \delta \frac{\partial}{\partial w_j} \left( b + \sum_j w_j x_j \right) = \delta x_j$$  \hspace{1cm} (2.38)

and the new value for the weight $w_j$ will be given by:

$$w_j = w_j - \eta \delta x_j.$$  \hspace{1cm} (2.39)
Since we can think at the bias signal as the weight of for a further input signal with a constant value of 1, plugging this value into the $x_j$ in equation (2.39), we obtain the update formula for the bias signal:

$$b = b - \eta \delta.$$  

(2.40)

We can extend our model to a generic layered architecture and study in detail how error backpropagation works for each neuron. To easily describe this scenario, which is syntetically depicted in Figure 2.8, we have to explicitly define the notation. For each weight, input, activation, potential or error signal, we will use a superscript to denote the layer. Input, output, or error signals, as well as potentials, will be denoted with a subscript denoting the neuron within the layer they refer to, so $z^l_j$ will denote the potential of the $j$-th neuron within the $l$-th layer. Weights are denoted with a double subscript,
where the first letter denotes the position of the neuron within the layer and
the second denotes the position of the neuron in the previous layer whose
activation is processed by the weight itself. If the current layer is the input
layer, the second letter in the subscript of the weight represents the input
signal. So, \( w_{ji} \) represents the weight that modulates the output of the \( i \)-th
neuron of the \( (l - 1) \)-th layer into the \( j \)-th neuron of the \( l \)-th layer. We will
show in detail how backpropagation works for the \( j \)-th input weight of the
\( i \)-th neuron of the \( l \)-th layer, namely \( w_{ji} \). We will use the letter \( j \) as the
index for the \( l \)-th layer, the letter \( i \) as the index of the \( (l - 1) \)-th layer, and
the letter \( k \) as the index for the \( (l + 1) \)-th layer.

As for the case of the single neuron, we will start computing the derivative
of the loss with respect to \( w_{ji} \) and applying the chain rule, obtaining:

\[
\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ji}},
\]

and writing explicitly the last term and differentiating with respect to \( w_{ji} \),
we obtain:

\[
\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ji}} (b + \sum_i w_{ji} y_{i-1}) = \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial z_j} y_{i-1} = \delta_j y_{i-1},
\]

where we have defined the error signal \( \delta_j \) as:

\[
\delta_j = \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial z_j}.
\]

Note that, to compute the first term of the right-hand side, namely the
gradient of the loss with respect to the output of the current neuron, we
have to consider all the backward contributions from the neurons of the
following layer. This means that, since \( y_j \) contributes the potential of each
neuron in the \( (l + 1) \)-th layer, this backward contribution will be given by
the summation:

\[
\frac{\partial L}{\partial y^l_j} = \sum_k \frac{\partial L}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial y_j^l},
\]

and applying the chain rule to the first term in the summation argument, we can expand to:

\[
\frac{\partial L}{\partial y_j^l} = \sum_k \frac{\partial L}{\partial y_k^{l+1}} \frac{\partial y_k^{l+1}}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial y_j^l}.
\]

The third term in the summation argument evaluates to \(w_{k_j}^{l+1}\), since, we have that:

\[
\frac{\partial z_k^{l+1}}{\partial y_j^l} = \frac{\partial}{\partial y_j^l} \left( b_j^l + \sum_{t=1}^J w_{kt}^l y_t^l \right) = w_{k_j}^{l+1}.
\]

Plugging it into (2.45) gives us the following:

\[
\frac{\partial L}{\partial y_j^l} = \sum_k \frac{\partial L}{\partial y_k^{l+1}} \frac{\partial y_k^{l+1}}{\partial z_k^{l+1}} w_{k_j}^{l+1}.
\]

Note that the first two terms of the summation argument are the error signal flowing back from the \(k\)-th neuron of the \((l + 1)\)-th layers, so that we can write:

\[
\frac{\partial L}{\partial y_j^l} = \sum_k \delta_k^{l+1} w_{k_j}^{l+1}.
\]

where the error signal flowing back from the \(k\)-th neuron of the following layer is weighted through the same value that, in the forward pass, weights the output of the neuron. Plugging this equality into (2.43), we have that:

\[
\delta_j^l = \left( \sum_k \delta_k^{l+1} w_{k_j}^{l+1} \right) \frac{\partial y_j^l}{\partial z_j^l}.
\]
of the weight $w_{ji}^l$ can be written as:

$$w_{ji}^l = w_{ji}^l - \eta \left( \sum_k \delta_{k}^{l+1} w_{kj}^{l+1} \right) \frac{\partial y_{j}^l}{\partial z_{j}^l} y_{i}^{l-1},$$

(2.50)

while for the bias signal of the $j$-th neuron in the same layer, this update formula becomes:

$$b_{j}^l = b_{j}^l - \eta \left( \sum_k \delta_{k}^{l+1} w_{kj}^{l+1} \right) \frac{\partial y_{j}^l}{\partial z_{j}^l}.$$

(2.51)

Summing up, the training phase of a neural network is a process during which the value of the parameters of the network are updated to minimize the loss function with respect to the training examples. The process of computing the loss and updating the parameters is performed iteratively many times. Ideally, the loss should be computed averaging the value of the loss across the whole training set. For large training sets, this would be highly impractical, so, typically the training set is split into batches of examples that are used to compute the loss and update the weights. This procedure falls into the **Stochastic Gradient Descent**, where the gradient of the loss function is approximated by the gradient of the same function computed on a batch of examples. The batch size is a hyper-parameter to be set at design time. The number of iterations must be set at design time as well: it can be fixed or we can decide to stop the training when the network reaches some expected performance level measured on a set of examples, typically called validation set. The latter approach is called **early stopping**.

### 2.2.4 Neural Networks and Natural Language Processing

Recently, Neural Networks have been successfully exploited in several Natural Language Processing tasks, such as language modeling ([9]), dependency parsing ([15]), machine translation ([7]) and sentiment analysis ([22]).
ing with natural language, there are two main problem that must be tackled. The first one is how to represent natural language in a numerical way, so that it can be fed into a Neural Network. The second one is how to deal with the intrinsically recurrent nature of natural language, where the meaning and the function of each word depends on the context of the whole sentence and on its temporal evolution. Low dimensional word embeddings, or just word embeddings, or word vectors, have been proposed as an effective solution for the first problem, while Recurrent Neural Networks have been vastly used in order to solve the second. Below we will describe such concepts in detail.

**Word Embeddings.** In order to be fed into a Neural Network, text must be numerically represented in a proper way. Said $V$ the vocabulary of the language we are dealing with, we can sort all the possible words and map each word in a sentence to a natural number representing its position within $V$. Given a word $w$, we will denote its index within $V$ as $V(w)$. The most straightforward way to convert a word from a vocabulary into a vector is to follow the so-called **one-hot** encoding. We can represent the word $w$ as a vector in $\{0, 1\}^{|V|}$ with all the components set to 0 and one corresponding to the word index, namely the $V(w)$-th, which is set to 1.

One-hot vectors are extremely simple but, at the same time, they can be highly impractical for two reasons. The first one is the so-called **curse of dimensionality**: they have the same size of the vocabulary $V$, so it can grow up to large values. The second one is that, composing two one-hot vectors with standard linear algebra operators—vector sum, dot products, and so on—we could obtain something that is **not** a one-hot vector anymore. In order to overcome these problems, words can be represented in a continuous vector space on $\mathbb{R}^d$, where $d$ is the dimension of the vector space. The value of $d$, an hyper-parameter to be set at design time, is typically way smaller.

---

6 As an instance, consider that the whole Wikipedia dump from April, 30, 2017, after replacing all the digits with a conventional symbol and ignoring casing, has more than 16 million words.
than \(|V|\), allowing us to avoid the first problem described above. Regarding the second, being such vectors defined in a continuous vector space on \(\mathbb{R}\), they can be manipulated with standard linear algebra operations, ending up always in vectors from the same space. Such vectors are called word vectors or **word embeddings**, since they come from the operation of embedding the word in a low-dimensional vector space. The hyper-parameter \(d\) is called **embedding size**.

Word embeddings can be learnt during the training of the network in a conceptually easy way. Let us assume that we fixed an embedding size of \(d\) for a network accepting words coming from a vocabulary \(V\) as its inputs. We can map discrete indexes of input words into continuous vectors in \(\mathbb{R}^d\) using a so-called **projection layer** as the first layer of the network. The parameters of this layer are collected in an **embedding matrix**, say \(E\), defined in \(\mathbb{R}^{|V| \times d}\). In this way, the \(i\)-th row of the matrix can be seen as the word vector for the \(i\)-th word in \(V\). Moreover, the \(d\) components of each word vector can be seen as the features defining the semantics of the corresponding word in a latent feature space. The projection layer accepts the index of a word, say \(V(x)\), as its input and return the corresponding word vector as output, namely the \(V(x)\)-th row of \(E\). From the linear algebra point of view, it is easy to see this row lookup operation as a matrix multiplication between the transposed embedding matrix, \(E^T\) and the one-hot vector \(e_i\) of the \(i\)-th word in \(V\). Note that such lookup operation can be seen as a special case of (2.32) where the bias vector is set to all zeros and the activation function is just an identity function. During the backpropagation phase, the values of the embedding matrix, which are the features for each word in the latent feature space of dimension \(d\), are adjusted as regular weights, so the network **learns** the most suitable representation for each word.

Word embeddings can be learnt separately before training the whole network. Such pre-trained word embeddings can be used as the initial values
for the embedding matrix and then fine-tuned during the training. Such pre-
training can leverage large amount of unlabeled data, as shown in [9, 54], so
that no human annotation is needed to build a meaningful representation of
the language in a continuous vector space model.

**Recurrent Neural Networks**  Natural language is intrinsically recurrent, in
the sense that the meaning and the function of each word depends on the
other words through syntactic and semantic dependencies. To capture and
manage such dependencies, a neural network must be provided with some
capabilities of tracking the evolution of the sentence through time, keeping
memory of what have been seen up to every timestep. Recurrent Neural
Networks (see Chapter 10 from [30]) have been proven extremely efficient
in dealing with sequential phenomena, and among them, natural language.
A Neural Network shows a recurrent behavior when the output of some of
its layer depends not only from the current state but also from some state
that happened previously in time. This behavior can be achieved in different
ways.

As a reference example, let us consider a neural network that, given a
word, labels it with the proper Part-of-Speech—such task is called Part-
of-Speech tagging and is a well known task in the field of Natural Language
Processing. A simple conceptual model for such task will be a possibly multi-
layered network accepting as an input a word vector representing the current
word in the sentence and producing as its output a vector of the same size
of the catalogue of all the possible Part-of-Speech tags: such vector will be
an estimation of the probability of each possible label. We could improve
such architecture adding some context in the input of the network just con-
catenating the word vectors of the surrounding words together with the one
of the word for which we are running the prediction of the Part-of-Speech
tag. This technique is called **windowing**, since we have a window of words
instead of just a single word as input, at each timestep. Moreover, we could extend the resulting input vector concatenating furtherly the previous output vector—in order to stress, for instance, that is unlikely that a determiner follows another determiner.

A more complex and effective way to model the temporal evolution of a sequence of symbols is to exploit recurrent activation functions within the layer of a Neural Network. A generic recurrent function \( g(\cdot, \cdot, ...) \), accepts as its input at timestep \( t \) some representation of the current timestep, say \( x_t \), and its own output from the previous one, say \( y_{t-1} \) (and possibly some other arguments), as in the following:

\[
y_t = g(x_t, y_{t-1}, ...).
\] (2.52)

Such recurrent functions are typically called cell functions and many different ones have been proposed along the years. The main trade-off when choosing the proper cell function dwells in the fact that a more complex cell function can handle longer term dependencies at the price of a larger amount of parameters and, as a consequence, a larger memory footprint and longer training time for the model.

To the extent of this work, we mostly used the so called Gated Recursive Units (see [16]), which have been proven capable to endow Neural Networks with effective short term memory. Moreover, compared to other widely used cell models like Long Short-Term Memory (see [41]) cells, Gated Recursive Units have a significantly smaller number of parameters effect using a number of parameters which is significantly smaller. This make them easier and cheaper to train. The whole cell model is synthetically depicted in Fig. 2.9.

The cell behavior is driven by two gate functions, whose value ranges in the \([0, 1]\) interval. The reset gate, denoted with \( r \), and the update gate,
denoted with \( z \) which, at the \( i \)-th timestep, are defined as follows:

\[
\begin{align*}
    r_i &= \sigma(W_r x_i + U_r y_{i-1}), \\
    z_i &= \sigma(W_z x_i + U_z y_{i-1}),
\end{align*}
\]

(2.53)  

(2.54)

where \( W_r, U_r, W_z, U_z \) are weight matrices that are learnt during the training, \( x_i \) is the current input of the cell, \( y_{i-1} \) is the previous cell activation, and \( \sigma(\cdot) \) indicates the element-wise logistic sigmoid function, that squeezes the values of each component of the vector between 0 and 1. The inner state of the cell is represented by a vector \( \tilde{y}_i \) defined:

\[
\tilde{y}_i = \tanh(W x_i + r \odot U y_{i-1}),
\]

(2.55)

where \( W \) and \( U \) are learnt weight matrices and \( y_{i-1} \) is the activation of the cell at the previous timestep. Intuitively, when the reset gate \( r \) gets close to 0, the piece of information carry by the feedback from the previous activation, namely \( U y_{i-1} \), tends to be ignored. The cell activation is given by the function \( g(\cdot, \cdot) \) defined as follows:

\[
y_i = g(x_i, y_{i-1}) = z \odot y_{i-1} + (1 - z) \odot \tilde{y}_i
\]

(2.56)

where \( 1 \) is a vector of the same size of \( z \) with all the components set to 1.
CHAPTER 2. BACKGROUND 2.2. AN OVERVIEW ON NEURAL NETWORKS

Intuitively, the update gate $z_j$ balances the amount of information to be kept from the previous activation and from the current state.

The set of the parameters to be learnt in the training phase can be summed up as $\theta_{gru} = W_r, U_r, W_z, U_z, W, U$.

2.2.5 The New Wave of Neural Networks

The first model of artificial neuron was proposed in [53] in 1943. A new paradigm for Artificial Intelligence started, called connectionism. The main idea of such paradigm was that human could build intelligent machines replicating the structure of the brain more than symbolic reasoning. During its history, Artificial Intelligence fluctuated forth and back between seasons of great enthusiasm and so called winters, characterized by a reduced interest in the field and reduced research fundings (see [60]). The connections paradigm, in particular, experienced its first winter around the 1980s, living some sort of partial re-birth by the end of the 1990s. But they have never been a mainstream techniques if compared to other extremely popular Machine Learning techniques.

In recent years, however, things changed dramatically and Neural Networks are living a sort of new wave. The wide spreading of GPU cards allowed engineers to radically speed up the training phase.\footnote{As a reference, considering that to the experiments carried out to the extent of the present work, each hour of training on a Tesla K80 GPU Card is equivalent to roughly 30 hours of training on a machine endowed with an Intel i7 vPro CPU card.} This innovation started a sort of chain reaction that produced two main outcomes. The first one is a significant leap ahead in the technological aspects with new frameworks and libraries, new paradigms and new approaches for distributed, faster, and large scale training processes. The second one is a renewed the scientific interest in the field, fostering new research efforts. In particular, the training of deep architectures, were several layers are stacked on top of the other, has become a common practice, leading to wide spreading expressions such
as “deep networks” and “Deep Learning”.

The main advantage of such deep architectures is that they can effectively learn expressive feature representation, as nicely shown in [49]. This characteristic has been exploited in many fields following the common rationale of reducing the feature engineering cost. In this way, many problems that were once tackled in a step-by-step manner, namely extracting subsequent representations of the data, are now faced in an end-to-end way, starting from the initial one and ending directly into the final representation of the problem.

The contribution presented in this thesis fits this description. A previous way of translating language into logic would have exploited subsequent feature extraction phases: Part-of-Speech tags, then constituents, then dependency structures, and so on. On top of everything, a set of hand-crafted rules (as in [87]) or some other statistical learning based system (as in [50]) would have performed the final transformation. What we tried to achieve, in our work, is to translate directly text into logical forms, without any intermediate step. These aspects will be discussed further in Chapter 7.
Chapter 3

From Definitions to Formulæ

As summarized in 2.1.3, many different approaches for Ontology Learning from text have been proposed along the years. Still, the vast majority of them is focused on the extraction of knowledge that is limited in its expressiveness. As a consequence, encoding expressive Ontologies is still a largely manual, time consuming, and error prone process.

In Section 3.1, we describe the problem of turning a natural language sentence into a corresponding logical form as we tackled in this thesis. We present the main challenges that we faced while defining this problem in detail and how we addressed them. In Section 3.2, we present the approach we followed to solve the problem and the challenges that we had to face realising the solution. The way we addressed such challenges is presented in detail in Chapter 4, Chapter 5 and Chapter 6.

3.1 The Problem

As stated in 88, the problem we are facing is to translate the definition of a concept expressed in natural language into a corresponding Description Logic formula through a syntactic transformation. This mean that, ideally, the
following sentence:

\[ A \text{ bee is an insect that produces honey,} \quad (3.1) \]

will be translated into the formula:

\[ \text{bee } \sqsubseteq \text{insect } \sqcap \exists \text{produces.honey.} \quad (3.2) \]

Note that all the extralogical symbols in the formula in (3.2) are equal to some words from the sentence in (3.1). To the extent of this work, this is what we mean when we say that the Description Logic formula is obtained from the input sentence through syntactic transformation.

While the simple example presented by means of sentence (3.1) and formula (3.2) is evocative enough to describe the general problem to a human reader, it is definitively underspecified to plan for a solution. Instead, just to make a simple example, not all the natural language sentences have a corresponding Description Logics formula and. Moreover, Description Logic languages are not a single formalism but a family of logics. Thus, once we stated the abstract problem as we did, we had to face two main challenges. The first one, was to precisely identify the source language. This means to define the particular subset of natural language that is used by human agents to express definitions of concepts that are meaningful from an ontological standpoint. The second one was to identify the target language, namely the particular Description Logics language we want to use to formally encode the knowledge expressed in the source language. The way we addressed both these challenges is presented below.

### 3.1.1 First Challenge: Choosing the Source Language

From an intuitive point of view, we are interested in sentences in plain English that can be used to express the salient characteristics of a species, giving an
intensional characterization of a set of entities. Investigating the literature for some theory of definition (see [59] and cited works), we found that many shared structural features of a definition can track back to the “Organon” by Aristotle, and especially to the sixth book of the “Topics” (see [4]). Following such theory, scholastic logicians came to the formulation that, for any species, “the definition is given by the closer genus and the specific difference”\textsuperscript{1} or, shortly, by genus and differentia. As an example, let us consider the well known aristotelian statement according to which:

\textit{A human being is an animal that has the capacity to reason.} \hspace{1cm} (3.3)

In this statement, the cluster of words “human being” acts like the \textit{definiendum}, i.e. the textual surface identification of the concept we are going to define. The word “animal” identifies the \textit{genus proximus}, i.e. that species that is close to the one we are defining and such that everything that can be predicated of an individual from such species, can be predicated of any individual of the species we are defining as well. Namely, the \textit{genus proximum} can be consider a hypernym of the \textit{definendum}. The final part of the definition is the \textit{differentia specifica}, namely the description of some peculiar characteristics of the \textit{definiendum} that differentiates it from the genus. In our example, such duty is fulfilled by the cluster of words “has the capacity to reason.” Authors in [59], tackling a problem of hypernym extraction, consider as proper definitions all the sentence following the aristotelian scheme, where at least the text surface realization of the \textit{genus} is not empty. As a consequence, a definition representing a pure taxonomical relation—like “cars are vehicles”—is considered a valid definition. Conversely, a sentence defining a species with an empty genus—like “cars have four wheels”—is not considered as a proper definition. On the opposite, the latter type of sentences are considered as valid definitions to the extent of this work. Indeed,

\textsuperscript{1}“\textit{definitio fit per genus proximum et differentiam specificam}.”
such sentences provide useful characterization of the definiendum, even without expressing explicitly the genus, that can be of interest for an ontological representation. We call all the sentences of interest for our work descriptive sentences, since they describe a salient feature of a set of entities. We use the expression Descriptive Language to denote the subset of natural language comprising all the descriptive sentences.

3.1.2 Second Challenge: Choosing the Target Language

We set the Description Logics language $\mathcal{ALCQ}$ as our target language. This choice can be motivated in terms of different theoretical and practical reasons.

We aim at a logical language and, more in detail, we are interested in Description Logics. These are the standard logic languages behind OWL-DL, a computable and tractable fragment of OWL$^2$, an ontological language which is the de facto standard for the Semantic Web community (for an introduction to OWL, see [3]). The Description Logics language on which OWL-DL is built is $\mathcal{SHOIN}^D$. The expressiveness of this language goes way beyond the definition of concepts, which is what can be expressed by our source language. As a consequence, we pruned all those constructs concerning roles and not directly related to the definition of concepts. Namely, we removed all those constructs related to nominals and individuals, since they do not fall within the scope of the present work, and the local reflexivity construct, which express some kind of knowledge more related to relations than to concepts. The result of this activity of pruning, is indeed the $\mathcal{ALCQ}$ language.

Summing up, we used the $\mathcal{ALCQ}$ Description Logics language as our target language, since it covers all the constructs we need for the definition of concepts, leaving outside those constructs from Description Logics (like inverse object properties or transitive role) which are beyond the scope of

this work. Hereafter, when using the term *formula* we will refer to a well-formed \(\mathcal{ALCQ}\) formula expressing a concept definition.

### 3.2 The Approach

In Section \ref{sec:problem}, we described the problem addressed in this thesis. In this Section we will explain which kind of approach we followed to solve this problem.

We exploited recent advances in the field of Deep Learning in order to train a neural architecture in an end-to-end fashion, accepting raw text as input and producing the corresponding Description Logic formula as output. We did not use any other intermediate explicit representation or any external feature coming from some other pre-trained Natural Language Processing toolkit in order to reduce the feature engineering efforts. As already said in Chapter \ref{chap:background} our system is trained by examples, in a supervised learning fashion. This means that no hand-crafted transformation rules are required and all the variability of the natural language syntax is learnt from the training examples.

Following this approach, the main challenge dwells in the nature of the natural language itself. Natural language has an intrinsic variability, which emerges both from a syntactic and a semantic point of view.

To exemplify the idea of what the **syntactic variability** of the language is and why it is critical for a Machine Learning based approach, let us start remarking how the piece of knowledge formalized in formula in (3.2) can be expressed, beside (3.1), also in different sentences. As an example, let us
3.2. THE APPROACH

CHAPTER 3. FROM DEFINITIONS TO FORMULÆ

consider the following ones:

\[
\begin{align*}
\text{Bees are insects that produce honey,} & \quad (3.4) \\
\text{A bee is also an insect that produces honey,} & \quad (3.5) \\
\text{Every bee is an insect and it also produces honey.} & \quad (3.6)
\end{align*}
\]

Now, let us assume that all such sentences are actually present among the training examples. Our system is actually learning the underlying structure of the problem space, and not just memorizing the association between each input and the relative expected output, if it is capable to process any unseen (i.e. not in the training set) sentence combining different parts of the training material, as an instance:

\[
\text{every bee is also an insect that produces honey.} \quad (3.7)
\]

Ideally, our system learns how to use the word “every” from (3.6), the word “also” from (3.5) and the word “that” from (3.4) and (3.5), and then is able to recombine them when they appear in the different context of (3.7).

With the expression semantic variability we denote a broad set of phenomena related to the meaning of words in the context of the sentence they appear in. Since to the scope of the present investigation we have been dealing only with syntactic transformations, the semantics of each word is grounded in the word itself and into the role it acts within the sentence,\(^3\) as highlighted with the example on (3.1) and (3.2). So, linguistic phenomena like semantic ambiguity, polysemy, and so on, are not taken into account to the scope of the present work. Though, even considering only plain words, we must consider that natural language is build on an extremely large vocabulary and not all the words, appearing in each possible function within a

\(^3\)To clarify with an example: think to the words “model” and “train”. In the sentence “I train a model”, the first behaves like a verb and the second like a noun, while in the sentence “I model a train”, it is the opposite.
sentence, can be actually included among the training examples. As a consequence, the model should be capable to infer the meaning and the role of a certain word and behave properly. As an example, consider a training set made of the sentences seen so far, plus the following

\[ a \text{ cow is a mammal that eat grass}, \]  

which can be considered as corresponding to the formula:

\[ \text{cow} \sqsubseteq \text{mammal} \cap \exists \text{eat gr}ass, \]  

so that our network has now also knowledge of “cow” and “mammal”. Given the unseen sentence

\[ a \text{ cow is a mammal that produces milk}, \]  

the network should be capable to understand that the milk has, for a cow, the same role of honey for a bee, and consequently output the formula

\[ \text{cow} \sqsubseteq \text{mammal} \cap \exists \text{produce milk}. \]  

The few example sentences listed before can already give us an intuition about some structural characteristics of natural language that should be considered in our scenario. Nouns such as “bee”, “insect”, “honey”, “cow”, “mammal”, “milk”, and “grass” denote the concepts involved in the domain described by the different sentences, while verbs like “is”, “produce”, or “eat” describe relationships occurring among such concepts. Word classes like nouns, verbs, adjectives and some adverbs are considered, from a linguistic standpoint, content words since they describe the actual content of a sentence and are strongly denoted in their lexical semantics. Dually, word classes like articles, pronouns, conjunctions, determiners, most adverbs, and so on, are considered function words: they express grammatical relation-
3.2. THE APPROACH

CHAPTER 3. FROM DEFINITIONS TO FORMULÆ

ships between words without carrying any lexical meaning—see words like “a”, “that”, “also”, “that”, “every” in the sentences presented above. Roughly speaking, two sentences that convey the same meaning will tend to share the same content words, even if their function words are different, like the above sentences (3.1) and (3.7). Vice versa, two sentences that are similar with respect to their grammatical structure, having similar function words, can present different content words, having very different meanings, like the above sentences (3.1) and (3.8). For a deeper discussion on the different classes words can fall into with respect to their syntactic role, we refer to Chapter 5 from [43].

To endow a Neural Network with such abilities, we have to operate in two complementary perspectives, which define the two main challenges we had to face to effectively use Neural Networks to translate definitions into formulæ by means of syntactic transformation.

3.2.1 First Challenge: the Dataset

First, the training set must be a valid sample of the problem space, so that the network is provided with as many different examples as possible and it is capable to learn a significant representation of the phenomenon under observation. This means that the training examples should contain as many different grammatical structure as possible and as many content words as possible, so that the network can interpolate across different grammatical constructs and over a large portion of vocabulary—in order to correctly process sentences (3.7) and (3.10) as said above.

The rationale behind the choice of the datasets used to the extent of this thesis is presented in Chapter 4.
3.2.2 Second Challenge: the Architecture

At the same time, given a dataset fulfilling all the desiderata expressed in 3.2.1, the network architecture must be powerful enough to extract such representation and generalize across its regularities. Note that, since we are using only raw text, the network architecture must be capable to learn also the syntax of the portion of natural language under investigation. Finally, since we are dealing with syntactic transformation, we must ensure that all the output symbols of the networks are logical symbols from the target Description Logics language, or words from the input sentence, acting as extralogical symbols.

To address this second challenge, we followed the recent advances in Deep Learning, mostly in so-called Recurrent Neural Networks, which are particularly suitable for dealing with natural language. Our investigation was developed into two main stages, in which we embodied the approach proposed here in two different ways. The two different architectures, together with their evaluation, will be presented in Chapter 5 and Chapter 6.
Chapter 4

Dataset for Training and Evaluation

In this Chapter, we present the rationale behind the construction of the datasets we exploited to train and evaluate the different models we investigated. Each of these models has some structural peculiarity that constrain both the type of training examples to be generated, and the format in which the examples are encoded. Thus, the general idea presented in this Chapter has to be implemented differently, depending on the particular model. The two different forms in which we embodied the data generation process are described, together with the model the data have been used for, in Chapter 5 and Chapter 6.

Later, we present a manually curated dataset that we developed manually annotating 500 natural language definitions covering different domains with the corresponding ALCQ formulæ. In the context of the present work, this dataset has been used to assess the capability of the model under evaluation to gain more knowledge on the real problem space through the extension of the training set by means of manually annotated examples.

4.1 Motivations and Desiderata

Being Neural Networks statistical learning-based models, they must be trained by examples. Therefore, we need to collect a set of examples, namely a set
of sentence and corresponding formula pairs, to be submitted to the Neural Network during the training and the evaluation phases. As outlined in Chapter 3, collecting this set of examples is one of the challenges we had to face in this thesis. Indeed, besides the difficulty of producing annotated examples, they must be a significant sample of the phenomenon we are trying to approximate with our Neural Network, namely the correspondence between a sentence and a formula. Recalling the simple driving example presented in Chapter 3, our ideal dataset should have the following characteristics:

- it should cover as many as possible syntactic structures that can be used by a human to correctly express a definition: in this way, the network can be trained to recognize the purpose of different function words and the syntactic structures they define;

- it should cover a significant portion of the human vocabulary: in this way, the network can be trained to recognize and interpret as many content words as possible and to be tolerant to words that are not present during the training phase.

To the best of our knowledge, the Semantic Web and Knowledge Engineering communities lack a dataset satisfying the desiderata presented above. This may be due to the fact that a pure Machine Learning approach for the task dealt with in this thesis has never been proposed.

Building such a dataset manually from scratch was not an option for us. Indeed this process would be extremely costly and time consuming, requiring considerable human-effort to collect, annotate, and validate a large quantity of data. As a reference, building the Penn TreeBank corpus for the task of Part-of-Speech tagging required the manual annotation of 40,000 training sentences and 2400 test sentences, the equivalent of 4.5 million words (see [51]). Moreover, the task of selecting the Part-of-Speech requires annotators that have a basic knowledge of the English grammar, while translating
a definition into a Description Logic formula would require highly trained annotators. In order to have a first estimation of the effort that the building process of such dataset would have required, we actually ran a limited annotation campaign involving manual annotators. From the experiment, we observed that a single annotator can produce 100 annotated examples in about 70 hours. We need to remark that the vast majority of this amount of time was devoted to scan collections of large encyclopaedic entries in search for descriptive sentences suitable as input sentences for our task. Moreover, this calculation keeps in account also some supervision activities that have to be performed to check the inter-annotator agreement and perform possible corrections. On the opposite, we did not consider the amount of time spent in the initial training of the annotators, since this training process is supposed to be performed una tantum.

Instead of manually annotating examples, a possible alternative was to use some distant supervision approach, like the one followed in [64]. Following this approach, when a large collections of input and output data are separately available, some heuristics or external systems are used to align some items from the input collection to some other from the output. In our case, following this approach would require large catalogues of natural language definitions and a large set of expressive Ontologies, both covering a large set of concepts. Even in this case, to the best of our knowledge, there are no such collections of suitable input and output data.

Therefore, to collect some dataset to train and evaluate our models against, we followed the practice of some notable examples in the literature (see [33, 93, 90, 89, 63, 11, 32, 65], just to name a few), setting up a data generation pipeline to produce extensive synthetic datasets.
4.2 Data Generation Process

In this Section, we describe the data generation pipeline that we run to obtain datasets to train and evaluate our models. Depending on the characteristics of the models, this pipeline can be practically implemented in different ways. In our investigation, we evaluated two different models, each of which required a different instantiation of the pipeline described below. These models and the associated data generation processes will be described in Chapter 5 and Chapter 6.

The training data generation process starts with a Context-Free Grammar which is capable to generate sentence templates. A sentence template $\bar{s}$ is a descriptive sentence in which the text surface expressing the involved concepts, the roles between pairs of such concepts, and the cardinal numbers involved in quantified cardinality restrictions are replaced by placeholders. We used $C$ as a placeholder for concepts, $R$ for roles, and $NUM$ for cardinal numbers. Such placeholders are augmented with an ordinal number representing the order of each type of placeholder in the sentence template. An example of a sentence template can be the following:

$$A \ C_0 \ is \ a \ C_1 \ that \ R_0 \ at \ least \ NUM_0 \ C_2,$$  \ (4.1)

where $C_0$, $C_1$, and $C_2$ are placeholders for text surface realizations of concepts, $R_0$ for a role, and $NUM_0$ for a cardinal number. The CFG is hand-crafted according to some description about the particular portion of the linguistic universe we want to model.

The same Context-Free Grammar used to generate the sentence template, is then used to parse the sentence template itself, generating a parse tree that represents the syntactic structure of the sentence with respect to the grammar. Such parse tree is then used by a set of transformation rules that turn it into a formula template, namely a Description Logic formula in which
the extralogical symbols are the corresponding placeholders from the sen-
tence template. So, given the sentence template in (4.1), the corresponding
formula template, say \( \bar{f} \) will be the following:

\[
C_0 \sqsubseteq C_1 \sqcap \geq \text{NUM} \circ R_0 \cdot C_2.
\]

(4.2)

To have a proper example, both the sentence template and the formula
template must finally go through what we call the \textit{actualization} stage,
producing respectively a sentence, say \( s \), and a formula, say \( f \). In the ac-
tualization stage, the placeholders for concepts, roles, and ordinal numbers
are filled with actual words, representing their text surface realizations, both
in the sentence template and in the formula template. Concepts are actual-
ized following a two-step process. First a pattern of Part-of-Speech tags and
prepositions is selected, then random words from the proper word class are
used to replace the Part-of-Speech tags. As an example, a concept could be
actualized through the pattern \( \text{JJ NN of NN} \), consequently realized at the
surface level with the noun phrase “\textit{sad wings of destiny}” or “\textit{magnificent}
\textit{sword of doom}”. Role placeholders are filled with random (transitive) verbs.
Finally, for ordinal numbers, we used different actualization strategies that
will be explained separately, within the context of the proper evaluation, in
Chapter \[ and Chapter \]. As an example, the sentence template in (4.1)
could be actualized in the following sentence:

\[
\text{A multi-instrumentalist is a smart musician}
\text{that plays at least 2 instruments}
\]

(4.3)

and the corresponding formula:

\[
\text{multi-instrumentalist} \sqsubseteq \text{smart musician}
\text{‰} \geq 2 \text{play.musical instrument},
\]

(4.4)
The resulting pair \((s, f)\) is said to be a **grammar generated example**. This data generation pipeline, synthetically depicted in Fig. 4.1 can be run multiple times to generate a desired amount of grammar generated examples to be used to train and evaluate the network. The sentences that are generated in this way are *actual* natural language sentences since they are grammatically correct, even if potentially unrealistic from a human point
of view. As an example, the sentence “a smoking hurricane is also something that pumps at least 18 orange stampedes” and the corresponding formula $\text{smokinghurricane} \subseteq \geq 18 \text{pump.orangestampedes}$ may be part of the dataset. Note that all the extralogical symbols appearing in the grammar generated formulae are words from the sentence. In this way, the idea of syntactic transformation as expressed in Chapter 3 is preserved.

In practice, the actualization is performed in a way so that the grammar generated sentences have the following characteristics:

- all the text is lower-cased;
- all the occurrences of the indefinite article “an” are replaced with the form “a”;
- all the nouns and verbs are lemmatised;
- all the occurrences of “does not” and “doesn’t” are replaced with “do not”.

These simple hack, allowed us to reduce the number of necessary examples. In addition, a special $<$EOS$>$ symbol denoting the end of sequence, is appended at the end of both the sentence and the formula. Such symbol has no meaning from the conceptual point of view, but it is used by the neural network to establish the actual boundaries of the input sequence (the sentence) and of the output sequence (the formula). Note that even real world text can be easily made compliant to all the constraints expressed above, with basics pre-processing operations.

Since one of the training objectives is to make the network robust to unknown words, namely words that have not been seen during the training phase but might appear in the evaluation phase, some words used to build the text surface realization of concepts and roles, are replaced with a special symbol $<$UNK$>$ denoting a generic unknown word. Such replacement has been
performed randomly, following different policies that will be described within the context of the actual experiments, see Chapter 5 and Chapter 6.

### 4.3 Discussion on the Grammar Generated Data

By running the process described in Section 4.2, we can produce datasets that are suitable with respect to the desiderata presented in Section 4.1. The tolerance to unknown words can be easily evaluated with the replacement of content words with the `<UNK>` symbol. The ability to generalize over different syntactic structures can be assessed by training the network on a small fraction of all the possible sentence and formula templates that can be generated by the grammar, and by evaluating its performance on a set of examples, built starting from templates not used for the training examples. This ensures that the Neural Network does not just memorize the training examples but learns from them how to deal with the different syntactic structures presented in the training phase and how to generalize over them.

Moreover, with the support of Figure 4.2, we want to clarify the relation between our grammar generated data and the actual universe of language used by human to express concept definitions. The large circle, in black, represents the universe of the Descriptive Language as defined in 3.1.1. Our grammar generated data are an approximation of this universe, depicted in the figure with the red dots. The blue circle represents a sentence template. All the sentence that can be realized through the actualization of the same sentence templates are a sub set of our approximation of the universe of language. In the figure, the sentence template “C0 is a C1” is represented by the blue circle. All the dots comprised within the circle, are all the sentences that can occur from the actualization of this template. In particular, we highlighted two examples, represented by blue dots. When we generate a dataset to train and evaluate a model, we are basically sampling from our
A cow is a mammal.

A bee is an insect.

Figure 4.2: Sampling over the universe of language used for definitions with grammar generated examples.

approximation of the universe of language.

A final remark, is about the way in which we built the set of rules used to process the parse tree of the sentence template in order to turn it into the formula template. Such set of rules can be seen as an automation of the annotation process. In other words, such rules act the same way of a set of guidelines for human annotators. Concerning the way we generated the formula templates, we are aware that the very same sentence can be formalized in many different ways, even only within the scope of a syntactic
4.4 COLLECTING MANUALLY BUILT EXAMPLES

Chapter 4. Dataset

Transformation—one concept like “long and winding road” can be formalized as long_road □ winding_road or long □ winding □ road. Being a system that learns from examples, our trained model will learn to translate sentences into formulae according to such guidelines. Clearly, different guidelines in the annotation of the training examples would lead to a substantially different dataset, and thus a substantially different formalization schema learned by the model. This does not hamper our evaluation, since the goal of this work is neither to define nor to verify that the model learns the best axiomatization scheme, but just the one underlying the training set.

4.4 Collecting Manually Built Examples

In addition to the synthetic datasets obtained running the instantiation of the process described above, we also developed a manually curated dataset consisting of 500 sentence-formula pairs. This dataset was collaboratively developed by three ontology engineers in three different manners: first, definitory sentences and corresponding axioms were collected from available ontologies (such as the Pizza Ontology, the SSN Ontology, the VSAO Ontology, the Wildlife Ontology, the Transport Disruption Ontology, the OBCS Ontology, the Ontology Transportation Network, and other ontologies on Atoms) and their documentation (e.g., ontology comments); second the ontology engineers have formalized in $\mathcal{ALCQ}$ sentences from textual resources such as glossaries (e.g., the NeON and the SEKT projects); third they

---

1. https://protege.stanford.edu/ontologies/pizza/pizza.owl
2. https://www.w3.org/2005/Incubator/ssn/ssnx/ssn
3. https://bioportal.bioontology.org/ontologies/VSAO
4. https://www.bbc.co.uk/ontologies/wo
7. https://www.pms.ifi.lmu.de/reverse-wgai/otn/OTN.owl—at the time of writing, this URL is not accessible.

76
CHAPTER 4. DATASET 4.4. COLLECTING MANUALLY BUILT EXAMPLES

where asked to create and formalize typical definitory sentences just for the present task.

The process for creating this manually curated dataset was as follow. One ontology engineer provided a sentence-formula pair, either collecting it from an existing ontology or formalizing a selected definitory sentence, while the other two validated the provided formalization. In case of disagreement between the three ontology engineers, the example was further discussed and collaboratively revised.

<table>
<thead>
<tr>
<th>name</th>
<th>size</th>
<th>min. len.</th>
<th>max. len.</th>
<th>avg. len.</th>
<th>exist.</th>
<th>univ.</th>
<th>card. restr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>500M</td>
<td>500</td>
<td>5</td>
<td>40</td>
<td>12.26</td>
<td>49.60%</td>
<td>4.20%</td>
<td>9.20%</td>
</tr>
<tr>
<td>75M</td>
<td>75</td>
<td>5</td>
<td>28</td>
<td>11.72</td>
<td>42.67%</td>
<td>2.67%</td>
<td>9.33%</td>
</tr>
<tr>
<td>425M</td>
<td>425</td>
<td>5</td>
<td>40</td>
<td>12.36</td>
<td>50.82%</td>
<td>4.47%</td>
<td>9.18%</td>
</tr>
</tbody>
</table>

Table 4.1: Manually curated dataset and splits.

After its creation, the dataset was manually split into a training part (75M, 75 pairs) and a testing part (425M, 425 pairs), trying to preserve the same distribution of axiom structures (e.g., simple subclasses, subclasses with one existential/universal/cardinality restriction, etc.) among the two resulting datasets. Their main characteristics are summarized in Table 4.1 both for the dataset as a whole and for the splits.
Chapter 5

Ontology Learning by Tagging and Transduction

In Chapter 3 we presented the idea of translating concepts definitions expressed in natural language into Description Logics formulæ by means of syntactic transformation. In this Chapter we present our first implementation of a model intended to realize such process. This approach has already been presented in one of our previous works, namely [65]. We designed and evaluated a system made of two different neural networks, combined together. First, we describe the main intuition behind this implementation in Section 5.1, and then, in Section 5.2 and 5.3 we give a technical description of the network architectures of the model. In Section 5.4 we will describe the evaluation procedure, stating the research questions we investigated (5.4.1), the evaluation metric we used to quantitatively measure the performance of the model (5.4.2), the data that we used for the training and the evaluation phases (5.4.3) and, finally, the experimental settings and the results (5.4.4). The approach presented in this Chapter is critically discussed in Section 5.5.
5.1 The Intuition

As stated at the beginning of Chapter 3, we intend to translate a sentence into a formula by means of syntactic transformations. This means that the resulting formula can be seen as a sequence of symbols, some of them drawn from the list of logical symbols of the target logical language, and the other being extralogical symbols occurring as words in the input sentence. The main intuition underlying the first Ontology Learning technique we present in this Chapter is that a sentence could be translated into a formula through two parallel processes that we illustrate with the help of an example. Given the sentence:

\[ A \text{ bee is an insect that produces honey,} \]  

and the corresponding formula:

\[ \text{bee} \sqsubseteq \text{insect} \sqcap \exists \text{produce} \cdot \text{honey}, \]  

it is easy to see that it can be represented in abstract terms using the following formula template:

\[ C_0 \sqsubseteq C_1 \sqcap \exists R_0 \cdot C_2, \]  

where \( C_0, C_1, \) and \( C_2 \) represent concepts while \( R_0 \) represents a role. These symbols will be hereafter called placeholders. Formula (5.2) can be obtained from the formula template (5.3) by filling the placeholders with proper words from the sentence (5.1), acting as extralogical symbols in the resulting formula. With respect to the example above, the placeholder \( C_0 \) is filled with the word “bee”, \( C_1 \) is filled with “insect”, \( R_0 \) is filled with “produces”, and, finally, \( C_2 \) is filled with “honey”. We can rewrite the sentence, augmenting each word with the label of the placeholder to be filled. If a word does not
fill any placeholder, we will use a conventional empty tag $w$, writing:

$$ A/w \ bee/C0 \ is/w \ an/w \ insect/C1 \ that/w \ produces/R0 \ honey/C2 \quad (5.4) $$

Note that we could have had a sentence where concepts and roles are expressed with multi-word terms, like the following:

An excellent bee is an insect that produces high quality honey. \hspace{1cm} (5.5)

In this case, always referring to the formula template \((5.3)\), the placeholder $C0$ would have been filled with the words “excellent” and “bee”, while the placeholder $C2$ would have been filled with the words “high”, “quality” and “honey”.

According to many experiences in the literature (e.g. see \[33, 31\]), we call \textbf{transduction} the process of turning an input sequence of symbols, namely the sentence, into an output sequence of symbols, namely the formula template. The process of identifying the proper placeholder for each word has been envisioned as a \textbf{tagging} process, where the label for the proper placeholder or the empty label $w$ is assigned to each word. Running these two processes in parallel, it is easy to rebuild the final formula, as depicted in Figure 5.1, just by replacing the placeholder in the formula template with the words from the sentence that have been tagged with the placeholder itself.

\begin{center}
\begin{tikzpicture}

\node (a) at (-2, 0) {A bee is an insect.};
\node (b) at (0, 0) {$C0 \sqsubseteq C1$};
\node (c) at (0, -1) {$w \ C0 \ w \ w \ C1 \ w$};
\node (d) at (2, 0) {bee $\sqsubseteq$ insect};

\draw[->] (a) -- node[above] {Transduction} (b);
\draw[->] (b) -- (c);
\draw[->] (c) -- node[below] {Tagging} (a);
\draw[->] (b) -- (d);
\end{tikzpicture}
\end{center}

\textbf{Figure 5.1: Sentence into Formula through Tagging and Transduction.}
5.2 Tagging Architecture Description

The Neural Network model used for the tagging task is an extremely simple one. The input layer is in charge of producing a continuous representation of the words in a continuous vector space model. At the $i$-th timestep, each index representing the $i$-th of the sentence is fed into the embedding layer that, as described in 2.2.4, returns the word vector for the given word, $x_i$. More in detail, for each word, we consider also some of the preceding and the following words, in order to keep in account some of the local context. This operation is called windowing, and the total number of words to be considered before and after the $i$-th one, $w$, is called window size and is a hyper-parameter to be set at design time. So, the $i$-th word in the sentence will be represented by the concatenation of the word vectors of all the words in the window. We will denote such windowed word vectors as $x_{i \pm w}$. From the practical point of view, the windowing operation is performed at the very beginning of the network. At the $i$-th timestep, a list of indexes representing a cluster of words centered in the $i$-th position is submitted to the embedding layer, which projects each of them into the proper word vectors and concatenates them into $x_{i \pm w}$. We indicate the window of indexes around a certain word $V(x_{i \pm w})$ where $x_i$ is the $i$-th word of the sentence.

Such vectors are subsequently fed into a recurrent layer, using GRU cells as described in 2.2.4 on top of which the output layer is stacked. Such output layer is in charge, at each timestep, to read the activation of the recurrent layer and predict which tag must be attached to the corresponding word. To obtain such prediction, we exploit a softmax layer. Such layer will produce at the $i$-th timestep a vector $y_i$ defined as:

$$y_i = \text{softmax}(W_{out}^T h_i + b_{out}), \quad (5.6)$$

being $W_{out}$ a matrix in $\mathbb{R}^{|T| \times h}$, and $b_{out}$ a vector in $\mathbb{R}^{|T|}$, where $|T|$ is the
size of the tag vocabulary and \( h \) is the hidden state. Recall that the hidden state is the output of a recurrent cell and, if the recurrent layer is made of several stacked cells, it is the output of the topmost one. As a result, both the vector resulting as the argument of the \( \text{softmax}(\cdot) \) function and \( y_i \) will be vectors in \( \mathbb{R}^{|T|} \). Applied to a generic vector, say \( \mathbf{v} \), the \( \text{softmax} \) function produces another vector \( \mathbf{y} \) of the same dimension, where the value of the \( j \)-th component is given by the:

\[
y[j] = \frac{\exp(v[j])}{\sum_{k=1}^{|T|} \exp(v[k])}.
\]

(5.7)

Note that, in this way, each component of vector \( \mathbf{y} \) falls in the range of \([0, 1]\) and it is easy to show that they all sum up to 1. Because of these two characteristics, we can use vector \( \mathbf{y} \) to approximate a probability distribution over a discrete random variable, having a number of possible outcomes equal to the size of the vector itself. In our case, being the size of \( y_i \) equal to \(|T|\), it can be seen as a probability distribution over all the possible tags. We call \( y_i \) a prediction vector for the \( i \)-th timestep. The position of the component with the highest probability within \( y_i \), say \( y_i \), will identify within \( T \) the predicted tag at the \( i \)-th timestep:

\[
y_i = \text{argmax}(y_i).
\]

(5.8)

Given the prediction vector \( y_i \), let us call \( \hat{y}_i \) the gold truth one, being the latter a one-hot representation of the correct tag for the word at the \( i \)-th timestep. Said \( N \) the number of timesteps, namely the number of words in the input sentence \( s \), the difference between the predicted sentence and the gold truth one is expressed through the so called categorical cross entropy function:

\[
\mathcal{L}_s = - \sum_{i=1}^N \hat{y}_i \log(y_i).
\]

(5.9)
which is the loss function that we used in all our experiments. Such function is extremely popular when training deep architectures for several properties. First, it is convex and can be optimized through Stochastic Gradient Descent methods. Moreover, the gradient tends to vanish later than when using other loss function, like Mean-Squared Error. Note that being $\hat{y}_i$ a one-hot vector, the dot product with $y$ will end up just in the component of the latter corresponding to the 1 value in the former. Because of the softmax normalization, all the components in $y_i$ are in the range $[0, 1]$; therefore, each logarithm in the summation will be a negative number, so $L_s$ is ensured to be a positive quantity by the minus sign in front of the summation. At each batch of examples $S$ in the training phase, the training objective is to minimize the value $L$ obtained summing all the losses for each example in the batch:

$$L = \sum_{s \in S} L_s.$$  \hfill (5.10)

Minimizing this loss, we adjust the values of the parameters to be learnt during the training phase, namely the word vectors, the parameters of the recurrent layer and $W_{out}$ and $b_{out}$ from the (5.6).

Figure 5.2: Network architecture for the Tagging task
The full architecture is depicted in Figure 5.2, which must be read from the bottom upwards. First, each word is mapped into its corresponding index within the vocabulary $V$, then the windowing operation is performed—these two operations are represented as a single layer in Figure. After that, the word embeddings for each window are computed, concatenating the word vectors of all the words in the window. Subsequently, such vector is fed as the input of a recurrent layer and, at each timestep, the activation of such layer is processed by a softmax layer in order to obtain a prediction over all the possible tags.

### 5.3 Transduction Architecture Description

The network used for the Transduction process has the same building blocks of the one designed for the tagging process, namely: the embedding layer, the recurrent layer, and the softmax layer. Though, the way they are assembled is radically different. Indeed, since the tag sequence produced by the tagging process is made of exactly one tag for each word, it is strictly coupled to the input sentence, both from the semantic and from the syntactic standpoint. Instead, transducing a sentence into the corresponding formula template requires the input and the output sequence to be strictly coupled from a semantic point of view only. Indeed, the input sentence and the formula template convey the same meaning from a logic based point of view but the two sequences are typically loosely coupled from a syntactic point of view. As a reference, consider that both the sentences “A bee is an insect” and “every bee is also an insect” would be coupled with the same formula template $C_0 \sqsubseteq C_1$, regardless of the differences between their syntactic forms. Neural models like the one here presented, capable of this particular way of translating a sequence into another, are commonly referred to with the umbrella category of Sequence-to-Sequence models.
We can obtain a network capable to preserving the same meaning even across different structures exploiting a so-called Recurrent Encoder-Decoder architecture like the one presented in [16], where two different recurrent layers are used in order to decouple the structure of the input and the output sequences, keeping them semantically tied. First, a recurrent layer is used to process the whole sentence. The activation corresponding to the last timestep, say $c$, is used as a distributed representation of the whole sentence at once. Such context vector, is then used as the constant input of a second recurrent layer. The activations of the latest recurrent layer are subsequently fed into a softmax layer that outputs a prediction vector over the vocabulary of all the formula terms, $F$. The first recurrent layer is called encoder, since it encodes the whole input sequence into a single vector. Conversely, the second recurrent layer is called decoder, since it decodes the distributed representation of the whole sentence into a new sequence, namely the output formula template.

![Network Architecture Diagram](image)

**Figure 5.3:** Network architecture for the Transduction task.

86
Since the mathematical description of each component of this architecture has already been presented in Section 5.2, we will not go into further details. The whole model architecture is depicted in Figure 5.3, which must be read from the bottom upwards. First, each word is mapped into its index within the input vocabulary $V$ and subsequently into the corresponding word vector through the embedding layer. Such word vectors are subsequently fed into the first recurrent layer, which acts as an encoder. The last activation of this layer is considered a distributed representation of the whole input sentence and is constantly fed into another recurrent layer, acting as a decoder. Such decoder produces a further set of vectors, called decoder states, one for each timestep of the output sequence. Finally, each decoder state is fed into a softmax layer that produces a prediction vector over all the possible terms to appear in the formula.

5.4 Evaluation

In this Section, we will describe the evaluation of the approach and the networks described above. First, we state the objectives of our evaluation by means of two research questions (5.4.1), and define the metric that we used to quantitatively measure the performance of the models and provide answers to the research question (5.4.2). After that, we describe the way we generated the datasets for the training and evaluation of the model (5.4.3). Finally, experimental settings and result are presented and discussed (5.4.4).

5.4.1 Research Questions

Neural Networks, like all Machine Learning algorithms, are a particular class of statistical learning models. Every machine learning based solution starts with a sampling operation of the real world phenomenon we want to model.
The result of this sampling operation is a collection of examples that are used to train and validate the model. The larger is the number of drawn examples, and the more they are homogeneously spread across the whole problem space, the more the network will be able to generalize over such examples and process correctly also sentences that have not been seen during the training phase. In our scenario, the generalization capabilities of the model have to be intended both from the syntactic perspective, in the sense that the model must be able to deal with different sentence structures, and from the lexical perspective, meaning that the model must be able to deal with a large part of the human vocabulary and, most important, to tolerate words that have not been seen during the training phase. We refer to these unseen words as unknown words. Increasing the number of training examples is often a good practice to boost the performance of the model, but it is typically an expensive operation. So, one pivotal feature of any machine learning model is the ability to generalize as more as possible in presence of a limited amount of examples. These considerations led to two research questions we addressed in our evaluation process:

**RQ1.** To what degree is the network capable to generalize over the syntactic structures of descriptive language?

**RQ2.** To what degree is the network capable to tolerate words that have not been seen during the training phase?

In answering these research questions, we have to remember the relationship between the dataset and the network architecture. As explained in Chapter 4, the synthetic data generation process is intended to provide an approximation of the universe of the descriptive language. Given a network architecture, we can evaluate its capabilities of generalization over the syntactic structures of the descriptive language using training sets of different sizes. The smaller this size is, the less likely the different syntactic constructs
will co-occur. As a consequence, reducing the training set size, will furtherly
force the network to generalize over the training examples. Similarly, adding
unknown words to our universe of the language can reveal how much the
network under evaluation is capable to leverage the context of the sentence
to handle them.

5.4.2 Evaluation Metrics

In our evaluation, we quantitatively evaluated the performance of the model
using three different metrics that are described below. These evaluation
metrics compare the expected formula together with the actual one, which is
built from the output of the two networks, replacing the placeholder within
the formula template according to the tag that has been predicted for the
words in the sentence.

The resulting formula $f$ is a sequence of $T_f$ symbols. For each predicted
formula $f = f_1, \ldots, f_{T_f}$ generated by the model for a sentence $s$. We indicate
the expected one with $\hat{f} = \hat{f}_1, \ldots, \hat{f}_{T_f}$, namely the one that the model should
have been predicted for such sentence. We say that the predicted formula
and the corresponding expected one are equals, $f \equiv \hat{f}$, if $f_j = \hat{f}_j$ for each
$j$ in $[1, T_f]$. Given a list of $M$ sentences $S = s^1, s^2, \ldots, s^M$ we indicate with
$F = f^1, f^2, \ldots, f^M$ the list of the predicted formulæ such that $f^k$ is the
formula that the model actually generated with $s^k$ as input. We indicate as
$\hat{F} = \hat{f}^1, \hat{f}^2, \ldots, \hat{f}^M$ the list of expected formulæ, such that $\hat{f}^k$ is the correct
formula for $s^k$. On those two lists, we have defined the three different metrics
to be used in the evaluation.

**Average Per-Formula Accuracy**

As our first metric, we define the Average Per-Formula Accuracy ($FA$) as
the ratio between the number of correctly predicted formulæ, $CF$, and the
5.4. EVALUATION

The total number of formulæ, $M$:

\[
FA(\hat{F}, F) = \frac{CF}{M} = \sum_{k=1}^{M} \begin{cases} 
1, & \text{if } f^k \equiv \hat{f}^k \\
0, & \text{otherwise}
\end{cases}
\]

The value of $FA$ is a real number in the range $[0, 1]$. The higher the value of $FA$, the better the network is performing. For the sake of clarity, sometimes we may need to report also the raw number of correctly predicted formulæ, $CF$.

**Average Levenshtein Distance**

The second metric is intended to take into account a realistic scenario in which a human operator exploits the system to translate a set of definitions into a set of formulæ. The main idea is that the less corrections the human operator will have to perform on the system output, the more the system can be considered accurate. Since we are dealing with syntactic transformation, so that all the extra-logical symbols of the target language are taken from the input sentence, the set of such possible corrections will be limited for each sentence-formula pair. In this scenario, the Levenshtein Distance can be extremely meaningful, since it measures the number of transformations that must be done on the predicted formula in order to turn it into the correct one. These transformations are basically of three types: inserting a missing symbol, deleting a symbol, replacing a wrong symbol with the right one. We define the Average Edit Distance ($ED$) as:

\[
ED(\hat{F}, F) = \frac{\sum_{k=1}^{M} \delta(f^k, \hat{f}^k)}{M}
\]

where $\delta(f^k, \hat{f}^k)$ is the Levenshtein Distance between the predicted formula $f^k$ and its gold truth $\hat{f}^k$. Since $\delta(f^k, \hat{f}^k)$ is a natural number greater or equal
than 0, the value of $ED$ is a real number greater or equal than 0. The lower the value of $ED$, the better the network is performing.

**Average Per-Token Accuracy**

The last metric is the Per-Token Accuracy ($TA$), which indicates how many symbols have been correctly predicted across the whole list $\mathcal{F}$. This metric is useful to give an overall evaluation of the translation capabilities of the model. It is defined as:

$$ TA(\hat{\mathcal{F}}, \mathcal{F}) = \frac{\sum_{k=1}^{M} \sum_{j=1}^{T_{f^k}} \begin{cases} 1, & \text{if } f^k_j = \hat{f}^k_j \\ 0, & \text{otherwise} \end{cases}}{\sum_{k=1}^{M} T_{f^k}} $$

(5.13)

where $f^k_j$ is the $j$-th symbol of the $k$-th formula, $\hat{f}^k_j$ the $j$-th symbol of the $k$-th gold truth formula, and $T_{f^k}$ the length of the $k$-th formula. The value of $FA$ is a real number in the range $[0, 1]$. The higher the value of $TA$, the better the network is performing.

**5.4.3 Datasets**

In order to train and evaluate the model we needed a dataset, which is a significant sample of the problem space according to the research questions articulated in Section 5.4.1. This means that the dataset should present different syntactic structures and that the examples used for testing should be different from the one used for the training, in order to verify that the networks are actually able to generalize over such syntactic structures. Moreover, unknown words must be present in the training and test data, so that the ability of the model to handle them, leveraging the remaining of the sentence as a context, can be proven. Such examples have been generated specializing the conceptual data generation process described in Section 4 in
order to have training examples suitable for both the tasks of tagging and transduction.

The first step in the implementation of the data generation process, was to design a proper grammar to generate the sentence templates. We started by verbalizing with ACE (see [28]) a set of OWL class definitions in order to have a first seed of definition-like sentences, as typically found in encyclopedias. We extended this seed by manually adding variations of every verbalization and other equivalent structures. So, for the sentence “all the dogs are mammals”, we added “every dog is a mammal”, “dogs are mammals”, “any dog is also a mammal” and so on. Or, for “a bass guitar has at least 4 strings”, we added “bass guitars have more than 3 strings”, “A bass guitar does not have less than 4 strings” and so on. Finally, we built a grammar capable to generate all such sentences, with placeholders instead of surface realizations of concepts, roles, and ordinal numbers used in cardinality restriction clauses.

Having designed this grammar, in order to produce examples for both the tasks of tagging and transduction, we had to slightly modify the data generation pipeline described in Section 4.2. Indeed, since we are training two networks for two different tasks, we need two sets of examples for training and evaluation, one for each network (and task). The actual data generation process is depicted in Figure 5.4 and will be described in detail below.

The first set of examples is the one for the tagging task. Each example is made of an input sentence and an output sequence of the same length, presenting at each position the label of the placeholder of the corresponding word within the sentence, or the empty label $w$, if the corresponding word is not intended to fill any placeholder. This output sequence will be henceforth referred to as the tag sequence. As an example, the sentence in (4.1) at page 70 will be associated to the following tag sequence:

$$w\ C0\ w\ w\ C1\ C1\ w\ R0\ w\ w\ \text{NUM}0\ C2.$$  \hspace{1cm} (5.14)
The second set of examples is the one for the transduction task. Each example is made of an input sentence and of an output formula template. As an example, the sentence in (4.1) will be associated to the formula template in (4.2).

Note that to represent the ordinal numbers to be used in cardinality restrictions, we used the symbol \texttt{NUM} followed by a number representing the cardinal zero-based position of the occurrence of a number in the sentence. So the first number will be \texttt{NUM0}, the second one \texttt{NUM1}, and so on.

Downstream the generation of the examples, data must be encoded in order to be fed into the proper neural network. The encoding process is straightforward. We collect all the words appearing in all the sentences in a vocabulary $V$, comprising the special symbols $<$EOS$>$, $<$UNK$>$, and all the
number replacing symbols. Then, given a sentence, we just replace each word with the index of the word itself within the vocabulary. A similar procedure is followed for the tags, collecting all of them in a tag vocabulary \( T \), and the formula terms, collected in the vocabulary \( F \). Note that the vocabulary \( F \) comprises all the logical symbols of the target logic language together with all the placeholders used for concepts, roles and ordinal numbers.

From our grammar, we generated more than 123 millions of different sentence templates, each of which has been associated to its equivalent formula template. We obtained more than 260 thousands different formula templates in total. We filled placeholders for concepts and roles, leaving those used for ordinal numbers as they are—and considering them as the output of a minimal preprocessing phase, as already said at the end of Section 4.2.

<table>
<thead>
<tr>
<th>adjective</th>
<th>noun#1</th>
<th>noun#2</th>
<th>of</th>
<th>adjective</th>
<th>noun#1</th>
<th>noun#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>magnificent</td>
<td>sword</td>
<td>sharpener</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>sword</td>
<td>sharpener</td>
<td>of</td>
<td>-</td>
<td>-</td>
<td>steel</td>
</tr>
<tr>
<td>magnificent</td>
<td>sword</td>
<td>sharpener</td>
<td>of</td>
<td>-</td>
<td>-</td>
<td>steel</td>
</tr>
<tr>
<td>sword</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>mountain</td>
<td>steel</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>sharpener</td>
<td>of</td>
<td>shining</td>
<td>mountain</td>
<td>steel</td>
</tr>
</tbody>
</table>

Table 5.1: Patterns for Concept placeholders filling.

placeholders are filled with verbs randomly selected from a list of 882 among the most common English verbs. Concept placeholders are filled with a two-steps operation. First a pattern of different word classes is randomly selected from a list of 66 hand-crafted patterns. These patterns are used to generate concept names as multi-words. The word classes that are used are one for adjectives and two for nouns. Moreover, the preposition “of” is used as a word class containing only one word. Such patterns can be long up to 7 items. Once a pattern as been selected, the actual words are selected, one for each word class. Adjectives are randomly selected from a list of 1522.

The first noun list is made of 2425 words, while the second comprises 192 nouns\textsuperscript{2}. Some of them, are reported in Table 5.1.

In order to simulate the presence of unknown words, we have filled the placeholders in the sentence templates replacing some of the actual words with the \texttt{<UNK>} special symbol. More in detail, when drawing a word from a word class, we randomly mark it with \texttt{<UNK>}. Words used to fill concept placeholders have 60\% of probability to be marked as unknown, while words filling role placeholders have 20\% of probability—this inequality comes from the empirical observation that, even across different domains, verbs are less variant than nouns and adjectives. Once all the placeholders have been filled, we re-scan the whole sentence, ensuring that the global percentage of unknown words used for placeholder filling is between 20\% and 40\%. These probability values comes from our overestimation of the actual probability of having an unknown word for each class word and in a whole sentence.

All the resulting formulæ have a left hand side class description and a right hand side class description, not necessarily atomic. Relations between such class descriptions can be subsumptions, disjunctions or equivalences. Each side is made of one or two atomic concepts or roles. Both atomic concepts or role can be negated. In case of two items, these can be in a conjunction or disjunction relation. Each role can be qualified with up to two cardinality restrictions. This limitation has been arbitrarily set, since we found that is complex from a grammatical standpoint to express more than two cardinality restrictions in a natural language class description. If two, these restrictions can be in a conjunction or disjunction relation.

\textsuperscript{2}The list of words for each word class have been originally taken from a website (now offline) used to generate random Java class names.
5.4. EVALUATION

5.4.4 Experimental Settings and Results

We evaluated the two networks under several settings. The first one refers to the size of the training and testing examples used. We run the data generation process several times, generating training sets of different sizes, namely 1000, 2000, 3000 and 4000 examples, for both the tasks. The networks are then evaluated using a test set of 2 millions of sentences. Being both the sentence templates and the actual words used to fill the placeholders randomly selected, the probability that many of the sentences from the test set have not been seen in the training set is extremely high. This ensures that the network does not just memorize the training examples, but actually learns how to leverage the knowledge acquired during the training to process different sentences sampled from the same universe of language. Having different sizes for the training set, we could quantitatively measure how much the network is capable to leverage such examples to generalize across the whole problem space. Similarly, having unknown words in the training and test examples, we can measure how much the network is capable to handle them, leveraging the remaining of the sentence.

The value of the hyperparameters of both the networks have been set empirically, according to similar experiences in the literature and to some preliminary experiments. All these values are reported in Table 5.2. Both the networks have been trained using the AdaDelta (see [94]) training algorithm, a variation of Stochastic Gradient Descent that adapts the update value dynamically, so that the less frequently updated parameters are subject to a deeper update. We used batches of 50 examples for both the networks. All the training settings, which were also empirically defined, are reported in Table 5.3. All the experiments were run on a workstation endowed with a NVidia Tesla K40 GPU card, taking a time laps of about 2.5 hours for the tagging network and 4.5 hours for the transduction network.
Table 5.2: Network hyperparameters.

<table>
<thead>
<tr>
<th>Networks hyperparams.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td># words</td>
<td>∼5000</td>
</tr>
<tr>
<td># tags</td>
<td>11</td>
</tr>
<tr>
<td># terms</td>
<td>21</td>
</tr>
<tr>
<td>word window</td>
<td>5</td>
</tr>
<tr>
<td>dim. embedding</td>
<td>100</td>
</tr>
<tr>
<td>dim. hidden (tag.)</td>
<td>200</td>
</tr>
<tr>
<td>dim. enc/dec (tra.)</td>
<td>1000</td>
</tr>
</tbody>
</table>

Table 5.3: Training parameters.

<table>
<thead>
<tr>
<th>Training settings</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>training steps</td>
<td>10000</td>
</tr>
<tr>
<td>batch size</td>
<td>50</td>
</tr>
<tr>
<td>learning algo.</td>
<td>AdaDelta</td>
</tr>
<tr>
<td>learning rate</td>
<td>2.0</td>
</tr>
<tr>
<td>ρ</td>
<td>0.95</td>
</tr>
<tr>
<td>ε</td>
<td>10⁻⁶</td>
</tr>
<tr>
<td>GPU Card</td>
<td>Tesla K40</td>
</tr>
<tr>
<td>time (tag.)</td>
<td>∼2.5h</td>
</tr>
<tr>
<td>time (tra.)</td>
<td>∼4.5h</td>
</tr>
</tbody>
</table>

Table 5.4 reports the evaluation results for the different training sets in terms of number of correct formulæ (CF), Average Per-Formula Accuracy (FA), Average Edit Distance (ED), and Average Per-Token Accuracy (TA) on the 2 millions test examples. Such values have been computed comparing the expected formula and the one that can be rebuilt combining the output of the two different networks. More in detail, the words labelled with a certain tag by the tagging network are used to replace the corresponding placeholder in the formula template produced by the transduction network. As an example, let us consider the sentence “a bee is an insect”. If the tagging network produces the tag sequence \( w \ C_0 \ w \ w \ C_1 \) and the transduction network produces the formula template \( C_0 \sqsubseteq C_1 \) and the transduction network produces the formula template \( C_0 \sqsubseteq C_1 \), we can rebuild the final formula as \( \text{bee} \sqsubseteq \text{insect} \).

Table 5.4: Accuracy on the test set

<table>
<thead>
<tr>
<th>training set size</th>
<th>CF</th>
<th>FA</th>
<th>ED</th>
<th>TA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>10</td>
<td>0.5e-5</td>
<td>2.67</td>
<td>0.90</td>
</tr>
<tr>
<td>2000</td>
<td>161</td>
<td>8.05e-5</td>
<td>1.34</td>
<td>0.95</td>
</tr>
<tr>
<td>3000</td>
<td>60</td>
<td>3.00e-5</td>
<td>1.22</td>
<td>0.96</td>
</tr>
<tr>
<td>4000</td>
<td>22</td>
<td>1.10e-5</td>
<td>1.07</td>
<td>0.97</td>
</tr>
</tbody>
</table>

The model achieves good results in terms of Average Edit Distance and Average Per-Token Accuracy. According to these metrics, the performance of the model increase monotonically with the size of the training set, showing a
significant boost passing from 1000 to 2000 training examples. Recalling that the Edit Distance is the number of edits that a human operator must perform to fix the predicted formula and make it equal to the expected one, we noticed that the predicted formulae are on average very close to the actual ones. The fact that we obtained 0.90 of Average Per-Token Accuracy with just 1000 training examples, gives us an evidence that our system is able to learn the syntactic structures typical of descriptive language and to generalize over them (RQ1). Moreover, having between 20% and 40% of unknown content words, gives us evidence that the model is tolerant to unseen words (RQ2).

Though, observing the performance of the model in term of corrected formulæ (and, consequently, in terms of Average Per-Formula Accuracy), we observe that the model is unable to directly produce a significant number of correct formulæ. Moreover, we can observe how the Average Per-Formula Accuracy is not monotonic with respect to the number of examples in the training set but it reaches its top value when the network is trained with 2000 training examples, while decreases as we extend the training set. Given the good performance in terms of Average Per-Token accuracy and Average Edit-Distance, the low scores in terms of Average Per-Formula Accuracy indicate that the model tend to produce few wrong predictions but consistently across the vast majority of the test set—recall that, in a hypothetical test set of 100 sentence mapped to formulæ of 100 terms each, if the model predicts one wrong term per formula, we would have an Average Per-Token Accuracy of 0.99, an Average Edit-Distance of 1.00 but an Average Per-Formula Accuracy of 0.00.

The behavior of the model is critically discussed in the next section.
5.5 Discussion

In this first stage of our work we exploited the very same intuition underlying many pattern-based approaches for Ontology Learning, namely the idea that syntax matters, pushing it into a totally different direction, with a learn by examples approach. We trained a model that learns to model encyclopaedic language and its syntactic structures, it learns how to parse their occurrences in the text and how to translate them into corresponding logical constructs.

More in details, we presented an approach for Ontology Learning where two Recurrent Neural Network-based models are trained in a end-to-end fashion to translate definitions of concepts in plain english, compliant to the definition of descriptive sentence provided in Chapter 3, into $\mathcal{ALCQ}$ formulæ. A GRU based Recurrent Encoder-Decoder is used to transduce a definition into its corresponding formula template, while a GRU based Recurrent Neural Network tags each word with a label representing the role of the word itself within such formula template. Roughly speaking, our tagging model can be seen as a POS tagger, and our transduction model can be seen as a syntactic parser. Both of them extremely specialized w.r.t. the type of language of interest. Our models store in their parameters the embedding of each word in the vocabulary (comprising the $\texttt{<UNK>}$ and $\texttt{<EOS>}$ symbols), how to deal with function words, and many other syntactic constraints. Being our model trained in a end-to-end fashion, this knowledge—namely the features learnt by the model—remains in the subsymbolic form and is not made explicit.

We trained and tested our approach on a newly created dataset of sentence-formula template pairs, sampling from more than 123M distinct sentence templates and more than 260K distinct formula templates. Our system achieved 0.90 of Average Per-Token Accuracy and 2.67 of Average Edit Distance starting from only 1000 training examples. Our experiments show that
statistically learning an ideal set of syntactic transformation rules is feasible. Furthermore, our contribution presents several advantages over state-of-the-art pattern-based approaches: (i) it does not require to manually define pattern rules for each possible linguistic natural language variation to be covered, something practically unfeasible; (ii) our model is trained in an end-to-end fashion, from raw text to OWL formulæ, without relying on any NLP tool and requiring no feature engineering cost for the input representation; finally, (iii) being our approach purely syntactic, it does not need any domain-specific training: content words of our test set are selected randomly, showing that our model does not rely only on their lexical meaning but also on their syntactical features.

We want to remark that all the sentence templates for both the training set and the test set, were randomly generated and being the test examples way more than the training examples, we are confident that large part, or even possibly none, of the test examples have not been seen during the training phase. This means that there is no possibility for the networks to just memorize the examples. Another important characteristic that we want to stress here, is that, even if the number of possible tag sequences and formula templates is limited, the transduction network does not just classifies examples, learning to associate a sentence to a formula template. The transduction sentences learn to generate a new sequences of symbols, namely a formula template.

Despite being our dataset generated starting from sentences in ACE, our system can deal with language variability that goes well beyond controlled English. To confirm this, we generated 4000 random sentence templates and, from them, a set of sentences, filling the various placeholders with a simplified vocabulary compliant with the Attempto Parser Engine (APE)\(^3\). The APE engine could parse only 13 sentences. A qualitative analysis through the

\(^3\)https://github.com/Attempto/APE (last accessed on January 31, 2018).
sentences not or incorrectly parsed by the APE service gave us an idea of some of the linguistic phenomena that our system can handle beyond the controlled language. We can roughly split them in two groups:

1. function words that are not parsed by the controlled language but that are actually used in natural language, such as:

   (a) “anything” and “any” acting as universal quantifier;
   (b) “at least”, “no more than” or “exactly” for cardinality restrictions;
   (c) “but” as a conjunction in cardinality restrictions, e.g. “less than 3 but more than 10”;
   (d) “also”, in the right hand side of an implication, e.g. “if something is a mammal, it is also an animal”;
   (e) “everything that is” as a quantifier;
   (f) the use of “some” to indicate an intersection, e.g. “some birds are flightless”;
   (g) “do not” as a negation of a role.

2. constructs that are not parsed by the controlled language but are actually used in natural language, such as:

   (a) ellipsis of the demonstrative pronoun “that” in conjunction or disjunctions, e.g. “everything that has a tail and (that) is a dog, also chases cats”;
   (b) ellipsis of the demonstrative pronoun, role and range concept in conjunction or disjunction of cardinality restrictions, e.g. “a bass is an instruments that has at least 4 (strings) or (that has) at most 6 strings”;
   (c) ellipsis of the adverb in cardinality restriction, instead of “exactly”, as in “a bee is an insect that has (exactly) 6 legs”;

101
Even if converting arbitrary natural language text to an arbitrarily expressive DL language is still an ambitious, out-of-reach goal, these results give evidence of the capabilities of our approach in translating definition-like sentences to (complex) DL axioms, while showing good syntactic and lexical generalization capabilities and a reduced annotation effort of 1000 sentences.

Though, the approach presented so far in this Chapter suffers some severe limitations. The first one dwells in the bijective nature of tagging: each word can be tagged only once and with only one label. So, consider a sentence like “a property is a quality of an event or object” and assume that we want to translate it into the formula $\text{property} \sqsubseteq \text{quality\_of\_event} \sqcup \text{quality\_of\_object}$. The corresponding formula template generated by the transduction process would be $C_0 \sqsubseteq C_1 \sqcup C_2$. The tagging network should tag twice the words “quality” and “of”, first with $C_1$ and then with $C_2$, which is structurally impossible using the architecture described in this Chapter. The second limitation is given by the fact that the tags representing the placeholders for concepts, roles, and numbers are indexed with a number. Supposing that in the training examples the maximum amount of concepts in a single sentence is 2, the network will be capable to use the tags $C_0$ and $C_1$. At run time, if a sentence with 3 involved concepts is fed into the network, it will not be able to emit a $C_2$ tag for the third concept, since such symbol is not in the output vocabulary of the network. As a consequence, it is not possible to add more complex (i.e. with a larger amount of tags per sentence) examples, without changing the output dimension of the network, since new tags will have to be added to the output vocabulary as well.

The last limitation is the fact that we are currently exploiting two different architectures since we are decoupling the translation of a sentence into a formula into two separate tasks, doubling many efforts: tuning of hy-
perparameters, training time, annotation of training examples, and so on. Moreover, being the two different networks trained separately, they cannot share any knowledge on the problem space. We hypothesize that this is one of the main cause of the misalignment between the metrics presented in Table 5.4, where the local prediction is way more accurate than the global one and the amount of predicted formulæ equal to the expected ones is reduced, even if the difference between each expected formula and the corresponding predicted one is small—as evidenced by the extremely reduced value of the Average Edit Distance and the high value of the Average Per-Token Accuracy.

Following the recent advances in Deep Learning architectures, we were able to overcome all such issues, exploiting a single Neural Network capable to perform syntactic transformations on a definition in order to turn it into a DL formula. Such further stage of our work will be described in Chapter 6.
Chapter 6

Ontology Learning as Neural Machine Translation

In Chapter 5 we presented our first attempt to exploit Neural Networks trained in a end-to-end fashion to translate plain english concept definitions, into corresponding \texttt{ALCQ} Description Logics formulæ. Such approach uses \textit{two} different neural networks, one to transduce a sentence into a formula template with placeholders for concepts, roles, and ordinal numbers, and another one to tag each word in the sentence with the proper placeholder. By doing this, it is possible to rebuild the final formula just filling the placeholders in the formula template with the words within the sentence that have been properly tagged. With an example, the sentence “\textit{a bee is an insect}” will be transduced into a formula template $C_0 \sqsubseteq C_1$ and, in parallel, tagged with “$a/w\ bee/C_0\ is/w\ an/w\ insect/C_1$” so that the final formula can be rebuilt as $\texttt{bee} \sqsubseteq \texttt{insect}$. Recall from Chapter 5 that $w$ is the \texttt{empty} tag, denoting a word which is not a filler of any placeholder.

Exploiting \textit{two} neural networks means dealing with two training processes, two training sets to be created and maintained, two sets of hyperparameters to be defined and tuned, and so on. Moreover, such an approach is limited in terms of the syntactic structures that the model can handle, since each word can be tagged only once and the maximum amount of concepts and
roles is constrained at the training phase, as discussed in Section 5.5. Such severe limitations make this approach still limited from a practical point of view, even if it has the notable merit of having shown how Recurrent Neural Networks-based architectures are capable to deal with the syntactic structures typical of descriptive language.

In this Chapter, we present a further model that we investigated which uses just one network to overcome the intrinsic limitation of the solution described in Chapter 5. The model we present in this Chapter falls into the Sequence-to-Sequence family, like the one presented in Chapter 5 for the transduction task. Moreover, both the models are particular flavors of the Recurrent Encoder-Decoder configuration. The main difference is in the fact that the model we are presenting does not need to rely on any tagging system, since it can copy words from the input sentence into the output formula exploiting a so-called pointing mechanism (see [85]).

In Section 6.1 we describe the main underlying intuition from a conceptual point of view. Then, in Section 6.2 we provide the full mathematical description of the Neural Network architecture that has been used. Finally, in Section 6.3 we will present the evaluation phase in terms of what we investigated (6.3.1), the metrics that we used to quantitatively measure the model performances (6.3.2), the datasets that we used to train and evaluate the model (6.3.3), and the results of the different experiments that we run (6.3.4). Differently from the model investigated in Chapter 5, the one presented in this chapter has no hard constraints on the structure of the input sentence and is therefore more promising. So it was considered worth to be evaluated in a deeper and more structured manner. A final discussion on the approach is presented in Section 6.4.
6.1 The Intuition

As stated at the beginning of Chapter 3, we intend to translate a sentence into a formula by means of syntactic transformations. This means that the resulting formula can be seen as a sequence of symbols, some of them drawn from the list of logical symbols of the target logical language, and the other being extralogical symbols occurring as lemmas in the input sentence. Recall the example used in Section 5.1, namely:

\[(6.1) \quad A \text{ bee is an insect that produces honey.}\]

that we want to be translated into the formula:

\[(6.2) \quad \text{bee } \sqsubseteq \text{ insect } \land \exists \text{ produces honey,}\]

Differently from the approach presented in Chapter 5, here we rely on two operations. The first one is called \(\text{emit}(\cdot)\), accepts a logical symbol as an input, and returns the symbol itself. The second one, called \(\text{copy}(\cdot)\), accepts an integer as input and returns the corresponding word from within the sentence. Using these operations, we can turn the sentence in (6.1) into the formula (6.2), as shown in Figure 6.1, where the \# symbol is used to denote that the following number is the position of a certain word within the input sentence. At this stage of our investigation, we exploit so called Pointer Networks (see [85]) in the flavor proposed in [36] for a Neural Machine Translation task, where the ability of copying symbols from the input sequence, has been used to handle rare or unknown words, named entities, and so on. We exploited the same architecture to build a single Recurrent-Encoder Architecture trained to read a sentence and generate a formula using the \(\text{emit}(\cdot)\) and \(\text{copy}(\cdot)\) functions defined above. In particular, at each timestep of the output formula, the decoder will be in charge of deciding if the current output symbol will be a logical one, selected from a certain vocabulary, or an
extralogical one, to be copied from the input sentence. In both cases, the decoder will be also in charge of deciding which logical symbol to emit or which extralogical symbol to copy from the sentence. Note that the output vocabulary is reduced only to the logical symbols of the target Description Logic language. For this reason, we use the term shortlist to refer to such vocabulary and describe this particular setting with the expression quasi-zero vocabulary.

### 6.2 Architecture Description

The network architecture used at this stage of our work falls within the class of Recurrent Encoder-Decoder models, like the one presented in Section 5.3 for the Transduction task. Though, some substantial differences can be noticed between the two models. At a glance, both of them work the same until the encoding phase is completed: word indexes within the vocabulary are projected into their corresponding word vectors and subsequently fed into a recurrent layer that produces a set of encoder states. At this point,
the Transduction model exploits the latest encoder state as the constant input of the decoding layer. Each decoding state produced by the decoder, is subsequently fed into a softmax layer. This latest layer provides a probability estimation over all the possible terms to be used in the output formula template.

The main differences between the present model and the one presented in Chapter 5 are in the interaction between the encoder and the decoder and in the output layer. More in detail, at each timestep of the output formula, the output layer is in charge of a complex decision: indeed, it has to decide if the current output symbol is a logical symbol from the shortlist or it is an extralogical symbol to be copied from the sentence. In both cases, it also has to decide which is the proper symbol, within the possible choices that depend on the source. Such complex decisions are made possible by the several components of the output layer and by the so called attention module. The latter, is in charge of modulating the interaction between the encoder and the decoder, providing the latter with a different context vector at each timestep. All such components will be described in detail in the remainder of this Section, with the help of Figure 6.2, which illustrates the decoding process for the $j$-th timestep of the formula.

6.2.1 The Decoder

Generally speaking, the decoder is in charge of producing a set of vectors, called decoder states, to be consumed, one at each timestep, by the output layer. At the $j$-th timestep of the output formula, the decoder state $d_j$ can be written in terms of the decoder cell function:

$$d_j = g(x_j^{dec}, d_{j-1}),$$

(6.3)
where the function \( g(\cdot, \cdot) \) is the proper decoder cell function, and the vector \( x^{dec}_j \) is the **decoder input**, namely the external input of the cell function. For the model in Section 5.3, at each timestep, the decoder input is given by the context vector \( c \), namely the last encoder state. In this case, the decoder input is slightly more complex. It is given by the following vector concatenation:

\[
x^{dec}_j = d_{j-1} \oplus c_j \oplus \tilde{y}_{j-1}.
\] (6.4)

where \( d_{j-1} \) is the previous decoder state, \( c_j \) the context vector for the current timestep, and \( \tilde{y}_{j-1} \) is an approximation of the previous final output, called...
the feedback vector. The way the latest two vectors are computed, will be discussed later.

In particular, note that the context vector \( c_j \) depends on the timestep. This differs from what we presented in Section 5.3. At each timestep, such context vector \( c_j \) represents the way how the whole input sentence contributes to the current decision about the output symbol. This context vector is produced by the attention module, described in the following section.

### 6.2.2 The Attention Module.

The attention module is in charge of producing, at each decoding timestep, a context vector that indicates how the whole input sentence has to be taken into account in the production of the current output symbol. Moreover, it produces a weights vector which weights the importance of each input word for this decision.

Let us consider a set of \( H \) vectors, say \( \mathcal{H} = h_1, ..., h_H \). These vectors, called attention states, are defined in the same vector space, say \( \mathbb{R}^h \). Given another vector \( q \), called query vector, defined in \( \mathbb{R}^q \), we can learn an alignment function, that computes some alignment score between \( q \) and each of the attention states, as:

\[
    a_i = \text{align}(q, h_i). \tag{6.5}
\]

Let \( a = [a_1, ..., a_H] \) be the collection of all such scores into a single vector. Applying a \( \text{softmax}(\cdot) \) function to the vector, we constrain all its elements to fall in the \([0, 1]\) range and to sum up to 1. The softmax function applied to the \( i \)-th alignment score can be written as:

\[
    \alpha_i = \frac{\exp(a_i)}{\sum_{k=1}^{H} \exp(a_k)}. \tag{6.6}
\]

We call such normalized score \( \alpha_i \) the attention weight for the \( i \)-th attention state with respect to the query vector. We can exploit the \( \text{align}(\cdot, \cdot) \) function
and the \( \text{softmax}(\cdot) \) normalization to define a function \( \text{weights}(\cdot, \cdot) \). This function, given a query vector \( \mathbf{q} \) and a set of attention states \( \mathcal{H} \), returns a vector \( \mathbf{w} \) of size \( H \), the number of attention states. This vector is called \textit{weights vector} and contains the attention weights for each of the attention states with respect to the query vector. We write:

\[
\mathbf{w} = \text{weights}(\mathbf{q}, \mathcal{H}) = [\alpha_1, \ldots, \alpha_H]. \tag{6.7}
\]

In other words, the weights vector is obtained collecting all the alignment scores between the query vector and each of the attention states into a single vector which is subsequently normalized through the application of the softmax function.

We can exploit the weights collected in the weights vector to compute a weighted sum of the attention states. We can define a function \( \text{context}(\cdot, \cdot) \) that accepts our query vector \( \mathbf{q} \) and our set of attention states \( \mathcal{H} \) as inputs, and returns a single \textit{context vector} of the same size of each attention state, namely \( h \). We can write:

\[
\mathbf{c} = \text{context}(\mathbf{q}, \mathcal{H}) = \sum_{i=1}^{H} \alpha_i \mathbf{h}_i. \tag{6.8}
\]

Note that the higher the value of a weight, the more \textit{important} is the corresponding attention state for the network, since it gives a stronger contribution to the context vector.

The attention mechanism described so far, has been first presented in \cite{7} in order to train a network to implicitly align component of sentences from different languages in a Neural Machine Translation task. The main advantage of this mechanism is that it provides a way to \textit{focus} on some particular portion of the network internal state, represented by the set of attention states. In our work, we exploited this mechanism in order to compute, at each timesteps of the output formula, both the context vector and the weights
CHAPTER 6. TRANSLATION 6.2. ARCHITECTURE DESCRIPTION

vector. The former is used to feed the recurrent decoder in order to generate the current decoder state, while the latter will be used by the output layer to decide which word from within the sentence should be copied as an extralogical symbol in the formula. In the particular implementation we used, at the $j$-th timestep, the previous decoder state, namely $d_{j-1}$ and the encoder states $\mathcal{H} = h_i, ..., h_{T_x}$ are used as the query vector to generate the context vector for the current timestep, so that we can write:

$$c_j = context(d_{j-1}, \mathcal{H}) = \sum_{i=1}^{T_x} \alpha_i h_i. \quad (6.9)$$

The context vector represents the contribution of the whole input sentence to the decoding decision at the current timestep.

Concretely, the $\text{align}(\cdot, \cdot)$ function, we used the the one presented in [86], where the alignment score is given by the:

$$a_i = v_a^T \tanh(W_a h_i + U_a q), \quad (6.10)$$

where $\tanh(\cdot)$ is the hyperbolic tangent function. This equation fully defines the set of parameters of the attention module to be learnt, namely $v_a$, $W_a$, and $U_a$. The dimension $v$ of the vector $v_a$, is an hyperparameter to be set at design time and we call it the attention inner size. This hyperparameter constrains the other parameters of the attention mechanism. Being indeed in a dot product with $v_a^T$, the result of the $tanh(\cdot)$ must be a vector of the same dimension. This impose the constraints on all the parameter of the alignment function, so that $W_a \in \mathbb{R}^{v \times h}$ and $U_a \in \mathbb{R}^{v \times d}$.

Before describing the output layer of the network, we want to remark that, in our scenario, the weights vector $w_j$ is the only vector whose dimension is not fixed. The weights vector has a number of components which is equal to $T_x$, the number of words in the current input sentence. Since different
input sentences can have different lengths, the size of the weights vector can change from sentence to sentence. This is what will allow our network to copy extralogical words from the sentence into the formula, as we will explain later, selecting the word corresponding to the higher weight.

6.2.3 The Output Layer

At each timestep, the output layer has to decide if the current symbol in the formula is a logical symbol from the shortlist or an extralogical symbol, copied from the input sentence. In both cases, it also has to decide which is the correct symbol. Recall that the shortlist is the list of all the logical symbol of the \( \mathcal{ALCQ} \) Description Logics language. This complex decision mechanism is realized through the interaction of different modules that will be described below.

The switch network

The switch network is the module of the decoder that decides if the current symbol will be a logical one or an extralogical one. Note that the switch network will not decide which will be the next symbol of the formula, but only if it will be a logical or an extralogical one. We can model such module as a function \( z(\cdot, \cdot) \) accepting as inputs the context vector and the decoder state for the current timestep and returning as output a real number \( z_j \) between 0 and 1:

\[
z_j = z(c_j, d_j).
\]

We can consider the value of \( z_j \) as an estimation of the probability of the current symbol to come from the shortlist or not, given: i) the input sentence, summarized by the current context vector \( c_j \), and ii) the evolution of the network output so far, represented by the current decoder state \( d_j \). In other words, the closer the value of \( z_j \) is to 1, the more likely the symbol at
the current timestep will be a logical one, the closer to 0, the more likely it will be a word copied from the sentence. We model the function \( z(\cdot, \cdot) \) using a single output perceptron, as in:

\[
    z_j = \sigma(W_z d_j \oplus c_j + b_z),
\]

(6.12)

where the sigmoid function \( \sigma(\cdot) \) ensures the value of \( z_j \) to fall in the range [0, 1]. In this way, \( z_j \) can be used as an estimation of the probability of the symbol at the current timestep to come from the shortlist. Note that to have a coherent notation, we used the notation of matrix and vector for \( W_z \) and \( b_z \) respectively. Nonetheless, since \( z_j \) is a scalar, then \( b_z \) is a vector of one component (i.e. a scalar) and \( W_z \) is a matrix in \( \mathbb{R}^{1 \times (d+h)} \) (i.e. a row vector). Moreover, the concatenation between the decoder activation and the context vector should be read as a column-vector of size \( (d+h) \). The learnt parameter of this component are \( W_z \) and \( b_z \).

The shortlist softmax

The shortlist \( L \) comprises all the logical symbols of the \( \mathcal{ALCQ} \) Description Logics language. The decision about which logical symbol has to be chosen from the shortlist is up to the shortlist softmax. This module is a function \( u(\cdot) \) that accepts the decoder state as its input and returns a vector of real numbers:

\[
    u_j = u(d_j).
\]

(6.13)

The goal of this vector is to represent the estimation of a probability distribution over the logical symbols. This means that the size of \( u_j \) will be the same of the shortlist \( L \), that the value of all its elements will fall in the range [0, 1] and, finally, that they will all sum up to 1. If the value of \( z_j \) in (6.12) is close to one, the symbol whose position within the shortlist corresponds to the position of higher value within \( u_j \) will be chosen.
This component can be implemented using a single layer perceptron that projects the decoder output onto a vector in $\mathbb{R}^{|L|}$, where $|L|$ is the size of the shortlist, and by exploiting a $\text{softmax}(\cdot)$ as the activation function, so that the output values are properly normalized. We can write:

$$u_j = \text{softmax}(W_u d_j + b_u),$$

(6.14)

with $W_u \in \mathbb{R}^{|L| \times d}$ and $b_u \in \mathbb{R}^{|L|}$. The output vector of the shortlist softmax $u_j$ can be seen as an estimation of the probability over the set of symbols from the shortlist to be emitted as the current symbol of the formula. The learnt parameter of this component are $W_u$ and $b_u$.

The location softmax.

If the value of $z_j$ is close to zero, the network will emit an extralogical symbol by copying it from the input sentence. Recalling what we said before, the weight vector $w_j$ produced by the attention module has the same size of the input sentence with real values in $[0, 1]$ and summing up to 1. Hence, we can consider this vector as a probability distribution over the words in the input sentence. If the switch network decides that the current symbol in the formula has to be copied from the input sentence, the one corresponding to the higher value in the weight vector will be chosen. As already remarked, the weight vector is a vector of variable length, meaning that the length is not fixed but depends on the length of the input sentence. Thus, the probability distribution is on the actual words in the sentence and there is no need to constrain the sentence length to a fixed value by padding or truncating.

The final output.

At each timestep, we can combine together the output of the switch network, the shortlist softmax, and the attention module in order to have a single
output that models a single probability distribution over all the possible symbols that can appear in the formula. This set comes from the union of all logical symbols from the shortlist $L$ and the words from the input sentence.

Recalling that $z_j$ is the probability for the current symbol to be a logical symbol, $1 - z_j$ is the probability for it to be an extralogical one, $u_j$ is a probability distribution over the symbols in the shortlist, and $w_j$ the probability distribution over the words in the sentence, we can combine them into the output vector, given by the following expression:

$$y_j = z_j \cdot u_j \oplus (1 - z_j) \cdot w_j,$$

where the $\cdot$ operator is the multiplication between a scalar and each component of a vector.

Elements in $u_j$ and $w_j$ separately, are in the range $[0, 1]$, and they sum up to 1. Multiply them by $z_j$ and $1 - z_j$ respectively, with $z_j$ in the range $[0, 1]$, makes all their elements sum up to $z_j$ and $1 - z_j$ respectively. As a consequence, all the elements of the vector $y_j$, resulting from their concatenation, are in the range $[0, 1]$ and sum up to 1. For this reason and being the size of $y_j$ equal to the length of the shortlist plus the length of the sentence, it can be seen as a probability distribution over all the possible logical and extralogical symbols that can end up in the output formula for the current input sentence. If the maximum value of $y_j$ occurs in a position which is between 1 and the size of $u_j$ (which is, remember, $|L|$) the corresponding logical symbol will be emitted in the output formula, otherwise, the word at position $\text{argmax}(y_j) - |L|$ will be copied.

The final decision of the network is fed back to the decoder through the feedback vector $\tilde{y}_j$, already introduced in (6.4). The feedback vector is a fixed size approximation of the actual output vector $y_j$. We need to exploit such approximation since the size of $y_j$ changes according to the size of $w_j$, that is equal to the length of the input sentence. This, would cause the
decoder input vector as defined in (6.4) to have the same variable size. On the equations of the GRU cell (see (2.53) and (2.54)) used in the decoder require the input vector to be statically determined, otherwise the dimension of some of the model parameters would be undefined. Thus, we set a fixed dimension and truncate or pad with zeros in order to fit it. We call this dimension feedback size and it is an hyperparameter of the model. We want to stress that the feedback size does not have any impact on the capability of the model to handle sentences of different length and to generate, at each timestep, an output vector of the same size of the input sentence.

6.3 Evaluation

In this section we introduce our evaluation for the model proposed in this chapter. Since the model under investigation is intended to overcome the limitations of the one presented in Chapter 5 and, we believe, it provides a promising contribution to Ontology Learning, we performed an evaluation that is deeper and wider of the one carried out for the previous model. First, we articulate the evaluation extending the research questions in 6.3.1. Then, we introduce additional quantitative metrics used to measure the performance of the model with respect to the evaluation objectives in 6.3.2. We describe how we generate the datasets used to train and evaluate the network in 6.3.3. Finally, in 6.3.4 we will present the experimental settings and results.

6.3.1 Research Questions

This evaluation aims at checking the ability to generalize over the grammar and the lexicon, similarly to what was evaluated in Section 5.4. Furthermore, we assessed the ability of the model to improve by extending the training set.

The reason why we want to evaluate the generalization capabilities of the
model has already been explained in 5.4.1. This evaluation has been carried out by answering the two research questions:

RQ1. To what degree is the network capable to generalize over the syntactic structures of descriptive language?

RQ2. To what degree is the network capable to tolerate words that have not been seen during the training phase?

Recall from 1.3.2, one of the long term goals of our work is to devise a system that can be effectively bootstrapped with grammar generated data, and is capable to increase its performance when trained again, after that new examples have been added to the training set. The operation of incrementally extending the training set with new examples must be carried out with some caution. Indeed, it is not always true that just more examples lead to better performances. If the new examples are radically different from the one used to bootstrap the network, performances can even decrease. We tried to investigate the behavior of the model in such scenario, addressing a third research question:

RQ3. To what extent is the model capable to improve its performances with the addition of few annotated examples?

6.3.2 Evaluation Metrics

In our evaluation, we used the very same evaluation metrics described in Section 5.4.2, namely the Average Per-Formula Accuracy (FA), the Average Edit Distance (ED) and the Average Per-Token Accuracy (TA).

6.3.3 Datasets

As for the customization of the data generation pipeline described in Chapter 4.2, we started designing a grammar to generate sentence templates. With
with respect to the grammar used to build the dataset for the tasks presented in Chapter 5, we changed the production rules in order to have more realistic sentences. Before starting to hand-craft the grammar, we qualitatively analyzed different catalogues of definitions, Wikipedia entries, and comments from well-known Ontologies that could be formalized into a formula. In this way we could have an overview of typical grammatical complexity and variability used by humans in writing concepts definitions. Indeed, the grammar presented in Section 5.4.3 was mainly a combinatorial explosion of single constructs for portion of formulæ. This led to the generation of sentences which are correct from the grammatical standpoint but, in many cases, far away from how a human would typically formulate a descriptive sentence. Moreover, we added different cases where a single word should be used to fill different placeholders of the formula template, which is a phenomenon that cannot be handled by the approach presented in the previous Chapter, but that is likely to occur in real world definitions. As an example, think about the concept of “long and winding road” that should be formalized as long\_road \sqcap winding\_road: this would require the word “road” to be used twice as a placeholder fillers. To the extent of the present work, we name such phenomenon coordination.

The actual pipeline used to generate the training examples is similar to the conceptual one described in Chapter 4. The production rules of the grammar, in this case, do not outputs placeholders for roles and concepts, but directly patterns of Part-of-Speech tags. A generic sentence template, generated by our grammar, like:

\[
A \text{ NN is a NN},
\]

will be parsed into the following formula:

\[
\#2 \sqsubseteq \#5
\]
CHAPTER 6. TRANSLATION

6.3. EVALUATION

where the number following the # symbol represents the position of the Part-of-Speech tag within the sentence. All such Part-of-Speech tags, will be replaced by an actual word of the proper class in the actualization phase. Note that, whatever words will be randomly selected in such actualization, the formula will always be a valid formula for the sentence. As an example, consider the two actualization phases, one generating the sentence:

\[ A \text{ snake is a reptile,} \quad (6.18) \]

and the other generating:

\[ A \text{ cow is a mammal.} \quad (6.19) \]

Both of them can be coupled with the formula represented in (6.17). Resolving the copy operation, indeed, we will end up into \( \text{snake} \sqsubseteq \text{reptile} \) and \( \text{cow} \sqsubseteq \text{mammal} \) respectively. In this way, when generating the sentence templates, and corresponding formula template resulting from the parsing and annotation process, we can keep into account coordination phenomena from the very beginning. As an example, let us consider the sentence:

\[ A \text{ loner is a man that walks a long and winding road,} \quad (6.20) \]

to be encoded in the formula:

\[
\text{loner} \sqsubseteq \text{man} \sqcap \exists \text{walk.}(\text{long\_road} \sqcap \text{winding\_road}). \quad (6.21)
\]

and let us focus on the range of the predicate \text{walk}, which is realized at the text surface level with the fragment “long and winding road”. Such fragment is an actualization of the pattern \text{JJ and JJ NN}, which is produced by some production rule of our grammar. When an occurrence of such pattern is parsed, the set of hand-crafted annotation rules turns it into the formula
Our grammar is capable to generate more than 16.5 millions of sentence templates. For the different word classes, we used the same word lists used in 5.4.3. For the ordinal numbers to be used in cardinality restrictions, we used just the NUM special word. Note that even if more than one of such symbols occur in the sentence, there is no ambiguity. Indeed, the extralogical symbols in the formula are defined in terms of their position within the sentence.

Once generated, such examples must be encoded in a machine readable way, namely a way that makes them suitable to be fed into a neural network. First, we collect all the words used in the sentences into the vocabulary $V$, and all the logical symbols into the shortlist $L$. We replace every word of the sentence with its index within the vocabulary, turning the sentence into a set of natural numbers ranging from 1 to $|W|$, the size of the set. The encoding of the formula is slightly more complex. If a symbol in the formula is a logical one, we indicate it with its index within $L$, as for the words. If the symbol must be copied from the $p$-th word of input sentence, we replace it with $|L| + p$. In this way, being $|L|$ fixed for all the sentences, there is no ambiguity in the interpretation of the symbol when turning the formula back into human readable form. This encoding for the output formula is perfectly coherent to the description of the output of the network, that has been given in Section 6.2. An explicit example of this encoding process is given in Tab. 6.1, where $V(\cdot)$ and $L(\cdot)$ are the functions that return the position of their argument in the vocabulary $W$ and the shortlist $L$ respectively, and $\#p$ indicates that the $p$-th word in the input sentence must be copied into the output formula.

The process described so far has been used to build most of the training and evaluation data for our experiment. Though, in the experiment presented
6.3. EVALUATION

<table>
<thead>
<tr>
<th>sentence</th>
<th>human readable form</th>
<th>formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;A bee is an insect.&quot;</td>
<td>( V(A), V(\text{bee}), V(\text{is}), V(\text{an}), V(\text{insect}), V(.) )</td>
<td>( #2 \sqsubseteq #5 ), (</td>
</tr>
</tbody>
</table>

Table 6.1: Sentence and formula encoding.

In Section 6.3.4 we used the collection of manually curated examples presented in Section 4.4 to assess the capabilities of the model to improve its performance against real world examples by extending the training set.

6.3.4 Experimental Settings and Results

In this section, we present all the experiments we run to evaluate the model and answer the research questions presented in the previous section. All the results of the experiments are presented in terms of the evaluation metrics previously defined. All the experiments were run on NC6 Microsoft Azure virtual machines with Ubuntu 16.04 as operating system and provided with one NVidia Tesla K80 core, taking a time lapse between 6 and 7 hours each.

Closed Vocabulary Evaluation

In our first evaluation, we trained different model configurations to learn to translate sentences into formulae in a closed vocabulary setting, i.e. when all the words in the evaluation set appear also in the training data. The objective of this evaluation was to test the network in its capability of dealing with the syntactic structure of the Descriptive Language, in order to answer RQ1.

We trained the network on training sets of grammar generated data of different sizes, and evaluated it against a test set of other 30000 grammar generated examples. We tested different model configurations, i.e. different hyperparameters settings and settled the best configuration to the one reported in Table 6.2, together with the settings for the training algorithm.
The predicted formulae over the set of 30000 evaluation examples have been compared against their gold truth and the metrics presented before have been computed. Results for different sizes of the training set are reported in Table 6.3.

| embedding size | 256 |
| encoder cell | GRU |
| encoder size | 512 |
| encoder dropout | 0.2 |
| attention inner size | 256 |
| decoder cell | GRU |
| decoder size | 512 |
| decoder dropout | 0.2 |
| output feedback size | 30 |
| training algorithm | SGD |
| learning rate | 0.2 |
| training steps | 20000 |
| batch size | 200 |

Table 6.2: Network settings and hyperparameters for the closed vocabulary evaluation. SGD is an abbreviation for Stochastic Gradient Descent, GRU for Gated Recurrent Unit.

<table>
<thead>
<tr>
<th>training set size</th>
<th>FA</th>
<th>ED</th>
<th>TA</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>0.61</td>
<td>2.48</td>
<td>0.92</td>
</tr>
<tr>
<td>5000</td>
<td>0.84</td>
<td>0.60</td>
<td>0.98</td>
</tr>
<tr>
<td>10000</td>
<td><strong>0.89</strong></td>
<td><strong>0.47</strong></td>
<td><strong>0.99</strong></td>
</tr>
<tr>
<td>20000</td>
<td>0.81</td>
<td><strong>0.46</strong></td>
<td>0.98</td>
</tr>
</tbody>
</table>

Table 6.3: Results for the closed vocabulary evaluation.

We achieved an Average Per-Formula Accuracy of 0.89 with 10000 examples, together with an Average Edit Distance of 0.46. Increasing the number of training examples, we lose 0.08 of Average Per-Formula Accuracy, gaining 0.01 in the Average Edit Distance. Looking at the results, it is notable how the huge performance gap is between 2000 and 5000 examples, while with larger training sets the performance improvement is narrower. According to
our evaluation, 5000 training examples are already a good sample for the universe of the language we are dealing with, so that the network can learn how to generalize over it. This provides evidence to the thesis that the syntactic complexity of the definitory language can be handled with just one neural network with pointing mechanism in the scope of a syntactic transformation into a formula, offering support for a positive answer to RQ1.

**Open Vocabulary Evaluation**

To answer RQ2, we run other experiments in a different setting, that we called **open vocabulary**, in which words in a sentence can be unknown. Intuitively, the lexical variability of the language, mostly concerns content words, while the grammatical structure and the behavior of the function words are basically constant. Trying to answer RQ2, we evaluate the ability of our model to leverage the latter to compensate the former, measuring its tolerance to unknown words.

We trained and tested the same model presented in 6.3.4, but on different datasets. Such datasets have 10% of the nouns, 5% of the adjectives, and 5% of the verbs marked with the `<UNK>` symbol and are considered as unknown words. We trained the network on different training sets of different sizes, and evaluated them on 30000 examples. As in the closed vocabulary setting, sentences of both the training and the evaluation sets were sampled from the Descriptive Language generated by our grammar, and then automatically annotated. The model configurations were the same of those reported in Table 6.2, while results are reported in Table 6.4.

Using 20000 training examples, we achieved an Average Per-Formula Accuracy of 0.89, together with an Average Edit Distance of 0.38. Even with half of the training examples, we lose only 0.04 of Average Per-Formula Accuracy and 0.13 of Average Edit Distance. Providing support for a positive answer to RQ2, this confirms that our system is capable to handle such lan-
6.3. EVALUATION

<table>
<thead>
<tr>
<th>training set size</th>
<th>FA</th>
<th>ED</th>
<th>TA</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>0.62</td>
<td>1.51</td>
<td>0.94</td>
</tr>
<tr>
<td>5000</td>
<td>0.86</td>
<td>0.63</td>
<td>0.98</td>
</tr>
<tr>
<td>10000</td>
<td>0.85</td>
<td>0.51</td>
<td>0.98</td>
</tr>
<tr>
<td>20000</td>
<td>0.89</td>
<td>0.38</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Table 6.4: Results for the open vocabulary evaluation.

language variability, exploiting its knowledge of the grammatical structure of the descriptive sentences to infer the proper way of dealing with unknown words.

As in the case of the open vocabulary, also in the closed vocabulary setting the number of examples needed to reach a good result is limited. The trend of the metrics in this case is slightly different from the one visible in the Closed Vocabulary evaluation and summarized in Table 6.3. In the Open Vocabulary setting, all the three metrics improve monotonically following the enlargement of the training set size, while in the Closed Vocabulary setting, only the Average Edit-Distance is following such trend. With respect to Average Per-Formula and Per-Token accuracy, the best performing setting is the one with 10000 training examples. In both cases, anyway, the huge performance gap is between the 2000 examples and the 5000 example cases, suggesting evidence that 5000 examples can be considered as an already effective sampling of the problem space.

Reference Set Evaluation

The grammar generated examples we used in our experiments are intended to be used to bootstrap the model. We envision an evolution process for the training set of our model in which, on the long term, such bootstrap data are progressively enriched and eventually replaced with real world examples, coming from human annotation. However, as previously mentioned, this operation can have drawbacks: if the added examples are few, non-
homogeneous, and radically different from the other ones already in the training set, the performance of the model is not likely to improve and, possibly, it could even get worse. An ideal model is smart enough to capture the regularities expressed by the new examples but also tolerant to the possible dispersion that such examples can bring into the training set. As a matter of fact, the addition of new examples should \textit{normalize} the training set and make it closer to the actual problem space.

We tried to investigate this aspect in RQ3, replicating on a small scale this process of training set enrichment. We trained our system on only bootstrap examples, then we enriched the training set with 75 manually annotated examples, retrained the model and compared the performances of the two trained instances. Such evaluation has been performed against a \textbf{reference set} of 425 examples made of real world examples. The extension of the training set and the reference set come from the split of the dataset of 500 manually curated examples described in Section \ref{sec:dataset}.

We used different amounts of bootstrap examples---2000, 5000, 10000, 20000---generated by our grammar in the Open Vocabulary setting used in \ref{sec:evaluation}. In Table \ref{tab:evaluation} we report the results of the evaluation, showing the explicit number of correctly generated formulæ together with the values of the metrics previously used.

$$
\begin{array}{c|cccc}
\text{training set size} & \text{CF} & \text{FA} & \text{ED} & \text{TA} \\
\hline
2k & 35 & 0.08 & 4.80 & 0.47 \\
2k+75 & 143 & 0.34 & 3.44 & 0.60 \\
5k & 38 & 0.09 & 4.58 & 0.48 \\
5k+75 & 126 & 0.30 & 3.55 & 0.59 \\
10k & 39 & 0.09 & 4.59 & 0.48 \\
10k+75 & 82 & 0.19 & 4.06 & 0.55 \\
20k & 38 & 0.09 & 4.55 & 0.49 \\
20k+75 & 55 & 0.13 & 4.53 & 0.50 \\
\end{array}
$$

\begin{table}[h]
\centering
\caption{Evaluation with manual training set extension of 75 examples for different bootstrap datasets. The models are tested against 425 examples. CF indicates the number correctly translated sentences.}
\end{table}
We obtained the best performances using a learning rate of 0.4 for training the model with 2000, 5000, and 10000 bootstrap examples, and of 0.2 for training the model with 20000 bootstrap examples.

In all the cases, adding the 75 examples to the bootstrap training set, we observe an improvement in the performance of the model in terms of all the evaluation metrics used. This provides support for a positive answer to RQ3, showing that an addition of manually annotated real world examples can actually drive the model to a better comprehension of the actual problem space sampled, in our case, by the reference set of 425 examples.

Analyzing the outcome of this evaluation, we can note that the measure of such improvement changes with the size of the bootstrap training set. In particular, using less bootstrap training examples, we have a higher increase of performance. This happens because the new examples, which are closer to the test examples than the bootstrap examples, are a larger fraction of the extended training set when dealing with a smaller bootstrap training set. In contrast, with a larger size of the bootstrap training set, the performance is more contained.

Nonetheless, in the perspective of a continuous process of extension of the training set and retraining of the model, it can be meaningful to exploit a large bootstrap, capable to provide the model an initial coverage of a wide range of syntactical constructs and of a large vocabulary, with many words used in as many contexts as possible.

6.3.5 Evaluation against baselines

To complement our analysis, and put the overall performances of our RNN-based approach in perspective, we compare it with two baseline systems against the same reference set of 425 manually curated dataset and report the obtained results in Table 6.6.

The first baseline (Grammar Parser) is the system that we used to an-
notate the examples in the synthetic data generation pipeline described in Section 4.2. The sentences of the 425 examples are transformed in suitable sentence templates applying the inverse of the actualization process of the pipeline. Such re-built sentence templates are then fed into a syntactic parser built on the same grammar used in the data generation process. Since the parser is capable only to perfectly parse a sentence or to fail, for this baseline we reported only the Average Per-Formula Accuracy and the number of correct formulæ. Only 17 sentences can be parsed by the grammar. This can give us a measure of the difference between the distribution from which the manually curated data and the bootstrap data come from, explaining the decrease of performance we register when we go from testing the model in the closed-vocabulary evaluation (see Table 6.3) or the open-vocabulary evaluation (see Table 6.4) to testing it against the 425 reference set examples. Indeed, in the former cases, the training data and the test data come exactly from the same uniform distribution, the one across the language that can be generated by our grammar, while the 0.04 of Average Per-Formula Accuracy registered by the Grammar Parser baseline indicates that the difference between the bootstrap training data and the test data from the reference set is remarkable. Nonetheless, even if only trained on grammar generated data, the network is somehow already capable to generalize over the sentence structures defined by the grammar and correctly process roughly twice the sentences that the latter would be capable to convert. The second baseline is the model based on tagging and transduction that has been presented in Chapter 5 learning.

<table>
<thead>
<tr>
<th>system</th>
<th>CF</th>
<th>FA</th>
<th>ED</th>
<th>TA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grammar Parser</td>
<td>17</td>
<td>0.04</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>0</td>
<td>0.00</td>
<td>11.7</td>
<td>0.10</td>
</tr>
<tr>
<td>20k open vocabulary</td>
<td>38</td>
<td>0.09</td>
<td>4.55</td>
<td>0.49</td>
</tr>
</tbody>
</table>

Table 6.6: Comparison of the performances of the proposed approach against two baselines.
The results show that the approach proposed in this Chapter outperforms both baselines. On the one hand, the comparison with the grammar parser confirms that our model learns more than the actual templates used for building the bootstrap dataset (and used for training). On the other hand, the comparison with the model presented in Chapter 5 confirms that the pointing mechanism is more robust than the tagging and transduce solution, overcoming some of the limitations of the latter, such as the 1-to-1 nature of tagging (see Section 7.2).

6.4 Discussion

In this Chapter, we proposed a single network architecture capable to turn a natural language concept definition into an $\mathcal{ALCQ}$ formula through syntactic transformation. This architecture is a Recurrent Encoder-Decoder, endowed with pointing capabilities. In this way, the network can generate an output symbol copying it from the input sentence. This allows the model to use symbols outside the proper output vocabulary, here represented by the shortlist $L$ of the logical symbols. In this way, the network can learn how to use these output symbols even without having an explicit representation of their meaning. This design choices regarding the network architecture has been driven by the goal to overcome the limitation of the one investigated in an earlier stage of our work, namely its inability to deal with coordinations.

We evaluated our model with respect to the capabilities of generalizing over the syntactic structures of the descriptive language and the tolerance to unknown words. Since we used no syntactic or grammatical features (e.g. POS-tags, dependency labels, and so on) to represent the input sentence, the results that we had in our experimental evaluation shown how our model is actually capable to learn how to deal with the syntax of this particular subset of natural language, to understand the underlying logical structure.
of a definition, to understand when a content word must be used as an extralogical symbol, and so on. All this work can be carried out training a single network, while two separate networks have been used in the approach presented in Chapter 5. This allowed us to reduce the number of parameters to deal with and the related cost in terms of design, training, maintenance of the datasets.

We envision an evolution process for our system in which synthetic data produced to the extent of this work are used as bootstrap data and constantly integrated and eventually fully or partially replaced with real world examples. Such enrichment of the training set should ensure a better sampling over the problem space, allowing the network to exploit its generalization capabilities to the maximum.

Even if showing that only the bootstrap data are not per se an exhaustive sample over the Descriptive Language (which, anyway, was not the goal of our investigation), our evaluation gave evidence that it is possible to significantly increase the performance of the model by adding further real world manually annotated examples to the training set. We acknowledge that what provided so far is just a limited—but yet significant—example of the behavior of the model in the context of a training set extension, and a deeper investigation is left as a future work, following the direction that are outlined in Section 8.1.

6.5 Additional Remarks

During our work, we investigated further aspects not directly aiming at answering our research questions. We comment our supplementary findings here.
6.5.1 Different Cell Functions

Before finally opting for Gated Recurrent Units, we experimented with different, more powerful cell functions in our neural network architecture. Being more powerful, these cell functions are characterized by a larger amount of parameters. This implies that the resulting model is more complex and the training phase is more time and resource-consuming. For instance, we tested Long Short-Term Memory (LSTM, see [41]) cells but could not measure any improvement over GRUs in the network performance. For this reason, we suspended the investigation of more complex models and did not considered the idea of exploiting even more complex architectures, such as bidirectional encoders, exploited in the work of [36] that inspired the design of our neural network.

6.5.2 Impact of Random Initialization

In order to verify if the random initialization of the weights could affect the performance of the network we run the same experiments several times. Similarly to the different cell functions, we did not observe any significant variations.

6.6 Implementation Details

The code implementing the models described in this section together with the datasets used for the training and the evaluation are available online at the of the repository of the Deep Knowledge Extraction from Text (DKET) project, at the url https://github.com/dkmbk/dket\footnote{last accessed on January 31, 2018.} The code has been developed in Python 3.5\footnote{https://www.python.org/downloads/release/python-350/ (last accessed on January 31, 2018).} using the TensorFlow 1.2 framework\footnote{https://www.tensorflow.org/versions/r1.2/ (last accessed on January 31, 2018).}
Chapter 7

Related Work

Referring to the background we outlined in Section 2.1, the investigation we carried out in the scope of the present work falls in the area of Ontology Learning, a branch of knowledge extraction focusing on the extraction of terminologies (or TBoxes) from textual sources. More in detail, we are interested in the extraction of complex concept descriptions, namely capable to express something that goes beyond a taxonomical relation. From a technical point of view, the problem we tackled can be described as a sequence transduction problem, in which a descriptive sentence in natural language is translated into a subsumption $\mathcal{ALCQ}$ axiom, as described in Chapter 3. The approach has been practically implemented using different Neural Network architectures, as described in Chapter 5 and Chapter 6.

In this section, the approach we developed is compared against other approaches in the literature tackling the same problem—or, at least, a similar one—from the methodological perspective. We can roughly split the State-of-the-Art approaches into two groups. The first and larger one comprises all those approaches that are based on hand-crafted rules. The second one comprises purely Machine Learning-based approaches.
7.1 Rule-based Approaches

The approaches of this first group strongly rely on manually curated catalogues of rules used for parsing, manipulating and possibly generating text. Given a representation of the text in terms of raw words, and possibly other syntactic, lexical, or semantic features, such rules deterministically define how these features must be manipulated to obtain a formal representation. Being such transformation rules hand-crafted, the human effort is concentrated in the process of exploring how knowledge is expressed in natural language and in the process of identifying which features are actually significant to represent the text in the context of the task. Note that in these approaches, pre-trained statistical learning based systems are actually used with the goal to augment the representation of natural language with additional features others than raw text, such as Part-of-Speech tags, dependency labels, and so on.

LExO \cite{88} is one of the most notable examples in the literature. LExO consists of a semi-automatic approach to extract expressive axioms from complex natural language definitions through syntactic transformation. The pivotal intuition—that we exploited as well in our work—is that syntax matters. First, the input definition is processed with a syntactic parser in order to extract some features to be further processed. Afterwards, the resulting parse tree is manipulated through a set of transformation rules that turn the input sentence into an OWL axiom. Among all the ones presented in this section, the one presented in \cite{88} is the one that can be considered closer to the approach that we presented in this thesis in term of source and target language. More in detail, the authors present a system capable to translate a natural language definition into a $\text{SHOIN}^D$ axiom—which is actually more expressive than the logical language we considered in our work, namely $\text{ALCQ}$. Unfortunately, a direct comparison was not possible as the original
implementation provided in [88] was not regularly maintained, and does not run on current operating systems.

Recently, the authors of [37] proposed an effective approach for Ontology enrichment based on the mapping of natural language text onto Description Logic formulæ. The approach requires a hand-crafted Tree-Adjoint grammar and a hand-crafted lexicon to map a sentence into the corresponding logical form. The assessment of the system is performed verbalizing the Description Logic formula back to natural language and evaluating the BLEU score between the original sentence and the re-verbalized one. Such verbalization is performed through a third component of the system, called surface realizer, comprising a set of hand-crafted verbalization rules. All such components must be manually designed. We want to remark how this approach is tailored on a System Installation Design Principles (SIDP) text corpus, which contains mostly alethic statements, while our approach is focused on descriptive sentences, as defined in 3.1.1.

The work presented in [52] exploits an hand-crafted grammar used to parse natural language sentences. Downstream, a set of transformation rules is applied to the parse tree to obtain an Attempto Controlled English (ACE) compliant version of the input sentence. Afterwards, an ACE parser called Attempto Parser Engine† (APE) is used to turn the latter sentence into an $\mathcal{ALCQ}$ axiom. ACE is used also in [70]. In this other work, the authors have embedded an ACE parser into a larger parsing engine, so that fragments of sentences that fall outside the controlled language can still be tolerated without generating parsing errors. Downstream the parsing operation, a set of hand-crafted syntactic rules are used to turn the syntax tree into a Description Logic formula.

Some other approaches rely on the World Wide Web (WWW) as a source of evidence to support candidate axioms. The authors of [87] further devel-
oped the approach presented in [88] using Pointwise Mutual Information over the WWW in order to detect disjointness among classes. The authors of [82] aim at detecting axioms of symmetry, reflexivity, functionality, transitivity and inverse for object properties from a background Ontology. According to a set of hand-crafted patterns, these object properties are verbalized into text fragments, different for each type of axiom. These text fragments are subsequently used to query different web search engines. Frequency-based metrics on the number of the search hits for each query are used to provide evidence for the different axioms. The authors of [68], first extract a list of textual surface realization of concepts from a text corpus, and then use these text fragments as web search queries in order to enlarge their document base. The new documents are parsed and hypernym relations among concepts are detected checking the occurrence of different ad hoc lexico-syntactic patterns. Named Entities are identified using a pre-trained Natural Language Processing toolkit and subsequently aligned to the different concepts using another set of hand-crafted patterns. Finally, axioms of disjointness and class equivalence between concepts are stated considering the sets of entities aligned to each concept.

A radically different approach, mainly focused on Ontology Population, is presented in [67]. In this work, Discourse Representation Theory (DRT, see [44]) is used in order to represent the content of a single document in formal structures called Discourse Representation Structures (DRSs). First, the text is parsed using a pre-trained parser (Boxer, see [10]) capable to map text surface to DRSs. Those structures are then mapped to OWL constructs applying a set of translation rules and exploiting several external lexical (FrameNet[2] and VerbNet[3]) and ontological (FOAF[4], DBpedia[5], Dolce+DnS


136
Related Work

7.1. Rule-Based

Ultralite resources. The translation from DRSs to OWL constructs relies on a set of Ontology Design Patterns (ODPs). An ODP is “a reusable successful solution to a recurrent modeling problem” and can be seen as a sort of template to be used in some particular and recurring situations. Acting as constraints, they are supposed to ensure quality for the final Ontology construction.

An approach for the extraction of $\mathcal{EL}++$ concepts definitions from text is presented in [50]. Text fragments involving concepts from the SNOMED CT ontology are matched and their lexical and ontological features are used to train a maximum entropy classifier in order to predict the axiom describing the involved entities. Still, downstream of this classifier, hand-crafted patterns are used to generate the formulæ. The approach has been further developed in [64] into an entirely Machine Learning-based approach, where no hand-crafted rule is used.

The main difference between our work and the approaches presented above is that the latter strongly rely on manually designed components: lexico syntactic patterns, text generation rules, and so on. In particular, focusing on the text processing, rule-based approaches for Ontology learning are indeed rigid with respect to the grammatical structure of the text they can process. Therefore, as acknowledged in [58], several linguistic phenomena—such as conjunctions, negations, disjunctions, quantifiers scope, ellipsis, anaphora, and so on—can be particularly hard to parse and interpret. Extending a catalog of hand-crafted rules to handle as more linguistic phenomena as possible can be an extremely expensive task, leading to unsustainable costs in engineering, maintaining and evolving the system. On the opposite, our approach has no hand-crafted rule and its evolution is intended to happen through the extension of the training set by adding further manually annotated training data.

---

7.2 Machine Learning-based Approaches

Recently, another class of approaches to the problem of turning natural language sentences into some formal representation has been proposed. These approaches rely on Machine Learning techniques so that the transformation rules are learnt by examples and not manually designed. The models are trained by submitting examples made of the actual input, namely a text fragment represented with some pre-defined features, and the expected output, namely the corresponding logical form. The model tunes its weights according to the measured difference between the actual output and the expected one. Beside feature engineering, the large part of the human effort is required by the collection of the examples to be used to train and evaluate the model.

Stemming from [50], the authors of [64] present a Machine Learning based approach to formalize textual definitions of biomedical concepts. Training examples are generated via distant supervision, aligning definitions coming from several collections of biomedical texts—such as MeSH\textsuperscript{9} MEDLINE\textsuperscript{10} and WikiPedia articles on the topic—with a catalog of relations coming from well known resources on the same domain, like SNOMED CT and the SemRep\textsuperscript{11} corpus. First, all the textual occurrence of the concepts are associated to their semantic types. These semantic types together with a minimal set of linguistic features are subsequently used as input features to train different classifiers.

On the opposite, our approach does not rely on any manual feature engineering and the input representation is implicitly learnt as a vector space

model. Moreover, differently from ours, this approach is focused on a specific domain and there is no tolerance to unknown words. In addition, the target logic language is the one used in SNOMED CT, namely $\mathcal{EL}^{++}$, which is less expressive than $\mathcal{ALCQ}$. A more subtler difference between our approach and the one presented in [64] is in the way the output formula is produced. In our case, both the sentence template produced by the Transduction network presented in Section 5.3 and the formula produced by model presented in Chapter 6 are not selected from a pool of predefined ones through a classification operation. Indeed, these output sequences are generated and the classification operation concerns the single symbol to be chosen at each timestep. On the opposite, the structures of the concept description axiom that the approach presented in [64] can produce are fixed. Borrowing some terminology from Chapter 5, this approach can be seen as capable of classifying—and not generating—the proper formula template and, at the same time, aligning the textual surface realization of concepts and roles.

Outside the Ontology Learning and Knowledge Engineering communities, the authors of [21] exploit a neural network to translate natural language sentences into a logical representation of database queries. The presented architecture shares some similarities with the one presented in Chapter 6, since a Recurrent Encoder-Decoder configuration endowed with attention mechanism is used in both the approaches. Though, the former is not provided with any pointing mechanism. As a consequence, unknown must be replaced by their types and some numeric identifier—e.g. NUM$_0$ for the first number occurring within the sentence. This constrains the model to the same limitations we pointed out about the tag numbering for the approach we presented in Chapter 5. Introducing the pointing mechanism, we completely overcome such limitation since unknown words can be copied into the output sequence as they are.
Chapter 8

Conclusions and Future Work

The investigation carried out during this thesis aimed at designing and evaluating a system capable to extract Description Logic terminological axioms from natural language text. More in detail, we considered as a suitable input language that particular subset of natural language that we identified as Descriptive Language, namely the set of all the sentences that a human agent can use to express salient characteristics of a species. Likewise, we constrained the output formula to be a concept description expressed as an $\mathcal{ALCQ}$ inclusion axioms. Given a descriptive sentence, the corresponding formula is obtained through syntactic transformation, so that every symbol appearing in the formula will be a logical symbol from the target logic language or a word from the input sentence. Our approach is fully based on Machine Learning techniques, in particular Neural Networks, while the vast majority of the akin works in literature follows a rule-based approach. In our work, the syntactic transformation of a descriptive sentence into the corresponding $\mathcal{ALCQ}$ formula is not performed following some pre-determined rules but is learnt by examples, namely pairs made of a descriptive sentence and a corresponding formula.

The first solution we proposed exploited two different Recurrent Neural Networks, each of which was trained for a different task. The first network
was trained to perform a transduction from the input sentence to a formula template, namely a corresponding formula were placeholders are used instead of extralogical symbols. The second network is trained for a tagging task, so that the label of the corresponding (possibly none) placeholder can be assigned to each word in the sentence. In this way, we are able to rebuild the final output formula using as extralogical symbols, for each placeholder, those words that have been tagged with the same label. This solution has been presented in Chapter 5.

The evaluation process carried out for this model allowed us to assess the capabilities of Recurrent Neural Network-based architectures to deal with the descriptive language as input. In particular, we assessed how the model is capable to generalize over the syntactic structures of the Descriptive Language and to tolerate unknown words. Moreover, our evaluation found out that some strong limitations make this approach unpractical. The main limitation is that the tagging phase can assign one and only one placeholder label to each word. As a consequence, a word cannot be used as a filler for more than one placeholder in the formula template. This makes the approach unusable when dealing with all those phenomena that we referred to as “coordinations” (see Section 6.3.3).

Following the advances in the field of Deep Learning, we could design a second model capable to overcome all the limitations of the previous one. Exploiting so-called Pointing Networks, we trained a single Neural Network which is capable, at each timestep, to generate directly the output formula producing a logical symbol, drawn from the set of all the logical symbol of the target language, or copying a word from the input sentence to be used as an extralogical symbol. This allows us to process descriptive sentences of arbitrary structure. Moreover, the output vocabulary we have to use is extremely limited, comprising only the logical symbol of the target Description Logic language. This second solution has been presented in Chapter 6.
CHAPTER 8. CONCLUSIONS AND FUTURE WORK

As for the first proposed model, the evaluation assessed the capability of the model to deal with the syntactic structures of the descriptive language and with unknown words using only raw text and no other additional feature. In addition, we also assessed the capability of the network to improve its performance in a scenario where new training examples are added to a first training set used to bootstrap the training process. In this latest stage of evaluation, both the additional training examples and the test set have been taken from a manually curated dataset.

The test set made of manually curated examples has been used also to compare the two models one against the other. The latter model outperforms the former and it outperforms also the grammar used to generate the bootstrap data, used in a deterministic parser as a baseline.

To sum up, we believe that, if adequately trained, the model we proposed in the latest stage of our investigation can provide a sound and general solution to the problem of turning a descriptive sentence into an $\mathcal{ALCQ}$ concept description through syntactic transformation.

One of the prominent issue we had to tackle during our work is the fact that the Knowledge Engineering community lacks a shared dataset suitable for a purely Machine Learning-based approach to the Ontology Learning problem. Manually building such a dataset would have been extremely costly and time consuming, so we set up a synthetic data generation pipeline. In this way, we could generate datasets suitable to evaluate our models with respect to our goal. Though, such datasets still remain a reasonable but yet limited approximation of the descriptive language that we want to model. Putting a further research effort in the direction of providing more an better bootstrap training data could strengthen our model from the empirical point of view. This goal sets the agenda for future investigation activities.
8.1 Future Work

Large part of the future effort shall be put in the direction of improving the quality of the bootstrap data with more and better—namely more realistic—examples. Such improvement will have to cover both the increasing complexity of the input language and the amount of logical constructs of the target language so to cover the whole OWL-DL language. The most straightforward way would be to improve the grammar underlying the generation of the bootstrap data. We are not planning to follow such direction since, on the long run, we would face a maintenance cost comparable with the one of rule-based approaches. Moreover, this way will reflect any prior bias of the Knowledge Engineer in charge of the grammar development, with the risk of ending up in some biased representation of reality.

Instead, a valuable direction of investigation could be the usage of some distant supervision based approach, as in [64], or the exploitation of generative models, as in [47], in order to generate new annotated examples.

In a distant supervision approach, a set of heuristics is exploited to align input items, in our case descriptive sentences, to corresponding output items, in our case terminological axioms. The main cost of such process is in the collection of the catalogues of such collections of input and output items and in the design of the heuristics to be used to align the new training examples.

Generative models can be used to extend the training set as well. When a seed of training examples—obtained from manual annotation, distant supervision, or from our data generation pipeline—is already present, together with other unannotated sentences, we can jointly train a Neural Network for two different tasks. The first task is to translate the input sentence into the corresponding output formula, like in our approach. The second task is to rebuild the input sentence. When submitting a training example, this can be a pair made of a sentence and the corresponding formula or just an annotated...
sentence. In the latter case, the only contribution to the loss—and consequently to the learning process—is given by the error in the reconstruction of the sentence. Once such a system has been trained, the training set is then extended with new pairs made of an unannotated input sentences and the corresponding output formula generated by the network.

Though still artificial and carrying some noise, examples coming from distant supervision and from generative models are expected to be more realistic with respect to the current grammar generated ones. Upstream, some semi-automatic approach to align descriptive sentences and Description Logics formulæ could be explored in the future to set up a more robust and effective bootstrap data generation process.

In this perspective, an already ongoing research effort is aiming at discriminate descriptive sentences within a text. In this way, large encyclopaedic entries can be quickly processed in order to keep only those sentence expressing some statement which is meaningful from an ontological modeling point of view. This possibility could dramatically reduce the cost of the building of a suitable training set for our model. Indeed, as we pointed out in Chapter 4 in the small annotation campaign we ran in order to estimate the cost of the annotation process, the vast majority of the time spent was actually devoted to the activity of mining descriptive sentences from larger text fragments. Moreover, combining such a system with the one we designed and evaluated in this thesis could bring to the construction of an all-in-one pipeline capable to scan a large text, isolate all the descriptive sentences, and produce the corresponding formalizations.

The network is trained in a end-to-end fashion: raw text is the only feature used to represent the input and no further annotation (such as POS tags, dependency labels, and so on) that could be produced upstream by some Natural Language Processing toolkit. Pre-processing the input text in order to have such annotations, and in particular representing the text as a de-
dependency graph, could allow us to exploit more complex neural architectures that rely on graph representations of the input such as Recursive Neural Networks (see [78]) or Graph Convolutional Neural Networks (see [46]), which are capable to exploit expressive representation of the sentence. Though, we should consider that the toolkits exploited to produce these annotations are pre-trained on a portion of the natural language which is different from the one we are interested in. Even if adding more features can ideally improve the generalization capabilities of the model, this misalignment could introduce a significant amount of noise, capable to cancel such improvement. As a consequence, the introduction of such features must be tested carefully.

In general, from the point of view of the architecture of the model, other more complex model could be exploited like Long Short-Term Memory Networks (see [41, 72]) or Memory Networks (see [90]). Though, during our evaluation we did not measure any significant improvement over the simpler Gated Recurrent Unit configuration. This is probably due to the fact that Gated Recurrent Units are enough powerful to capture the expressivity of our bootstrap data. Though, we are confident that more complex models will be capable to learn the structure of a more expressive language. So we think that this direction of investigation could be profitable once a better—in the sense expressed before—training set has been produced.

In the end, we are confident that the approach we proposed in this work can be an effective tool for the Knowledge Engineering and Semantic Web community, and we hope that our contributions will encourage a shared effort in building a widely accepted dataset for the task.
Appendices
Appendix A

Data Generation Pipeline Grammar

In this chapter we present the production rules of the Context Free Grammar used at the first stage of our data generation process. Each production rule is represented with:

- left hand side, representing a nonterminal symbol;
- an arrow;
- a right hand side, comprising one or more nonterminal symbols or a single terminal symbol.

Each terminal symbol is made itself of two parts:

- an actual word, such as “the”, or a placeholder for a content word to be actualized in a latter stage. The placeholder we used are @NN for names, @JJ for adjectives, @VB for verbs and @NUM for a cardinal number;
- a part of speech tag, taken from the Penn Treebank catalogue, where we clusterized all the nouns to NN and VB for all the verbs used to express roles.

The full catalogue of these production rules are reported below.

\[ S \rightarrow \text{ARDEF} \_\text{PERIOD} \_\text{EOS} \]
\[ S \rightarrow \text{DEFPRED} \_\text{PERIOD} \_\text{EOS} \]
APPENDIX A. DATA GENERATION PIPELINE GRAMMAR

a -> 'a/DT'
all -> 'all/DT'
are -> 'are/VBP'
also -> 'also/RB'
at -> 'at/IN'
and -> 'and/CC'
any -> 'any/DT'
anything -> 'anything/NH'
but -> 'but/CC'
do -> 'do/VBP'
every -> 'every/DT'
everything -> 'everything/NH'
extactly -> 'exactly/RB'
is -> 'is/VBZ'
least -> 'least/JJS'
less -> 'less/JJR'
more -> 'more/JJR'
most -> 'most/JJS'
no -> 'no/DT'
nor -> 'nor/DT'
not -> 'not/RB'
of -> 'of/IN'
only -> 'only/RB'
or -> 'or/CC'
some -> 'some/DT'
something -> 'something/NH'
than -> 'than/IN'
that -> 'that/WDT'
the -> 'the/DT'
_COMMA -> ',./'
_PERIOD -> './'
EDS -> '<EDS/>/<EDS>'

NN -> '@NN/NN'
JJ -> '@JJ/JJ'
NUM -> '@NUM/CD'
PRED -> '@VB/VB'

A^EVERY -> a | every
ARE^ALSO -> are | are also
AND^OR -> and | or
AND^OR^BUT -> and | or | but
ALL^THE -> all | all the
COMMA^OR -> or | _COMMA or
COMMA^AND -> and | _COMMA and
IS^ALSO -> is | is also

ROLE -> SINGLE^PRED^ROLE | DOUBLE^PRED^ROLE

TH^THAT^ROLE -> ANYTH^THAT^ROLE | SOMETH^THAT^ROLE | EVERYTH^THAT^ROLE
ANYTH^THAT^ROLE -> anything that ROLE
SOMETH^THAT^ROLE -> something that ROLE
EVERYTH^THAT^ROLE -> everything that ROLE
APPENDIX A. DATA GENERATION PIPELINE GRAMMAR

A^EVERY^CLASS^NP -> A^EVERY CLASS^NP
ALL^THE^CLASS^NP -> ALL^THE CLASS^NP
CLASS^NP^THAT^ROLE -> CLASS^NP that ROLE
DIFF^SING -> a CLASS^NP^THAT^ROLE | SOMETH^THAT^ROLE
DIFF^PL -> CLASS^NP^THAT^ROLE | TH^THAT^ROLE
CLASS^CONJ^PL -> CLASS^NP AND^OR CLASS^NP
CLASS^CONJ^SING -> a CLASS^NP AND^OR a CLASS^NP
JJ^CLASS^NP -> JJ | CLASS^NP
DIFF^COPULA^PL -> CLASS^NP that are JJ^CLASS^NP
DIFF^COPULA^SING -> a CLASS^NP that is JJ^CLASS^NP

ARDEF -> A^EVERY^CLASS^NP IS^ALSO DIFF^SING
ARDEF -> A^EVERY^CLASS^NP IS^ALSO CLASS^CONJ^SING
ARDEF -> A^EVERY^CLASS^NP IS^ALSO DIFF^COPULA^SING
ARDEF -> ALL^THE^CLASS^NP ARE^ALSO CLASS^CONJ^PL
ARDEF -> ALL^THE^CLASS^NP ARE^ALSO CLASS^NP^THAT^ROLE
ARDEF -> CLASS^NP ARE^ALSO DIFF^PL
ARDEF -> CLASS^NP ARE^ALSO CLASS^CONJ^PL
ARDEF -> CLASS^NP ARE^ALSO DIFF^COPULA^PL

DEFPRED -> CLASS^NP ROLE | A^EVERY^CLASS^NP ROLE | ALL^THE^CLASS^NP ROLE

CARD^EXACT -> NUM | exactly NUM
CARD^GT -> more than NUM
CARD^GTE -> at least NUM | not less than NUM
CARD^MIN -> CARD^GT | CARD^GTE
CARD^LT -> less than NUM
CARD^LTE -> at most NUM | no more than NUM
CARD^MAX -> CARD^LT | CARD^LTE
SINGLE^CARD -> CARD^EXACT | CARD^MIN | CARD^MAX
DOUBLE^CARD -> CARD^MIN AND^OR^BUT CARD^MAX
DOUBLE^CARD -> CARD^MAX AND^OR^BUT CARD^MIN

REG^RANGE -> CLASS^NP
REG^RANGE -> OR^JJ^CHAIN | AND^JJ^CHAIN
REG^RANGE -> OR^NP^CHAIN
REG^RANGE -> OR^DET^CHAIN | AND^DET^CHAIN
FULL^RANGE -> REG^RANGE
FULL^RANGE -> AND^NP^CHAIN

SINGLE^PRED^ROLE -> PRED FULL^RANGE
SINGLE^PRED^ROLE -> PRED only FULL^RANGE
SINGLE^PRED^ROLE -> PRED also FULL^RANGE
SINGLE^PRED^ROLE -> PRED SINGLE^CARD REG^RANGE
SINGLE^PRED^ROLE -> PRED DOUBLE^CARD REG^RANGE
SINGLE^PRED^ROLE -> SINGLE^ZERO^PRED REG^RANGE
SINGLE^ZERO^PRED -> do not PRED any | do not PRED | PRED no

DOUBLE^PRED^ROLE -> PRED^OR^PRED FULL^RANGE
DOUBLE^PRED^ROLE -> PRED^OR^PRED only FULL^RANGE
DOUBLE^PRED^ROLE -> PRED^OR^PRED also FULL^RANGE
DOUBLE^PRED^ROLE -> PRED^AND^PRED FULL^RANGE
DOUBLE^PRED^ROLE -> PRED^AND^PRED only FULL^RANGE
APPENDIX A. DATA GENERATION PIPELINE GRAMMAR

DOUBLE^PRED^ROLE -> PRED^AND^PRED also FULL^RANGE
DOUBLE^PRED^ROLE -> DOUBLE^ZERO^PRED REG^RANGE

PRED^OR^PRED -> PRED or PRED
PRED^AND^PRED -> PRED and PRED
DOUBLE^ZERO^PRED -> do not PRED nor PRED any | PRED or PRED no

CLASS^NP -> NP
NP -> NP^MIN | NP^OF^MIN | NP^LARGE
NP^MIN -> NN | JJ NN | NN NN | JJ NN NN
NP^OF^MIN -> NN of NN | JJ NN of NN | NN of JJ NN
NP^LARGE -> JJ JJ NN | NN NN of NN | NN of NN NN
OF^MIN -> of NN | of JJ NN

OR^JJ^CHAIN -> JJ or JJ NN | JJ _COMMA JJ COMMA^OR JJ NN
AND^JJ^CHAIN -> JJ and JJ NN | JJ _COMMA JJ COMMA^AND JJ NN
OR^NP^CHAIN -> NP^MIN or NP^MIN | NP^MIN _COMMA NP^MIN COMMA^OR NP^MIN
AND^NP^CHAIN -> NP^MIN and NP^MIN | NP^MIN _COMMA NP^MIN COMMA^AND NP^MIN
OR^DET^CHAIN -> NP^OF^MIN or NP^MIN
AND^DET^CHAIN -> NP^OF^MIN and NP^MIN
Bibliography


[8] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the


[34] Nicola Guarino, Daniel Oberle, and Steffen Staab. What is an ontology? In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, In-


[82] Luis Del Vasto Terrientes, Antonio Moreno, and David Sánchez. Discovery of relation axioms from the web. In Yaxin Bi and Mary-Anne Williams, editors, Knowledge Science, Engineering and Management,


[88] Johanna Völker, Pascal Hitzler, and Philipp Cimiano. Acquisition of OWL DL axioms from lexical resources. In Proceedings of the 4th Eu-


