UNIVERSITÀ DELLA CALABRIA

Università della Calabria

Dipartimento di Matematica e Informatica

Dottorato di Ricerca in Matematica e Informatica

XXXV ciclo

Tesi di Dottorato

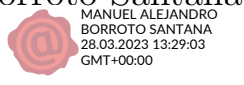
SPARQL-QA: A system for Question Answering over RDF(S) Knowledge Bases

Settore Scientifico Disciplinare INF/01 – INFORMATICA

Coordinatore: Ch.mo Prof. Giorgio Terracina

Supervisore: Ch.mo Prof. Francesco Ricca

Dottorando: Dott. Manuel Alejandro Borroto Santana



Contents

Sommario

I significativi progressi nei campi del Semantic Web e dell'Intelligenza Artificiale hanno portato alla nascita e all'evoluzione delle basi di conoscenza (KB in inglese), come un'alternativa ai database tradizionali per l'archiviazione, l'organizzazione e la condivisione della moltitudine di informazioni eterogenee generate ogni giorno. I meccanismi utilizzati per modellare e memorizzare i dati nelle KB consentono ai primi di acquisire significato semantico e permettono l'interconnessione di più insiemi di dati per creare databases ancora più estesi. Questa realtà viene evidenziata dall'esistenza di grandi basi di conoscenza come DBpedia e Wikidata. Tuttavia, l'accesso alle numerose informazioni disponibili nei KB pubblici potrebbe essere difficoltoso per gli utenti che non hanno familiarità con i linguaggi formali; come il linguaggio di interrogazione SPARQL e i linguaggi di definizione delle ontologie.

In questo contesto, il presente lavoro propone *SPARQL-QA*, un sistema rivolto al campo del Knowledge Base Question Answering (KBQA). L'obiettivo principale di KBQA consiste nel superamento dei problemi di accessibilità alle basi di conoscenza, fornendo sistemi in grado di rispondere alle domande poste in linguaggio naturale su una KB. Molti sistemi di KBQA mirano a tradurre automaticamente le domande nelle corrispondenti query SPARQL da eseguire sulla KB, per ottenere le risposte. In quest'area, i sistemi basati sul Neural Machine Translation (NMT) si sono dimostrati una valida proposta per affrontare il problema della KBQA, ma le loro prestazioni diminuiscono nel riconoscere parole fuori dal vocabolario (OOV in inglese) del set di addestramento. L'approccio proposto si basa su una combinazione innovativa di Neural Machine Translation, Named Entity Recognition (NER) e Entity Linking (EL) per affrontare il problema della generazione di query SPARQL da domande poste in linguaggio naturale, così da minimizzare l'impatto delle parole OOV. A tal fine, *SPARQL-QA* sfrutta un modello NMT per ottenere un template di query SPARQL in cui le possibili entità della KB sono sostituite da segnaposti. Utilizza infine un modello NER per rilevare le entità che vengono menzionate nella domanda. Le già menzionate entità sono collegate alla KB attraverso il meccanismo di EL ed utilizzate poi, per istanziare il template di query, precedentemente citato, così da ottenere la query finale. Un'analisi empirica dimostra che il sistema è efficace e resistente alle

parole OOV e fornisce buone prestazioni in set di dati noti per il Question Answering su DBpedia e Wikidata.

SPARQL-QA ha vinto la decima edizione della Question Answering over Linked Data (QALD) Challenge, tenutasi nel maggio 2022.

Abstract

The significant advances in the Semantic Web and Artificial Intelligence fields have led to the rise and evolution of knowledge bases (KB) as an alternative to traditional databases for storing, organising and sharing the large volumes of heterogeneous information generated today. The mechanisms used to model and store data in the KBs allow information to acquire semantic meaning and enable the interconnection of multiple datasets to create even more extensive databases. As evidence of this reality, there are today vast knowledge bases such as DBpedia and Wikidata. However, accessing the valuable information available in public KBs might be complicated for those users unfamiliar with formal languages, such as the SPARQL query language and the ontology definition languages.

In this context, this work proposes *SPARQL-QA*, a system targeting the field of Knowledge Base Question Answering (KBQA), which focuses on overcoming the knowledge base accessibility issues by providing systems capable of answering questions posed in natural language on a KB. Many KBQA systems aim at translating questions automatically into the corresponding SPARQL queries to be executed over the KB to get the answers. In this area, Neural Machine Translation-based systems proved to be a valid proposition for dealing with the KBQA problem but easily fail to recognise words that are Out Of the Vocabulary (OOV) of the training set. OOV words are a serious problem when dealing with large ontologies where the list of entities is huge and easily evolves over time. The proposed approach relies on a combination of Neural Machine Translation (NMT), Named Entity Recognition (NER) and Entity Linking (EL) to address the problem of generating SPARQL queries from questions posed in natural language while minimising the impact of the OOV words. To this end, *SPARQL-QA* leverages an NMT model to predict a SPARQL query template where possible KB entities are replaced by placeholders and then uses a NER model to detect entity mentions in the question. The mentions are linked to the KB and used to instantiate the former query template to obtain the final query. An empirical analysis shows that the system is effective and resilient to OOV words and delivers good performances in well-known datasets for Question Answering over DBpedia and Wikidata.

SPARQL-QA won the tenth Question Answering over Linked Data (QALD) Challenge held in May 2022.

Introduction

Context and motivation. Information has always played a fundamental role in the need for human beings to know and control the reality surrounding them. In order to preserve and transmit information, humanity has used various methods throughout time, where writing on stone marked the beginning of a process that has not stopped to this day. In this process, the paper has been the quintessential form of data storage, allowing the creation of documents, manuscripts, books, letters and others, which have made it possible to transmit information for centuries. Nowadays, the world lives in the Digital Age, where the adoption and proliferation of computers, in all its extension, have propitiated the management and storage of information digitally, relegating the so-called hard copy formats to a second place. In this context, the enormous development achieved in the IT field has moved the world in a direction where knowledge is generated and shared at great speed and volume. As evidence of this, there are now vast and complex knowledge bases that allow gathering large volumes of information through the intercommunication of thousands of datasets from different domains in what is known as Linked Data [20]. Unlike other forms of structured information, such as relational databases, these knowledge bases commonly follow an ontological organisation where the data is represented according to a conceptual hierarchical structure, and the properties and relationships between entities are well-defined. This data representation allows the information to acquire semantic meaning [47]. The existence of knowledge bases means that people have at their disposal an unimaginable amount of rich information, with DBpedia [68] and Wikidata [118] as real and successful examples of this reality. However, searching and retrieving this data poses a big challenge for lay users since it requires knowledge and understanding of both the knowledge base definition (in terms of entities and properties) and the formal query language, such as SPARQL [3]. Thus, systems implementing the

task of answering questions posed in natural language on a knowledge base, called Knowledge Base Question Answering (KBQA), have the potential to overcome this problem because they remove all technical complexity from the final users.

Question Answering (QA) is an old but very active field of research within computer science, which attempts to find direct answers to questions posed in natural language by humans [61]. There are two families of QA: open-domain QA, where there is no restriction to the domain of the questions, and closed-domain QA, in which questions are bound to a specific domain, such as medicine or sports. Some very early QA systems [46, 120] were designed for closed domains and were conceived as natural language interfaces to databases [61]. QA systems are commonly implemented using two fundamental paradigms: Text-based Question Answering, which relies on the vast amount of information found in the form of text on the Web or in specific document collections, and Knowledge Base Question Answering, which relies on knowledge bases to obtain the answers [61].

Today, Artificial Intelligence (AI) techniques for question answering in natural language have taken a central role in the area of the Semantic Web to address the data accessibility issue, contributing to making linked data and AI two of the hottest interrelated topics in computer science. Indeed, in recent years, many AI systems have been proposed in the field of KBQA that can create SPARQL queries [28, 87, 104, 131]. The most recent and effective proposals take advantage of the great development achieved by Deep Learning in the last few years and use deep neural networks to tackle the problem [67]. Roughly, the approach aims at developing an AI model that learns from a training set of question-query pairs how to perform translation from natural language to SPARQL; ideally, the AI is expected to generalise, i.e., it is expected to translate correctly also other questions not present in the training set.

Advances in the Neural Machine Translation field have led to a new family of approaches that build on the idea of treating the problem as the translation between two languages. Existing systems based on neural-machine translation [76, 87, 104] are very effective but are sensitive to the problem of recognising words that are Out Of the Vocabulary (OOV) of the training set. It is worth noting that OOV is a serious problem while querying large or evolving ontologies with very many individuals that are not (or cannot be) mentioned in the training phase.

In this context, this thesis proposes *SPARQL-QA*, an NMT-based AI sys-

tem for KBQA that more explicitly considers the problem of dealing with OOV words. The core of the architecture is based on a Neural Machine Translation (NMT) [16] model, which relies on Bidirectional Recurrent Neural Networks [110], *trained side-by-side* with a Named Entity Recognition (NER) model, implementing a BiLSTM-CRF network [58]. The NMT module translates the input NL question into a SPARQL template, whereas the NER module extracts the entities from the question. The final SPARQL query is the result of combining the outputs of the two modules, i.e. instantiating the query template with the identified entities. As part of the proposed approach, a training dataset format is also introduced, which helps reduce the output space and is essential for the system's proper functioning. This format also allows tackling the problem with OOV words, a major weakness of most related approaches today. The system was empirically tested on several question-answering datasets targeting DBpedia and Wikidata knowledge bases, namely the Monument dataset [104], QALD-9 [83], QALD-10 [9], and LC-QuAD v1 [114].

Contributions. Intending to deal with the accessibility issues affecting the knowledge bases, the main contributions of the thesis are:

- An NMT-based system to query knowledge bases using natural language questions that relies on ideas presented in the literature but mitigates the impact of the OOV words problem, an issue that can hinder the application of state-of-the-art KBQA implementations in real-world scenarios. To this end, the QA system exploits a combination of Neural Machine Translation, Named Entity Recognition and Entity Linking.
- A novel training dataset format that reduces the output space in the NMT task, thus improving the training times and allowing for dealing with the OOV words problem.
- An ensemble entity linking method that builds on top of state-of-the-art entity linking approaches to support the correct entity identification during the query instantiation phase.

It is worth mentioning that the approach presented in this thesis won the tenth Question Answering over Linked Data (QALD) Challenge held in May 2022, demonstrating that it can achieve state-of-the-art performance on renowned datasets.

Publications. Some of the contributions of the thesis have been subject of the following scientific publications.

- Manuel Borroto, Bernardo Cuteri, and Francesco Ricca. Mitigating the impact of out of vocabulary words in a neural-machine-translation-based question answering system. In *DeepOntoNLP*, volume 2918 of *CEUR Workshop Proceedings*, page 20. CEUR-WS.org, 2021.

- Manuel Borroto, Bernardo Cuteri and Francesco Ricca. A system for translating natural language questions into SPARQL queries with neural networks: Preliminary results. In *SEBD*, volume 2994 of *CEUR Workshop Proceedings*, page 226. CEUR-WS.org, 2021.

- Manuel Borroto, Bernardo Cuteri, and Francesco Ricca. A Neural-Machine-Translation System Resilient to Out of Vocabulary Words for Translating Natural Language to SPARQL. In *AI*IA*, volume 13196 of *Lecture Notes in Computer Science*, pages 171–184. Springer, 2021.

- Manuel Borroto, Francesco Ricca, Bernardo Cuteri, and Vito Barbara. SPARQL-QA enters the QALD challenge. In *NLIWoD7*, volume 3196 of *CEUR Workshop Proceedings*, page 25. CEUR-WS.org, 2022.

- Manuel Borroto and Francesco Ricca. (under review). SPARQL-QA-v2 System for Knowledge Base Question Answering. In *Expert System with Applications*, 2022.

Hereafter, the document is structured as follows. Chapter 1 provides some notions necessary to facilitate the comprehension of knowledge bases and introduces some essential technologies in this field. The chapter ends with an analysis of the knowledge bases utilised during the research. Chapter 2 focuses on the topic of Machine Translation, with particular emphasis on translation using neural networks. To this end, the chapter visits the types of neural networks, architectures and techniques commonly used in this field. The content in Chapter 3 introduces and discusses the problem of answering questions using information from knowledge bases, which is the main focus of this research. Chapter 4 describes the architecture of the proposed system for Knowledge Base Question Answering. The experimental results are presented and discussed in Chapter 5, while Chapter 6 analyses some approaches related to the Knowledge Base Question Answering field. Finally, the document presents the conclusions and states possible works for the future.

Chapter 1

Knowledge Bases

Traditionally, digital data has been stored in different formats ranging from simple text documents to complex relational and non-relational databases that organize the information in a way that makes sense to support, for example, a given information management system. As an alternative to these kinds of storage and motivated by the development of Artificial Intelligence (AI), the knowledge bases (KB) propose to model and store the information in such a way that it can also be interpretable by machines [47]. Informally, a knowledge base is a set of data about a domain of interest that is suitable to be managed by an engine reasoning about the facts modelled in the knowledge base itself, e.g., query existing knowledge or obtain new knowledge. However, to correctly understand how KBs effectively allow this kind of reasoning, it is necessary to refer to a set of concepts and languages developed over time to support the creation of such KBs. To this aim, this chapter covers some of those concepts, formalisms and languages that allow a better understanding of the knowledge bases and how they are created, accessed and shared.

1.1 Ontologies

Philosophers first defined the term *Ontology* to name the field of philosophy concerned with studying reality, being, or existence. It includes questions about how entities are grouped into categories and how they interact with each other. In computer science, the term was adopted by AI researchers to denote a way of representing information that makes it possible to add semantic meaning to a set of data, enabling certain kinds of automated rea-

soning. An *Ontology* (computer science) is a *formal explicit description of knowledge as a set of concepts within a domain and the relationships that hold between them*. To enable such a description, it is necessary to formally specify components such as individuals, classes, attributes, and relations, as well as restrictions, rules, and axioms [47]. As a result, an ontology definition allows for:

- Sharing common understanding of the structure of information among people or software agents;
- Enabling reuse and/or extension of domain knowledge, as it is possible to use existing ontologies to describe portions of a new one or enrich actual knowledge by extending the underlying ontology with new attributes, properties, etc.;
- Making explicit domain assumptions that are easy to change if the knowledge about the domain changes;
- Analyzing domain knowledge as a declarative specification of terms is available.

Ontologies are the most popular way of conceptually modelling information in knowledge bases in the AI field, so it is common to state that: *An ontology, together with a set of class membership statements, constitutes a knowledge base*. More details on the formalisms for writing ontologies will be given in Section 1.3.

Pasta Ontology Figure 1.1 graphically depicts the *Pasta Ontology*, a simple and hypothetical ontology based on the domain of *pasta*, a typical Italian food. It will be used as a reference during this chapter to explain different processes supporting KB development. The ontology is composed of a class *Pasta* representing all types of pasta, which in turn has two subclasses, *Long* and *Short*, representing more specific types of pasta. The classes *Liscia* and *Rigata*, in turn, refine the class *Short*. There are also the classes *Pasta_Factory* and *Wheat* that group all the pasta factories and wheat types, respectively. Instances of *Pasta* can have properties describing the name, the cooking time, the type of drawing, the producer and the kind of wheat used for its production. The last two properties establish interactions with

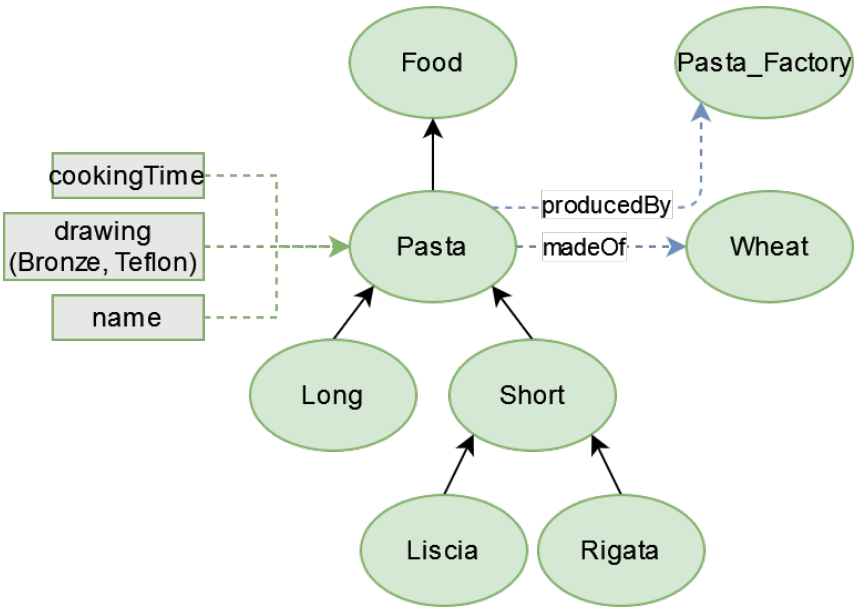


Figure 1.1: An ontology in the Pasta domain. Ovals are used to denote classes, while rectangles correspond to attributes of a class.

instances of the *Pasta_Factory* and *Wheat* classes. The pasta drawing property is constrained to have one of two values, *Bronze* or *Teflon*. Finally, the class *Food* is the superclass for all kinds of food.

1.2 Resource Description Framework (RDF)

RDF is a standard model for data interchange on the Web that is used for representing highly interconnected data. RDF simplifies the merging of data even when the schemas representing the data are different and allows the evolution of data schemas over time without the need to change all data consumers [7]. It is one of the three foundational technologies of the Semantic Web, along with SPARQL [3] and OWL [1].

In the RDF standard, the basic structure for representing the information is a triple (or statement) in the *subject-predicate-object* form, where predicate p establishes a binary relationship between the subject s and the object o. A key aspect is that RDF uses Universal Resource Identifiers (URI) to identify the elements of a triple, allowing them to be consistent across multiple

databases, even if the databases are not local. This mechanism allows for linkages between different triples by referring to a given resource URI (either in subject or object position). In a triple, the *subject* can be a URI, or it can also be a *blank node*[1]. The *object* can be a URI, a literal value, or a blank node. Finally, predicates are always identified by a URI.

More formally, an RDF triple can be defined as $t = (s, p, o) \in \mathcal{T}$, where $\mathcal{T}$ is the set of all triples defined as:

$$\mathcal{T} = (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L});$$

with $\mathcal{U}$(for URIs), $\mathcal{B}$(for blank nodes), and $\mathcal{L}$(for literals) being three pairwise disjoint, infinite sets and $uris(t) = \{s, p, o\} \cap \mathcal{U}$.

The linking structure created among a set of RDF triples forms a directed, labelled graph, commonly known as *RDF-graph*, where the predicates are the edges of the graph, while the subjects and objects constitute the nodes [103]. An RDF graph can be formally defined as a directed labelled graph $G = (V, E, L)$, where $V \subseteq (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ is the set of nodes, $E \subseteq (V \cap (\mathcal{U} \cup \mathcal{B})) \times L \times V$ is the set of edges, and $L \subseteq \mathcal{U}$ is the set of edge labels [54].

When working with the RDF data model, triples are usually stored in databases known as triplestores. In addition, there are some serialisation formats, such as Turtle (the most widespread) [8], N-Triples [4], and RDF/XML [5], among others, that allow sharing, exchanging and storing RDF triples. It is important to note that RDF has a built-in vocabulary [6] to describe resources and their relationships, which includes a small set of classes for defining resources of type containers (*rdf:Alt* [2] , *rdf:Bag, rdf:Seq*), lists (*rdf:List*) and properties/predicates (*rdf:Property*). On the other hand, there are some predefined properties, where *rdf:type* is the best known and is used to state that a resource is an instance of a given class. Classes and properties are RDF resources as well. While this small set of classes and properties does not adequately allow for the representation of complex ontologies that generally define a more extensive set of classes, subclasses and properties, it can be extended to create richer vocabularies.

To materialise what has been seen so far, Listing 1.1 shows a set of triples written in Turtle notation, defining some resources and properties in the

[1] A Blank Node represents a resource for which a URI or literal is not given

[2] **rdf:** is just a prefix abbreviation for the <http://www.w3.org/1999/02/22-rdf-syntax-ns#> namespace.

scope of the ontology introduced in Figure 1.1. The URI namespace for the Pasta Ontology is assumed to be: *<http://pastaontology.it/>*. In Listing 1.1, The first line declares the prefix for the RDF vocabulary namespace. The second line defines a property *cookingTime*, and the last two lines describe two resources employing the property *cookingTime* with values 10 and 9, respectively.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://pastaontology.it/cookingTime> rdf:type rdf:Property .
<http://pastaontology.it/Spaghetti_Garofallo>
              <http://pastaontology.it/cookingTime> 10 .
<http://pastaontology.it/Spaghetti_DeCecco>
              <http://pastaontology.it/cookingTime> 9 .
```

Listing 1.1: RDF triples in Turtle notation.

In summary, RDF proposes a flexible data representation model using the concept of triples and URIs, which allows easy integration and exchange of heterogeneous information, as well as an intuitive representation of the interaction between different resources. Nowadays, most knowledge bases are based on the RDF-graph model, so it is common to refer to them as *Knowledge Graphs* [53].

While RDF is a robust framework, it has been necessary to create new languages to expand the RDF vocabulary so that it is possible to correctly describe the complex interactions between entities that arise when addressing real-life applications. The following section describes the details of these languages.

1.3 Ontology languages

This section exposes some details about OWL and RDFS, two well-known languages for ontologies formalisation.

1.3.1 RDF Schema

RDF Schema, or RDFS, is an extensible knowledge representation language that provides mechanisms for describing groups of related resources and the relationships between those resources [6]. RDFS is a semantic extension of the basic RDF vocabulary, introducing a set of classes and properties written in RDF, which can be extended to describe complex ontologies.

Intuitively, in RDFS, the classes are themselves resources that serve to divide other resources into groups, where the members of a class are known as instances of

the class. The most relevant classes introduced in the vocabulary are *rdfs:Resource*, *rdfs:Class*, *rdfs:Literal* and *rdfs:Datatype*. The class *rdfs:Resource* is the class of everything, and all other classes are subclasses of this class. *rdfs:Class* indicates that a particular resource is a class. The resources *rdfs:Literal* and *rdfs:Datatype* are the classes of literal values, such as strings or integers and datatypes, respectively.

On the other hand, RDFS defines the *rdfs:subClassOf* and *rdfs:subPropertyOf* properties that can describe a hierarchy of classes and properties, respectively. Two other important properties that allow for introducing some level of restrictions to the properties created as an instance of *rdf:Property* are *rdfs:range* and *rdfs:domain*. The property range specifies the class or datatype of the object in an RDF triple whose predicate is that property. The domain of a property, instead, declares the class of the subject in an RDF triple whose predicate is that property. Finally, the *rdfs:label* property provides a human-readable version of the resource name. This property plays a central role in the approach described in this thesis (See section 4.3).

Listing 1.2 takes up the ontology defined in Figure 1.1 and proposes a possible definition using RDFS in the Turtle syntax. Note how the property *rdfs:domain* establishes that subjects in triples using *ps:cookingTime* have to be of type *ps:Pasta*; in other words, it is a property to describe *Pasta* instances. The same applies to *rdfs:range*, which says that the values of the property should be an integer number.

```
@prefix ps: <http://pastaontology.it/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
--Classes--
ps:Food rdf:type rdfs:Class .
ps:Wheat rdf:type rdfs:Class .
ps:Pasta_Factory rdf:type rdfs:Class .
ps:Pasta rdf:type rdfs:Class; rdfs:subClassOf ps:Food .
ps:Long rdf:type rdfs:Class; rdfs:subClassOf ps:Pasta .
ps:Short rdf:type rdfs:Class; rdfs:subClassOf ps:Pasta .
ps:Liscia rdf:type rdfs:Class; rdfs:subClassOf ps:Short .
ps:Rigata rdf:type rdfs:Class; rdfs:subClassOf ps:Short .
--Properties--
ps:cookingTime rdf:type rdfs:Property; rdfs:domain ps:Pasta;
   rdfs:range xsd:integer .
ps:drawing rdf:type rdfs:Property; rdfs:domain ps:Pasta;
   rdfs:range xsd:string .
ps:wheat rdf:type rdfs:Property; rdfs:domain ps:Pasta;
   rdfs:range ps:Wheat .
ps:producer rdf:type rdfs:Property; rdfs:domain ps:Pasta;
```

```
   rdfs:range ps:Pasta_Factory .
--Individual Instances--
ps:Wheat_Durum rdf:type ps:Wheat; rdfs:label ''Wheat Durum'' .
ps:Spaghetti_Garofallo ps:wheat ps:Wheat_Durum;
   rdfs:label ''Spaghetti Garofallo'' .
```

Listing 1.2: RDFS-based triples for defining the Pasta Ontology with some individuals

1.3.2 Web Ontology Language (OWL)

The vocabulary introduced by RDFS flexibly allows the creation of ontologies. However, it has several limitations, e.g. the inability to specify cardinalities to the relationships between objects or to describe logical operators (Union, Intersection, and others) between two or more classes, which makes it difficult to express many phenomena that arise when modelling ontologies. For example, assuming that the classes *Woman* and *Parent* are already defined in a family ontology, it might be helpful to establish that a new class *Mother*, is the intersection of *Woman* and *Parent*; that is, any instance of the class is also an instance of all classes in the specified intersection list. In other words, anyone who is a *Mother* is both a *Parent* and a *Woman*, and anyone who is both a *Parent* and a *Woman* is a *Mother*. This kind of description can be helpful for a reasoning engine to infer new knowledge. To address these and many other issues, the W3C introduced the Web Ontology Language (OWL) [1].

The W3C defined OWL as a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things, that is, ontologies [1]. A fundamental characteristic of OWL is that it is based on the expressive and reasoning power of *Description Logic* [15], a family of logic-based knowledge representation formalisms. So, the knowledge described in OWL can be exploited by computer programs, for example, to verify the consistency of that knowledge or to make implicit knowledge explicit.

The most recent version of OWL is known as OWL 2 [2]. Ontologies defined with OWL 2 can be used along with RDF information, and the ontologies themselves are primarily exchanged as RDF documents. The OWL 2 specification includes two sub-languages with different levels of expressiveness, namely OWL 2 DL and OWL 2 Full, where each sub-language is an extension of its predecessor. OWL 2 DL was designed to provide as much expressiveness as possible while retaining the computational completeness and decidability of practical reasoning algorithms. To this end, OWL DL 2 introduces some restrictions on how classes, properties and individuals must be used. The underlying formalism of OWL 2 DL

is the description logic $\mathcal{SROIQ}(D)$ [56]. OWL 2 Full has the same features as OWL 2 DL but loosens the restrictions, making it undecidable. It can be seen as a direct extension of RDFS vocabulary, with a richer set of classes and properties. There also exist three named *"profiles"*[3] for OWL 2, syntactic subsets of OWL 2 DL that are tailored towards different applications, trading expressivity of the language for efficient reasoning; they are OWL 2 EL, OWL 2 QL (Query Language) and OWL 2 RL (Rule Language). As this thesis does not aim to cover all the extensive knowledge about OWL, the reader can refer to the official documentation for helpful information[4].

OWL is the de-facto ontology language for the Semantic Web, and most of the ontologies created today use it in some way. A lot of work has been done around OWL since its inception, and as a result, there are now advanced tools ranging from ontology editors such as Protegè [82] to semantic reasoners such as Pellet [102], Racer Pro [48], and FaCT [115].

1.4 Accessing Knowledge Bases

So far in this chapter, a set of standards and languages have been introduced that allow structuring and describing data to create powerful semantic-centred knowledge bases. However, an important aspect to consider when dealing with such KBs is how to access and retrieve their information. To this aim, W3C created and standardized SPARQL [3].

SPARQL is an SQL-like language to query RDF graphs. As a query language, SPARQL is data-oriented because it only queries the information contained in the models; there is no inference in the query language itself. The syntax and semantics of the language allow the user to query a KB by defining triple patterns that search for a match with the *subject-predicate-object* triples within the graph. This is known as *graph pattern matching*. It is also possible to establish conjunctions and disjunctions among the patterns or to set a pattern as optional [3].

A triple pattern can be formally defined as $tp = (s, p, o) \in \mathcal{TP}$, where $\mathcal{TP}$ is the set of all triple patterns defined as:

$$\mathcal{TP} = (\mathcal{U} \cup \mathcal{B} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \cup \mathcal{V});$$

where $\mathcal{V}$ is an infinite set of variables disjoint from $(\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$. This definition is similar to that of RDF triples (section 1.2) but includes variables that are bound to the corresponding parts of each triple. Variables can be returned to answer a given user query. A set of triples patterns $BGP \subseteq \mathcal{TP}$ is called a *Basic Graph*

[3] `https://www.w3.org/TR/owl2-profiles/`

[4] `https://www.w3.org/TR/owl2-overview/`

Pattern (BGP) and constitutes the most basic kind of query [3]. Several BGPs can be organized in groups delimited with braces to form what is known as *Group Graph Pattern* (GGP). A SPARQL graph pattern matching is defined in terms of combining the results of solving each triple pattern by matching the values of any variables in common. This matching process, unless otherwise specified, can be considered as a conjunction among the triples.

The Listing 1.3 shows the basic structure of a SPARQL query, where the first two lines denote the prefix abbreviations, and the third one is a SELECT query that allows returning the values bound to the variable *?name*. The *WHERE* clause specifies the BGP to match against the data found in the RDF graph. Tokens beginning with a question mark (?) are considered variables, and the braces denote a BGP.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ps: <http://pastaontology.it/>
SELECT ?name WHERE { ?x rdfs:label ?name . ?x ps:drawing ‘‘Bronze’’}
```

Listing 1.3: SPARQL query returning the names of the individuals whose *drawing* is equal to *Bronze*.

The SPARQL syntax allows for the match against literals in the graph. The literals in a BGP are tokens enclosed in either double quotes or single quotes (see Listing 1.3) that can have either reference for the language (using @ after the quotes) or an optional datatype(using ^^after the quotes).

In a query, a graph pattern preceded by the OPTIONAL construct is considered optional, meaning that a query will keep all solutions from other patterns whether or not there is a matching solution in the optional part. A disjunction, instead, can be expressed by using the operator UNION among two graph patterns. Let us assume there is a query including two BGPs, G_1 and G_2, where the matching part is written as $\{G_1\}$ UNION $\{G_2\}$; the query matching process will include both the results of solving G_1 and the results of solving G_2.

A user can employ four different types of queries to retrieve information from an RDF graph. The variations are:

- CONSTRUCT returns an RDF graph constructed by substituting variables.
- DESCRIBE returns an RDF graph describing the resources that were found.
- ASK returns a boolean value indicating whether the query pattern matches or not.
- SELECT returns the variables bound in a query pattern.

The results of a query can be modified by using the modifiers ORDER BY, HAVING, DISTINCT, OFFSET and LIMIT, which have a similar meaning to those of SQL. Another powerful mechanism provided by SPARQL is the ability to apply restrictions to query solutions using the FILTER construct. Filters eliminate those solutions that do not cause an expression to evaluate *true*. A filter expression can contain logical and arithmetic operators, existence operators (EXISTS or NOT EXISTS), regular expressions and many others. The full language allows one to express highly complex queries. More detailed information about SPARQL can be found at [3].

1.5 Reference knowledge bases

Today there are a lot of public knowledge bases related to the most diverse domains, which empower society with valuable sources of knowledge. With the community's support, many of these KBs have become massive databases storing millions of cross-domain resources. These large KBs have been used as a reference for many researchers addressing issues in fields such as Information Retrieval and Question Answering. This thesis targets two knowledge bases, DBpedia [68] and Wikidata [118].

DBpedia is one of the most extensive knowledge bases available today. The DBpedia project focuses on extracting structured content from the information offered by various Wikimedia projects and making it publicly available as an RDF-based knowledge base. The RDF graph currently consists of over a billion triples and over 5 million resources. The core of DBpedia is an ontology created with OWL comprising 768 classes that are described by around 3000 properties. The DBpedia Ontology (2022-03) contains about 4,828,418 instances, including 1,592,912 persons, 967,491 places, 552,115 creative works, 317,867 organizations, 190,369 species and 1,207,664 resources belonging to other categories. The ontology is identified under the URI: <http://dbpedia.org/ontology/>. The knowledge graph can be accessed using standard web browsers, automated crawlers or SPARQL queries.

Wikidata is a free, open, cross-domain knowledge base supporting other Wikimedia projects. As is typical for these kinds of databases, it uses an RDF data model. Data is added by a community of volunteers in two ways, manually or using a software. Unlike DBpedia, Wikidata does not use an RDFS/OWL-based semantic and vocabulary to describe the data but has developed its own [49]. The RDF graph contains over 1.3 billion triples, with over 99 million items. In Wikidata, items can be classes or simple instances because a class can behave as an

instance as well. To describe these items, about 10200 properties are available. The data is available online and can be accessed in various ways, such as SPARQL, web browsers, Rest API and many more[5].

Analysis The knowledge bases described above differ from an ontological point of view. Still, their information overlaps to a certain extent because they have a common source, as DBpedia extracts the data from the Wikimedia projects. This overlap allows for a kind of mapping between the DBpedia and Wikidata individuals so that their information complements each other. To this end, DBpedia uses the predicate *owl:sameAs*, which in the OWL semantics state that two given individuals (subject and object) are equal. This mapping can also be done for classes using the *owl:equivalentClass* predicate, which states that two given classes are equivalent.

From a less positive perspective, these KBs are affected by some issues introduced by the fact that they are community-supported. In the case of DBpedia, there are several cases of predicates overlap. For instance, to refer to the leader of a country, city, or organization of any kind, DBpedia can use predicates such as *dbp:leader*, *dbp:leaderName*, or *dbp:mayor* interchangeably; this may introduce some difficulties, especially for question answering systems. DBpedia also has problems classifying individuals according to their real class; for example, the resource `https://dbpedia.org/resource/Japanese_wolf` is classified as *dbo:Insect* when it should be a *dbo:Mammal*. Wikidata, on the other hand, has a more controlled ontology because the definition of properties and classes goes through an editing process. However, it is affected by misclassification problems[6], as some individuals (items) are used instead of classes to classify a resource. Another common problem affecting Wikidata is the existence of loops in the class hierarchy.

Beyond the problems present in these large KBs, which affect a small part of the information found in them, it can be said that they are powerful tools that allow not only the availability of valuable information but also boost the development of knowledge-based systems.

[5] `https://www.wikidata.org/wiki/Wikidata:Data_access`
[6] `https://www.wikidata.org/wiki/Wikidata:WikiProject_Ontology/Problems`

Chapter 2

Neural Machine Translation

The ability to translate from one language to another has played a fundamental role in the development of humanity, enabling communication between cultures. In this context, machine translation has been a very active field of research over the years, especially Neural Machine Translation (NMT), which has led to high-quality translation models. Motivated by the critical role this task plays in the proposed system, this chapter introduces NMT and describes the neural networks, architectures and techniques that have enabled its development.

2.1 Machine translation

Machine Translation (MT) is the sub-field of the Natural Language Processing task that investigates the use of computers to automatically translate text or speech from one language to another without any human involvement [38]. MT has been an active field of research over time, in which several approaches have been presented to tackle this complex linguistic problem. The research community agrees that those approaches can be divided into three main groups: *Rules-based machine translation (RBMT), Statistical machine translation (SMT) and Neural machine translation (NMT).*

Rules-based machine translation Rule-based approaches [112, 14, 41] were the first systems to reach a level of translation performance that allowed them to be used outside the research environment. These approaches rely on linguistic rules created by field experts that capture the morphological, syntactic and semantic characteristics of both source and target languages, which allow for analysing the sentences in the source language and translating them. In addition to the rules, RBMT requires large bi-lingual dictionaries that allow a mapping between the

words of the two languages. While this approach does not require any training corpora and allows for some control in its output, implementing and maintaining the rules requires a great amount of expert time, especially considering that language is dynamic and constantly evolves over time. Other more evolved methods quickly outperformed this kind of MT approach [41].

Statistical machine translation Statistical models [64, 29, 126] were the most studied MT method before the inception of Neural Machine Translation approaches. This approach was the base of the most popular translators of the time, such as Google Translate and Microsoft Translator, among others. SMT methods rely on statistical models that search for the most probable translation for a given input. These models learn by analysing the statistical relationships between sentences in the input and output language in large bilingual text corpora. SMT seeks to model the probability distribution $p(t|s)$, that a string t in the target language is the translation of a string s in the source language. The standard approach for calculating $p(t|s)$ is to use the Bayes theorem [60] to decompose the distribution into $p(t|s) = p(s|t)p(t)$. The computation of $p(t|s)$ is split into two parts where it is necessary to learn both a *Translation Model* responsible for estimating $p(s|t)$ and a *Language Model* to estimate $p(t)$. A good translation is selected by doing $\arg\max_t p(s|t)p(t)$. Language and Translation models have been the subject of several studies [84, 74, 23, 51], as they are non-trivial processes that have to deal with various intrinsic language problems, such as word order and the difference in sentence size from one language to another. Like RBMT, SMT's general weakness is that it can only translate a phrase if it exists in the reference texts.

Neural machine translation The Neural Machine Translation approaches [110, 122, 16, 123] exploit the development reached by Neural Networks [45] in the last decades to create a model that effectively translates from one language to another. NMT works by recognising patterns in the source sentence to determine a context-based interpretation that can be used to predict the probability of a sequence of words in the target language. These approaches have made rapid progress in recent years and currently constitute state-of-the-art in the MT field. Since this thesis presents an approach that partially relies on NMT, some techniques that allow this type of MT will be discussed hereafter.

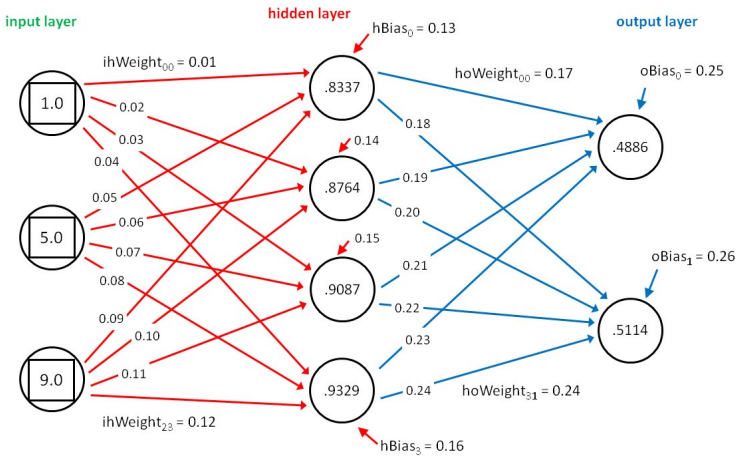


Figure 2.1: A representation of a Neural Network. [111]

2.2 Neural Networks

Artificial Neural Networks (ANN) are algorithms that learn how to map inputs to targets by observing many input-target examples in a training set [33]. The basic structure of an ANN consists of a set of *neurons* organized in layers, where each neuron is connected to all neurons of the next layer, such that the outputs of each layer are the inputs of the next one. This kind of neural network is commonly known as a *Fully Connected Network* [33]. A neural network learns by passing information through the layers in an iterative process called *training*. During training, the information is subjected to several transformations in search of particular patterns that subsequently determine the network's output. The specification of what a layer does to its input data is stored in the layer's weights, which in essence, are a bunch of numbers associated with an existing neuron link in the layer. In this context, learning means finding a set of values for all layers' weights so that the network will correctly map example inputs to their associated targets. Figure 2.1 shows what a Neural Network should look like from a graphical point of view. Three actors mainly guide the learning process of a neural network: the *Neuron Activation Function*, the *Loss Function* and the *Backpropagation process*.

The *Activation Function* calculates a neuron's output, or activation signal, depending on the information from the precedent layer and the weights associated with the links of that neuron. The activation signal is passed to the neurons in the next layer and represents how a particular neuron behaves in response to specific data. The action of communicating the values of neurons from one layer to a successive layer is called *Forward Pass*. There are different activation functions,

and their choice depends on the problem being addressed.

The *Loss Function* or, *Error function* allows knowing if the learning process is on the right way. For that purpose, the function computes a distance score between the final network's output and the training set's true targets (also known as Ground truth) that capture how well the network behaves. The choice depends on the problem being addressed.

Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a network based on the error calculated by the Loss Function in the current iteration. Proper tuning of the weights reduces error rates in successive iterations, making the model reliable by increasing its generalization. The most common way to propagate the error backwards to update the weights is using the Gradient Descent optimization algorithm. With Gradient Descent, it is possible to know in which direction and how significant the change to be applied to each weight is. The weight updates are calculated by mean of:

$$w_{i+1} = w_i - alpha \times J'(w_i) \tag{2.1}$$

where w_i is the current weight, w_{i+1} is the new weight, *alpha* is a hyperparameter called *Learning Rate* that determines how large the change should be, and $J'(w_i)$ is the partial derivative of the Loss Function $J(w)$ with respect to w_i. The training process should finish when the loss error or another desired measure converges to a local/global minimum/maximum, i.e., the network is no longer learning.

2.3 Recurrent Neural Networks (RNN)

Recurrent Neural Networks [96] are a particular type of neural network introduced especially for processing information expressed as time series or involving some sequence in the form $S = (x_1, ..., x_t)$. In data sequences, the order of the elements matters, which creates a dependency relationship between them. Former neural networks are only suitable for dealing with data points which are independent of each other. Another limitation that motivated the introduction of RNNs is that prior neural networks can only process fixed-length inputs, whereas sequences, by nature, can have variable lengths. RNNs have shown a good performance while addressing complex NLP tasks, as *natural language sentences are just semantically-ordered sequences of words* [33].

An RNN works by iterating over the elements of a sequence S and keeping a state h (known as the hidden state) that contains information relative to what was already processed so that the result of processing the input at time t is conditioned by the information at $t-1$ (see Figure 2.2). At each time step t, a new state h_t is computed, combining the input x_t and the state h_{t-1} by means of $h_t = f(h_{t-1}, x_t)$,

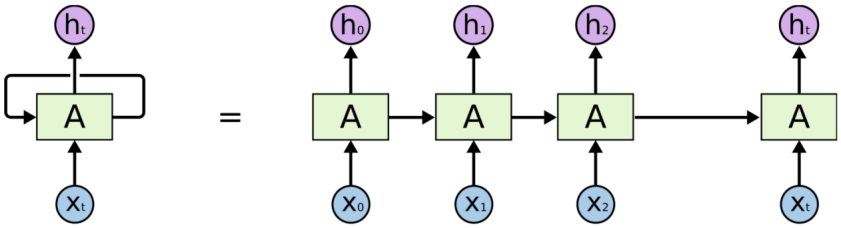


Figure 2.2: A Recurrent Neural Network cell. [85]

where f is a non-linear function. In a standard RNN implementation, the function f is implemented as:

$$h_t = \tanh(Ux_t + Wh_{t-1} + b) \tag{2.2}$$

where U are weights associated with inputs, W are weights associated with hidden units[1], and b is a bias vector. On the other hand, tanh is the activation function, which can also be another function depending on the case to be addressed. Note that the input to the tanh function in equation 2.2 is not a real value, but a vector, meaning the function is applied as a component-wise operation. The looping mechanism used by an RNN to process the data makes it possible to feed the network with variable-length sequences.

RNNs can be classified into four types depending on the sizes of inputs and outputs employed to solve a given problem. The types are: *one-to-one, one-to-many, many-to-one, and many-to-many*. The last type of network, also known as *sequence-to-sequence*, allows addressing tasks like *Text Summarization* and *Machine Translation*, where the input is a sequence, and the output should be a sequence as well.

2.3.1 Long Short-term memory

While RNNs have been demonstrated to perform well in many NLP tasks, they suffer from a phenomenon known as the *Vanishing Gradient Descent* problem [52, 19] that negatively impacts the proper training of the network. Vanishing Gradient Descent affects networks with many layers using *sigmoid* or *tanh* activation func-

[1]The number of units determines the size of the hidden state vector h.

tions. As the backpropagation method progresses backwards from the output layer to the input layer, the gradients frequently become less and smaller until they approach zero, leaving the weights of the initial layer almost unchanged. This problem is because the derivatives of the activation functions used to calculate the gradient are very close to zero. In an RNN, this problem is particularly relevant when working with long sequences since the longer the sequences, the more backward steps the backpropagation process has to perform. This problem causes particular difficulty in learning long-term dependencies, which is also caused by the fact that the memory (state) is overwritten at each step, allowing only short-term dependencies to be captured.

To address this problem, Hochreiter et al. [52] proposed a new type of RNN called Long Short-term memory (LSTM). LSTM adds a *memory cell* that is a way of transporting information through many time steps. Imagine a conveyor belt running parallel to the sequence being processed. Information from the sequence can jump onto the conveyor belt at any point, be transported to a later timestep, and jump off, intact, when needed. Essentially, an LSTM saves information for later, thus preventing older signals from gradually vanishing during processing [33]. To this end, LSTMs employ a mechanism called *Gates* when computing the hidden states. The gating can regulate the flow of information and decide what information is important to keep or throw away. The data processing is done by:

$$
\begin{aligned}
i &= \sigma(x_t U^i + h_{t-1} W^i) & f &= \sigma(x_t U^f + h_{t-1} W^f) \\
o &= \sigma(x_t U^o + h_{t-1} W^o) & g &= \tanh(x_t U^g + h_{t-1} W^g) \\
c_t &= c_{t-1} \circ f + g \circ i & h_t &= \tanh(c_t) \circ o
\end{aligned}
$$

where the input i, forget f, and output o represent gates that are squashed by the Sigmoid into vectors of values between 0 and 1. Multiplying the vectors determines how much of the other vectors to let into the current input state. g is a candidate memory state that is computed based on the current input and the previous hidden state. c_t is the new memory cell at time t, which is a combination of the previous memory c_{t-1} multiplied by the forget gate, which incorporates information about the current input depending on the input gate i and g.

Finally, based on the output gate o and the new memory state c_t, the network decides which information the new hidden state h_t should carry. It is important to note that in the previous equations, the operator "$\circ$" represents a Hadamard product, which performs a component-wise product of vectors/matrices. The memory cell design also allows the gradients to travel back many steps during backpropagation without vanishing too much, alleviating the Vanishing Gradient Descent problem. Figure 2.3 shows how the information flows through an LSTM cell.

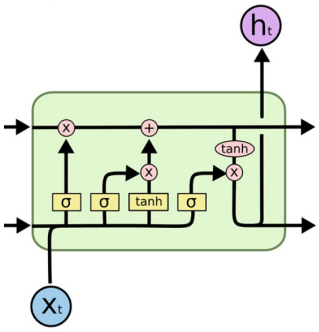


Figure 2.3: An LSTM cell mechanism. [85]

2.3.2 Bidirectional RNNs

The RNNs, in their original form, process the sequences in just one direction. However, there can be situations where information from the future may be important in the present. For example, predicting a word to be included in a sentence might require looking into the future, i.e., a word in a sentence could depend on a future event. A Bidirectional RNN (BRNN) addresses this problem by processing the sentence in both forward and backward directions. Usually, the BRNNs consist of two RNNs, in any variant (simple RNN, LSTM, or GRU[32]), each of which processes the sequence in one direction. The outputs of the RNNs are combined either by concatenation, multiplication, average, or sum.

2.4 Word Vectors Representations

Neural networks work by doing tons of mathematical calculations, which means that in order to use them, the input must be represented as numbers. Therefore, to work with natural language, it is essential to look for mechanisms to represent words as numbers or numerical vectors that, in turn, allow the representation of sentences as a sequence of numbers or vectors that can be fed into the network.

The first step in this process would be to break down the sentences into simple units known as *tokens*. The operation of dividing the text into these units is known as *Tokenization* [33] and can be done by: (*i*) Segmenting text into words and symbols; (*ii*) Segmenting text into characters and symbols; (*iii*) Extracting n-grams of words or characters. N-grams are overlapping groups of multiple consecutive words or characters [33]. The process of associating each token with a numeric vector can be done in different ways; among the best known are: *One-Hot Encoding* and *Word Embeddings*.

One-Hot Encoding It is one of the most common and straightforward techniques for converting a token into a vector. It consists of associating an integer index to each token and then converting this index i into a binary vector of size N, where N represents the number of tokens in the vocabulary[2]. The vector is all zeros except the *ith* position, which is 1. The problem with this representation is that the size of the vectors tends to grow as the size of the vocabulary increases, requiring huge memory for storing and processing them; in addition, the vast majority of the values are zero. Also, this token representation lacks any meaning, so the semantic relationship between the words is lost.

Word embeddings are a type of word representation that allows words with similar meanings to have a similar representation. With this technique, it is possible to capture the context of a word in a document, semantic and syntactic similarity, and relationship with other words. The characteristic that makes it a variant superior to *One-Hot Encoding* is that the vectors obtained have low dimensionality, and what each vector contains is learned from the data. Word embeddings are intended to map human language into a geometric space. For example, in a reasonable embedding space, synonyms are expected to be associated with similar word vectors, and in general, the geometric distance between two vectors is expected to be related to the semantic distance between the associated words [45].

The embedding representation can be learned jointly with the main task being addressed, such as document classification or machine translation. In this case, the vectors are initialized randomly and then adjusted in the same way that the weights of a neural network are learned.

Another way to calculate the embeddings is by using already pre-trained models like *Word2vec* [79], *GloVe* [89] or *fastText* [21], which can be used to obtain the word vectors directly without learning them in the main task, thus saving time and resources. Word2vec is a method introduced by Mikolov et al. [79] to efficiently create word embeddings using a two-layer neural network. Word2vec can be implemented using two different learning models, *Continuous Bag-of-Words (CBOW)* and *Continuous Skip-Gram*. The CBOW model learns the embedding by predicting the current word based on surrounding context words. The Continuous Skip-Gram model learns by predicting the surrounding words given a current word. GloVe is an unsupervised learning algorithm developed by researchers at Stanford University aiming to generate word embeddings by training a count-based model on aggregated global word-word co-occurrence statistics from a corpus [89]. fastText is basically an extension of Word2Vec, but instead of learning vectors

[2]Collection of tokens available in a given natural language corpus.

for complete words directly, it represents each word as an n-gram of characters. After each word is represented as n-grams, a Skip-Gram model is used to learn the embeddings. FastText works very well with rare words and can calculate word embeddings for those words that are not part of the vocabulary used to train the model. Unseen words are decomposed into n-grams to obtain partial representations, and then these are aggregated to obtain a single vector.

Although very powerful, the approaches described above do not consider that a word's meaning depends on its context, so a word will always have the same vector representation regardless of the sentence context. *Contextualised Word Embeddings* [90, 35] solve this problem by learning word representations with advanced neural network-based models using LSTMs or Transformers (see section 2.5.2) [117]. This way, a word will have a different vector representation depending on the context. For example, the vector representation for the word *apple* in the sentence *"I am eating an apple"* will be different from that of the sentence *"I like Apple MacBooks"* since, in each case, the same word has a different meaning. There are now huge pre-trained models such as *ELMo* [90], *BERT* [35] and *GPT-2* [94] that allow effortless incorporation of contextualised word embeddings into the NLP tasks. Contextualised Word Embeddings constitute the state-of-the-art in the field.

Although all the approaches mentioned in this section provide ready-to-use models pre-trained on large corpora such as Wikipedia, Google News or Twitter tweets, it is also possible to train them from scratch or refine them using other training sets. Another aspect to consider is that even if contextualised word embeddings are state-of-the-art, selecting a method to compute the word vector representation often depends on the addressed task, e.g. sometimes it is convenient to learn embeddings as part of the main task or sometimes the choice of one method or another does not impact in the final performance of an NLP model [36, 119, 91, 44].

2.5 Sequence-to-sequence for MT

Sequence-to-sequence learning (Seq2Seq) is the task of training models for mapping sequences from one domain to sequences in another domain [110]. Seq2Seq models are the de facto standard for solving the problem of machine translation, where a sentence (*sequence of semantically-ordered words*) in a source language needs to be mapped to a sentence in a target language. These models have also been successfully used in other NLP tasks such as Question Answering and Text Summarization. One of the first studies that proposed using neural network-based Seq2Seq models in the field of MT was the one by Sutskever et al. [110], which used

an end-to-end LSTM-based model to address the English-to-French translation problem, demonstrating the superiority of this type of approach over SMT-based systems.

The most common implementation of Seq2Seq is using an *Encoder-Decoder* architecture. In this model, the Encoder has to learn to compress the input into a fixed-length latent space v, also known as a context vector, that captures the main features of the input. When working with natural language, those features correspond to the semantic and syntactic properties of a given input sentence. On the other hand, the Decoder must learn to unfold the context vector into a new sequence. Intuitively, in the field of NLP, the Encoder and Decoder are neural networks suited to sequence processing, such as RNNs (including variants) or Transformers.

When training a model for MT, the Encoder and Decoder are trained together in order to estimate the conditional probability $p(y_1, ..., y_{T'}|x_1, ..., x_T)$, where $(x_1, ..., x_T)$ is the input sequence, and $(y_1, ..., y_{T'})$ is the target language sequence. Note that the length T may differ from T'. To calculate the probability, first, the Encoder network calculates a context vector v of the input sequence, and then the Decoder network calculates the probability of the output sequence by considering the context vector v [110]. Mathematically, the above can be defined as:

$$p(y_1, ..., y_{T'}|x_1, ..., x_T) = \prod_{t=1}^{T'} p(y_t|v, y_1, ..., y_{t-1}) \tag{2.3}$$

where each probability $p(y_t|v, y_1, ..., y_{t-1})$ is represented as a Softmax function over all the words in the output vocabulary. Note that the probability that an element y belongs to position t in the output sequence also depends on which are the previous elements in the sequence.

To illustrate the process more practically, one can assume an RNN-based model and an output vocabulary of size N. The input sentence is processed by an RNN-based encoder whose last hidden state, $h_t^{(e)}$ with $t = T$, represents the context vector v. Subsequently, the initial state of an RNN-based decoder is initialised with the context vector v, i.e. $h_0^{(d)} = v$. From this point on, the decoder starts an iterative process where at each time step t, the previous output y_{t-1} is taken as input to produce a new hidden state $h_t^{(d)}$ using equation 2.2, which in turn allows obtaining a new output y_t. For calculating the output y_t, it is common to use a fully connected network with N neurons and a Softmax activation function to calculate a probability vector $pr_t = (pr_1, ..., pr_n) \in [0, 1]^N$ indicating the probability of the *n-th* token in the output vocabulary to be the output y_t. The vector pr_t is

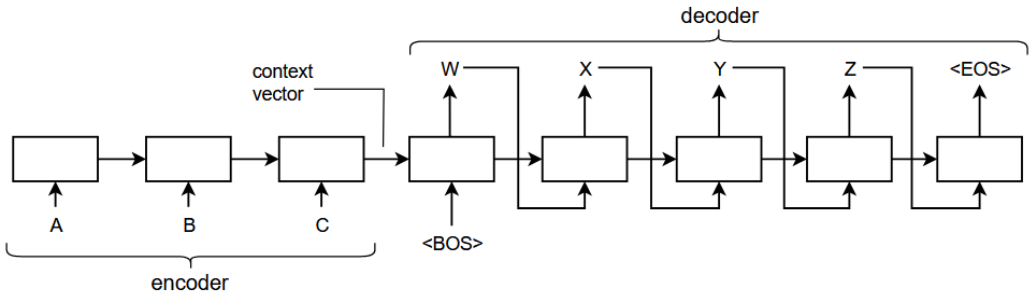


Figure 2.4: Sequence to sequence model by Sutskever et al. [110]

calculated by:

$$pr_t = Softmax(Wh_t^{(d)} + b) \tag{2.4}$$

where W are trainable weights, and b is a bias vector. From equation 2.4, it can be deduced that the probability $p(y_t|v, y_1, ..., y_{t-1})$ is equal to the maximum probability in the vector pr_t. Note that the hidden state $h_t^{(d)}$ carries information about v and $(y_1, ..., y_{t-1})$. Figure 2.4 illustrates the iterative process and shows how the output y_t serves as input for the next step *t+1*, thus letting the decoder incorporate information about the previously generated token. The $\langle BOS \rangle$ and $\langle EOS \rangle$ tokens are introduced to indicate the beginning and end of the sequence, respectively, with $\langle BOS \rangle$ always being the initial decoder input, i.e. y_0. The decoding process ends when the decoder generates the $\langle EOS \rangle$ token. To conclude, it is worth mentioning that the Seq2seq models are commonly trained using Teacher Forcing [45], a procedure in which, during training, the model receives the ground truth output y_t as input at time $t + 1$.

Because of its importance for a correct understanding, equation 2.5 defines the Softmax function, where $z = (z_1, ..., z_K) \in \mathbb{R}^K$ represents an input vector of size K, and e^{z_i} is the standard exponential function. The Softmax transformation forces the values in z to be positive and sum to 1, making them interpretable as a probability distribution. The Softmax function is commonly used as the activation function in the output layer of neural network models that predict a multinomial probability distribution.

$$Softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.5}$$

2.5.1 Attention mechanisms

Several studies [31, 16, 133] have shown that the performance of RNN-based Seq2Seq models in the context of MT is negatively affected as the length of the input sequence increases. This is mainly due to two problems, i.e., the Decoder depends on the limited information about the input summarised in the fixed-length context vector, and the longer the sequence, the more steps the backpropagation must perform, leading to the Vanishing Gradient problem (see Section 2.3.1). The last problem is partially alleviated by LSTM networks but not completely solved.

To address these problems, Bahdanau et al. [16] proposed a mechanism called *Attention*, which allows the Decoder, at each time step, to use relevant information directly from the input sequence to predict the current element in the output sequence, thus allowing the context to be preserved from beginning to end. The main idea behind the Attention mechanism is to retain all the hidden states of the input sequence generated by the Encoder at each time step and then create a weighted combination between each time step of the decoder output and those encoder states. This means that for each output the Decoder makes, it has access to the entire input sequence and can selectively choose specific elements (*those with the highest weights*) of that sequence to produce the output.

Bahdanau Attention works by performing three fundamental steps. The first step is calculating the alignment scores between the hidden states of the encoder, h_i, and the previous decoder output, s_{t-1}. The resulting alignment score vector indicates how well the input elements align with the current output at position t. The alignment model is represented by a function $score(s_{t-1}, h_i)$ implemented as:

$$e_{ti} = score(s_{t-1}, h_i) = v_a^T \tanh(W_a[s_{t-1}; h_i]) \tag{2.6}$$

where v_a^T and W_a are attention parameters that are learned during training. The second step is to calculate the weight α_{ti} of each alignment score by applying a Softmax operation to the previously computed vector:

$$\alpha_{ti} = Softmax(e_{ti}) \tag{2.7}$$

The Softmax function causes the values in the vector to be a probability distribution, representing the weightage each input holds at time step t. Finally, a context vector c_t is obtained, computed as the weighted sum of all weights in the previous step and the encoder's hidden states. Note that c_t is not fixed-length, as its size depends on the input sequence length. The context vector c_t is calculated as follows:

$$c_t = \sum_{i=1}^{T} \alpha_{ti} h_i \tag{2.8}$$

Due to the Softmax function in the previous step, if the score of a specific input element is closer to 1, its effect and influence on the decoder output are amplified. The c_t vector is concatenated with the previous Decoder output and then fed into the Decoder RNN cell to produce a new output. Figure 2.5 (a) summarises what had been explained before.

On the other hand, Luong et al. proposed two attention mechanisms inspired by the previous one. Basically, the idea focuses on solving the same problem but proposes different architectures than Bahnadau's. The two mechanisms are known as *Global Attention* and *Local Attention*. The main differences with Bahnadau's are the way of calculating the alignment score and that the alignment is calculated w.r.t. to the new decoder hidden state and not to the previous one, i.e. the attention mechanism is introduced after the RNN Decoder has produced a new state.

The *Luong General Attention* works by first leaving the RNN Decoder to calculate the new hidden state s_t based on the previous decoder's hidden state and the previous decoder's output. The next step is to calculate the alignment scores between s_t and all the encoder's hidden states h_i. There are three ways of calculating the alignment scores:

$$score(s_t, h_i) = v_a^T \tanh(W_a[s_t; h_i]) \tag{2.9}$$

$$score(s_t, h_i) = s_t^T W_a h_i \tag{2.10}$$

$$score(s_t, h_i) = s_t^T h_i \tag{2.11}$$

where v_a^T and W_a are attention parameters that are learned during training. These three mechanisms are known as *Concat*, *General*, and *Dot*, respectively. *Concat* is slightly similar to how alignment scores are calculated in Bahdanau Attention. Subsequently, the alignment scores are Softmaxed and multiplied by the hidden

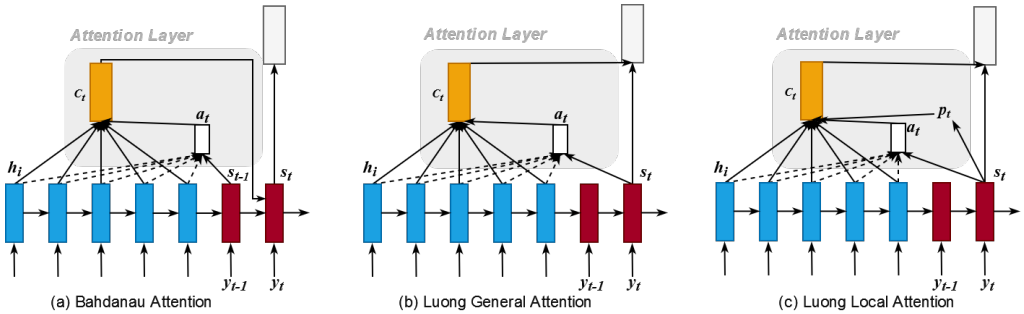


Figure 2.5: Bahdanau (a), Luong General (b) and Local(c) attention mechanisms illustrated. The encoder is in blue, and the Decoder is in red.

states of the Encoder to calculate the context vector c_t. Finally, the context vector is concatenated with the hidden state s_t calculated by the Decoder and passed through the fully connected network to obtain the new output. Figure 2.5 (b) captures the main differences against Bahdanau Attention.

Local Attention (see Figure 2.5 (c)) works the same way as above but realizes that attending to all words on the source side for each target word is computationally expensive, so it seeks to focus on only a subset of the input per target word. The local attentional model generates the context vector c_t by computing a weighted average over the set of source hidden states h_i, within a window $[p_t - D, p_t + D]$. The aligned position p_t is calculated for each target word at time t, and the value of D is selected empirically. Luong et al. propose two approaches to computing the position p_t. The first is *Monotonic alignment*, where the source and target sentences are assumed to be monotonically aligned, setting $pt = t$. The second one is *Predictive alignment*, where the model predicts an aligned position as follows:

$$p_t = S \times sigmoid(v_p^T \tanh(W_p, s_t)) \tag{2.12}$$

where W_p and v_p^T are trainable weights, and S is the source sentence length. As a result, $p_t \in [0, S]$. To favour alignment points near p_t, Luong et al. place a Gaussian distribution centred around p_t.

Attention mechanisms have revolutionised the way NLP models are created and are now a standard element in most state-of-the-art NLP models. Studies [25, 75, 87] have found that both mechanisms yield quite similar results when dealing with the same task, with Luong Attention requiring less computational power.

2.5.2 Transformers

Currently, the most advanced models for NMT [70, 95] build on top of the *Transformer* architecture proposed by Vaswani et al. [117]. The Transformer was the first NMT model to rely entirely on the *Self-Attention* mechanism to compute the input and output representations without using sequence-aligned RNNs [117]. Transformers are the current state-of-the-art in NMT and many other tasks in the field of NLP.

Self-Attention is an attention mechanism relating different positions (words) of a single sequence (sentence) that allows for calculating contextualised representation of the sequence elements. Self-Attention works by comparing every word to every other word in the sentence, including itself, and reweighing the word embeddings of each word to include contextual relevance. It takes in N word embeddings without context and returns N word embeddings with contextual information. This mechanism follows the intuition of the previous attention mechanisms and

works by mapping a *query* and a set of $key - value$ pairs to an *output*, where the query, keys, values, and output are all vectors [117].

The alignment scores are calculated by matching the specific query q under consideration to all key vectors k_i through a dot-product, following a similar approach as Luong et al. Subsequently, the resulting scores are weighted by using a Softmax function. The weighted vector captures the attention a word (represented by q) should pay to each other in the sentence. The final vector representation (output) is calculated as a weighted sum of the value vectors, v_{k_i}, where each value vector is paired with a corresponding key [117]. The entire process is done simultaneously for all the sequence words by packeting all the queries, keys, and values into respective matrixes and then using vectorisation and linear algebra.

The calculations are done by mean of:

$$attention(Q, K, V) = softmax(QK^T)V \tag{2.13}$$

with Q, K, and V the matrixes for the queries, keys and values, respectively.

Vaswani et al. also propose a *Multi-Head Attention* mechanism. The Multi-Head Attention mechanism linearly projects the queries, keys and values h times, each time using a different learned projection. The Self-Attention mechanism is then applied to each of these h projections in parallel to produce h outputs, which in turn are concatenated and projected again to produce a final result. The idea behind Multi-Head Attention is to allow the attention function to extract information from different representation subspaces, which would otherwise not be possible with a single attention head. The Multi-Head Attention function is represented as follows:

$$multihead(Q, K, V) = concat(head_1, ..., head_h)W^O \tag{2.14}$$

$$head_i(Q, K, V) = attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.15}$$

where QW_i^Q, KW_i^K, VW_i^V, and W^O are weight matrices learned during training. As seen in equation 2.14, the final vector representations are calculated by multiplicating the concatenated output with the weight matrix W^O.

Based on this attention mechanism, the original paper [117] proposes an NMT Encoder-Decoder model, where the encoding mechanism is a stack of encoders, and the decoding component is a stack of decoders of the same length (see Figure 2.6). At a high level, the encoders in the stack are basically composed of a self-attention layer where each output flows through a single fully-connected neural network; the final results are then passed to the next encoder in the stack. The decoder, instead, has both those layers, but between them, there is an encoder-decoder attention layer that helps the decoder focus on relevant parts of the input

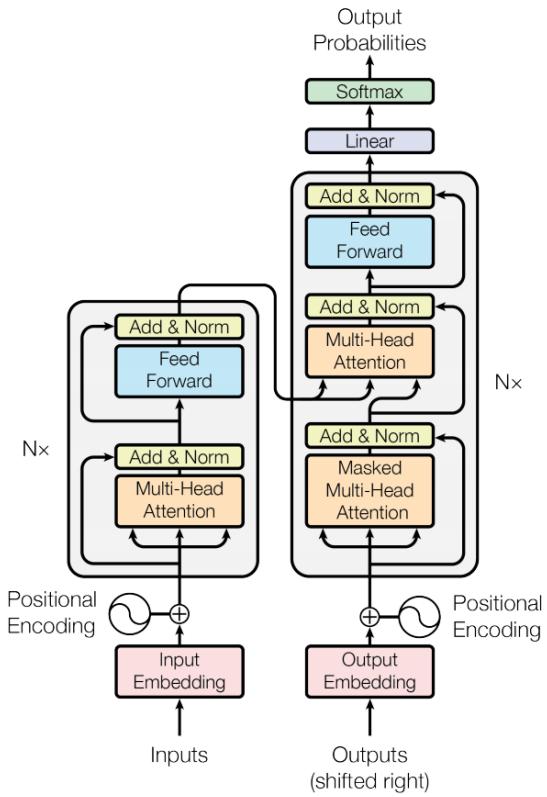


Figure 2.6: Transformer architecture by Vaswani et al. [117]

sentence. The Encoder-Decoder attention layer works similarly to Bahdanau Attention. The Decoder results are passed through a fully connected layer and then to a Softmax layer that converts the values into a probability distribution over the output vocabulary. The probability at index i indicates how likely the ith word is to be the output word at time step t. Figure 2.6 shows graphically the transformer architecture proposed by Vaswani et al.

Transformers paved the way for creating strong models capable of obtaining state-of-the-art results in almost all NLP tasks. Pre-trained models such as BERT [35], BART [70], T5 [95], GPT-2 [94], and GPT-3 [26] have demonstrated high performance in Machine Translation, Document Summarisation and Named Entity Recognition tasks, among others. Apart from using Self-Attention to represent input and output sequences, another advantage over RNN-based models is that Transformer computations can be parallelised, as data is not processed sequentially, significantly improving training times.

Chapter 3

The problem of KBQA

Question Answering (QA) is a well-known discipline of computer science usually referred to in the Information Retrieval (IR) and Natural Language Processing fields. This discipline focuses on the study and creation of automated systems to answer questions posed in natural language by humans [61]. The history of QA systems dates back to the 1960s with the creation of the BASEBALL [46] and LUNA [120] systems. These systems were able to accurately answer questions related to MLB Baseball statistics and information about the geological characteristics of rocks coming from the moon. Many other systems were developed after that, exploiting the development reached by computers, storage capabilities and algorithms for NLP and IR. One of the most impactful systems was IBM Watson [40], which in 2011 beat the human competitors on the TV show *Jeopardy*. The technologies used to develop it led to an effective advancement of the techniques of QA.

QA systems can be classified as either *Open-domain Question Answering or Closed-domain Question Answering*, depending on the domain of the questions they try to answer. Systems belonging to the former type are not restricted to any specific domain, and the search for answers is usually performed on extensive collections of documents that may come from the web or by querying big cross-domain knowledge bases. On the other hand, the second type comprises systems that answer questions in a specific domain, such as medicine, weather, sports and others. These systems typically use information from a small number of documents or specific-domain knowledge bases and can answer a limited number of questions, leading to high-quality answers. BASEBALL and LUNA are practical examples of such systems. Integrating different Closed-domain QA systems can lead to Open-domain QA systems.

Most QA systems focus on factoid questions, which can be answered in terms

of a short text representing a fact. The factoid-type questions commonly begin with the *wh-word*, e.g., *"Who is the president of Italy?"* There are other types of questions that condition the answer that a QA system should give, such as *List type questions, Confirmation questions, Causal questions, Hypothetical questions and Complex questions*. The List type questions need a list of facts or entities as answers, e.g. *"List names of movies in 2022"*. Confirmation questions need answers in the form of yes or no, e.g. *Is Michelle the wife of Obama?* Causal questions answers are not facts like factoid-type questions but descriptions or reasons about a fact. This type of question usually starts with the adverbs *why* and *how*. Hypothetical questions request information associated with any hypothetical event and not specific answers to these questions. The reliability and accuracy of these questions are low and depend upon users and context. Finally, the Complex questions are more challenging to answer and typically require a large body of knowledge and inference techniques. These are the questions whose answers need to be obtained from pieces of information scattered in multiple documents. Complex questions usually involve multi-hop reasoning, restricted relationships or numerical operations.

Regardless of the domain or question types that QA systems must answer, they are implemented by following two fundamental paradigms: *Text-based Question Answering and Knowledge Base Question Answering* (KBQA). Systems of the first paradigm rely on a vast amount of information found in the form of text on the Web or in specific document collections. Given a user question, the QA system uses techniques similar to those of IR to extract passages directly from the documents, guided by the text of the input question. The second paradigm is based on a semantic representation of the question in the form of a formal query on a knowledge base. In the following, the paradigms will be described in some detail by focusing more on KBQA, which is the main objective of this thesis.

3.1 Text-based Question Answering

The Text-based QA systems answer user questions by searching for answers in text segments taken from the Web or other document collections. It is common in such systems to use an architecture including three main phases: Question processing, Document and passage retrieval, and Answer extraction [61].

The Question processing phase allows valuable information to be extracted from the question and consists of two main steps: Query formulation and Answer type detection. In the query formulation step, the task is to create a list of keywords for retrieving relevant documents from which to look for the answer by employing an IR engine. The type of sources available influences this step, and in

the case of Web search, web search engines can sometimes be used directly. In the Answer type detection step, a classifier is used to classify questions based on the expected answer type. For example, the question *"Who is the president of Italy?"* has a PERSON-like answer. In this way, once the entity type of the expected answer is known, it is possible to analyse only the text segments that concern such entity type. The Document and passage retrieval phase builds on the query generated in the previous phase to retrieve a set of candidate documents using an IR engine. The search space is further narrowed down by analysing the most promising documents and extracting the text passages (sentences, paragraphs, etc.) that may contain the answer. Typically this phase follows a supervised approach where the features are relevant features of the individual steps through a multi-level match with the input question. Finally, the Answer Extraction phase involves an extractive process that consists of extracting a specific part of the text selected in the previous phase. This task is commonly modelled by span labelling: given a passage, identifying the span of text which constitutes an answer. Two common answer extraction techniques are answer-type pattern extraction [88] and N-gram tiling [24, 71], but currently, state-of-the-art results are achieved with neural network-based approaches. Transformer-based language models such as BERT, RoBERTa [73], T5 [95] and BART [70] have significantly boosted answer extraction approaches [98, 100], opening a new door in this direction.

3.2 Knowledge Base Question Answering

Beyond the large volumes of data offered in terms of text on the web or collections of documents related to a particular domain, information can also be stored in more structured forms such as relational databases or more semantically meaningful structures such as knowledge bases. In this context, the problem of answering questions in natural language by querying such information sources is known as Knowledge Base Question Answering (KBQA). Systems attempting to solve the KBQA problem typically employ two main approaches: *Information retrieval-based (IR-based) and Semantic Parsing-based (SP-based) methods.* The IR-based methods [108, 124, 81, 127] work by extracting a question-specific graph from the KB that delivers comprehensive information related to the subject and relations in the question and generate the final answers by reasoning on that extracted graph. On the other hand, SP-based methods typically map questions to a symbolic logic form and then execute it against the KB to obtain the final answers. The logical form of the question is either in the form of a SQL or SPARQL query or can easily be converted into one. SP-based methods have greatly benefited from the development of Seq2Seq and usually outperform IR-based methods in solving the

KBQA problem [67]. SP-based KBQA systems typically work by performing a process that includes Question Understanding, Logical Parsing and KB Execution phases.

Many KBQA systems propose to address the Question Understanding problem using syntactic parsers, which allow the representation of a question either by obtaining a dependency path [12, 11] between question elements or by obtaining an Abstract Meaning Representation (AMR) [62] graph. Other approaches also propose a Skeleton-based [109] parser to decompose the question into simpler structures represented as a directed tree. The main idea behind all these representations is to provide a better alignment between question constituents and logic form elements. The question parsers are often implemented as Seq2Seq models trained using dedicated datasets. The encoded question serves as starting point in the Logical Parsing phase to create uninstantiated logical forms that are subsequently instantiated and validated by doing some semantic alignment to the KB. As a result, an executable logical form containing the entities and predicates extracted from the KB is obtained. Note that the encoded question obtained in the first phase can often be considered a logical form. An alternate [77, 135, 131, 28] family of SP-based approaches treats the KBQA problem as that of generating a set of query paths on the KB and ranking them w.r.t. the given question. These query graph ranking approaches work by first constructing a list of candidate expressions by grounding on a query graph against the KB and then selecting the most appropriate one considering the question's lexical and semantic structure.

Most recently, the creation of big sets of supervised data [83, 9, 114, 50] consisting of natural language questions paired with their corresponding SPARQL queries has allowed the creation of approaches [37, 104, 128] to tackle the KBQA problem by exploiting neural machine translation techniques. These NMT-based systems allow for translating the question directly into an executable SPARQL query.

Finally, in the KB Execution phase, the parsed logical form is executed against the knowledge base to retrieve the answers to the given question. The execution is usually performed by means of an existing executor that can vary from KB to KB. Optionally the answers could be processed and elaborated to be presented to the end users.

3.2.1 Formal problem/task definition

In this thesis, the KBQA problem is seen as the following Natural Language Processing task: Given an RDF knowledge base O and a question Q_{nat} in natural language (to be answered using O), translate Q into a SPARQL query $S_{Q_{nat}}$ such that the answer to Q_{nat} is obtained by running $S_{Q_{nat}}$ on O.

One starts from a training set containing a number of pairs $\langle Q_{nat}, G_{Q_{nat}} \rangle$, where Q_{nat} is a natural language question, and $G_{Q_{nat}}$ is a SPARQL query, called the *gold query*. The gold query is a SPARQL query that models (i.e., allows to retrieve from O) the answers to Q_{nat}. The training set has to be used to learn how to answer questions posed in natural language using O so that, given a question in natural language Q_{nat}, the QA system can generate a query $S'_{Q_{nat}}$ such that $answers(S'_{Q_{nat}}) = answers(G_{Q_{nat}})$ for Q_{nat}. This problem is approached as a Machine Translation task, that is, compute $S'_{Q_{nat}}$ as $S'_{Q_{nat}} = Translate(Q_{nat})$, where $Translate$ is the translation implemented by the proposed QA system.

For example, given the Q_{nat} question *"Which instruments does Cat Stevens play?"*, and the $G_{Q_{nat}}$ query

```
SELECT DISTINCT ?a WHERE { dbr:Cat_Stevens dbp:instrument ?a  }
```

a trained model implementing $Translate$ can produce the following $S'_{Q_{nat}}$ query:

```
SELECT DISTINCT ?a WHERE { ?v dbp:instrument ?a .
?v rdfs:label "Cat Stevens"@en }
```

where both the gold and translated queries are syntactically different but hold the condition $answers(S'_{Q_{nat}}) = answers(G_{Q_{nat}})$ on O.

The out-of-vocabulary words problem Most of the solutions proposed up to now to convert from natural language to SPARQL make use of supervised learning techniques based on deep neural networks. For obvious reasons (e.g., a large number of entities in the knowledge bases, frequent updates), all available training datasets comprise only a part of the input/output vocabulary, generating the problem known as *Out-Of-Vocabulary* (OOV) words. Systems affected by the OOV word problem have difficulties dealing with words not seen during the training phase because they did not learn how to map those words to the output vocabulary. Typically, the input vocabulary corresponds to the language used to pose the questions, e.g. English. The output vocabulary, instead, includes the tokens defined by the SPARQL syntax and the names of entities, classes and predicates in the KB. The OOV is a well-known problem affecting many vocabulary-dependant NLP tasks, such as neural machine translation.

For example, let us assume there is a training set containing the *"Abraham Lincoln"* words mapped correctly to the output vocabulary and a KBQA system trained on it. In an attempt to translate the question *"When was Abraham Lincoln born?"*; the system will be able to identify the right KB resource (dbr:Abraham_Lincoln for DBpedia), but on the other hand, the system will fail to translate a question using the same question pattern, but changing *"Abraham*

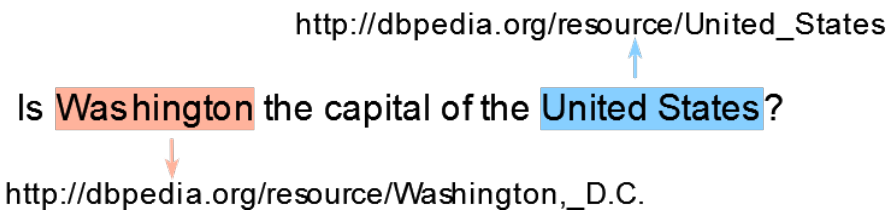


Figure 3.1: Example of Entity Linking on DBpedia knowledge base.

Lincoln" by another individual not present in the vocabulary, let say *"Barack Obama"*. To reduce the impact of the OOV words and improve the system performance, Chapter 4 of this thesis introduces some remedies, including an innovative system architecture and a new format to represent NL to SPARQL datasets.

3.2.2 Entity Linking task

A critical task in at least all KBQA systems is *Entity Linking* (EL). In the field of natural language processing, Entity Linking, also known as *Named-Entity Disambiguation*, is the task of linking a reference (named entity) within a text unit to the most suitable entity in a reference knowledge base such as Wikipedia, DBpedia or Wikidata. For example, for the question *"Is Washington the capital of the United States?"* the idea is to determine that Washington is the USA city and not the US founding father George Washington or another entity. The United States, on the other hand, should be linked to the corresponding entity of type country in the KB. Figure 3.1 illustrates an example of EL with DBpedia as the knowledge base.

Identifying and linking the entities in a question is crucial to building the question-specific graph in IR-based systems since starting from the KB node (or nodes) representing the entity (or entities), it is possible to initialize and expand a graph. EL is also essential in SP-based systems to create the candidate query paths in the query graph ranking approaches and to instantiate the logical forms obtained during the Logic Parsing phase, whether it is a SPARQL query or another type of logical form. Linking the entities to the KB also allows for identifying and validating the possible predicates that will be part of the query.

Most EL approaches follow a pipeline that starts by identifying the named entities in the question; this process is known as Named Entity Recognition. A named entity is a real-world object, such as a person, location, organization, or product, that can be denoted with a proper name. Once the entities are identified, the Disambiguation process follows, which is the task of correctly associating the named entities with one in the KB. During this process, several candidate entities

are selected from the KB by applying various techniques that can vary among the EL approaches, e.g. word similarity. The candidates are then ranked to select the most suitable one w.r.t to the entity being linked. An entity linking system has to deal with several challenges, such as name variation or ambiguity. The former can occur due to the presence of acronyms, misspellings, or aliases in the text; the latter occurs because the same mention can often refer to different entities in the KB, as in the case of Washington at the beginning of this section. The accuracy of existing linker approaches can vary depending on the dataset used to evaluate each system, and the targeted KB, making it challenging to select a suitable one to be used as part of a KBQA system. To find a compromise between these systems and to enrich the overall linking results, this thesis proposes an ensemble entity linking method in Chapter 4, which benefits from the strengths of each system.

3.2.3 Named Entity Recognition

Named Entity Recognition (NER), also known as Entity Extraction, identifies and classifies named entities in a text into predefined categories such as individuals, companies, places, organizations, cities, dates, product terminologies, etc (See Figure 3.2). For example, in the phrase "Washington was born in the United States", it is possible to recognize two entities, that are Washington that could belong to a *Person* category, and *United States* that could belong to a *Country* category. NER adds semantic knowledge to the text content and helps to quickly understand the main topic of any given text [44]. In a QA system, a NER phase makes it possible to identify the topics of the input question, classify them and eventually simplify the subsequent phases of NLP, as in the case of EL. Most NER algorithms are based on natural language grammar or statistical approaches such as machine learning. While offering good results in entity extraction, Grammar-based approaches [113, 30] require a consistent supply of expert-created grammar rules that may require months of human effort. Statistical approaches, instead, are the current state-of-the-art in the NER task, although they require large manually annotated training data sets. More recently, semi-supervised approaches have been developed for faster training set creation.

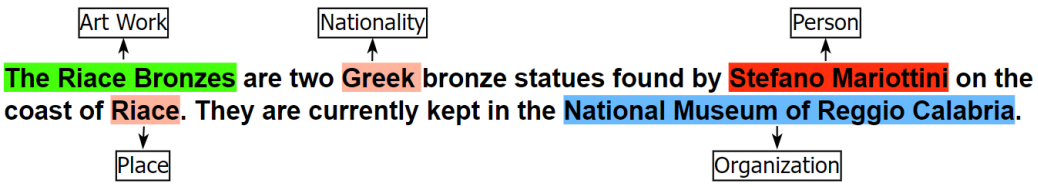


Figure 3.2: Example of Named Entity Recognition.

Table 3.1: BIO notation applied to the phrase: Washington was born in the United States. B-PER stands for the start of a Person type entity, while B-COUN and I-COUN tag an entity of type country.

Washington	**was**	**born**	**in**	**the**	**United**	**States**	.
B-PER	O	O	O	O	B-COUN	I-COUN	O

The approaches implementing a NER system often use the BIO notation to tag or add rich information to a text, differentiating the beginning B and the interior I of the entities. O is used for non-entity tokens. Table 3.1 shows an example of the BIO notation. Entity extraction tasks have received a natural boost with the use of neural networks. The NER task can be seen as a Seq2Seq task, where the input is a sequence $X = (x_1, ..., x_m)$, and the output is a sequence $S = (s_1, ..., s_m)$ of tags associated with each token of the sequence X (B-ORG, B-Date, etc.). Note that the lengths of X and S are equal. During training, a NER model must learn how to calculate the conditional probability $p(s_1, ..., s_m|x_1, ..., x_m)$ so that the model can determine how likely an element in X can be tagged with a particular tag. The conditional probability can be calculated as:

$$p(s_1, ..., s_m|x_1, ..., x_m) = Tagger(x) \tag{3.1}$$

where $Tagger(x)$ is a model that can be implemented in various ways, i.e., using LSTM networks, BiLSTM, *Conditional Random Fields* (CRF) [66], Transformers or combinations of these techniques, among others. Most current state-of-the-art systems [125, 59, 43] use models that include, in some way, CRF or Transformers.

Conditional Random Fields are a probabilistic framework for labelling and segmenting sequential data. They use contextual information from previous labels to increase the amount of information available to make a good prediction. CRFs rely on Feature Functions to express some characteristics of the sequence that the data point represents. Feature Functions could take several input values like the set of input vectors X, the position i of the data point being labelled, the label of data point $i-1$ in X, and the label of data point i in X. To build the conditional field, it is necessary to assign to each feature function a set of weights λ that the algorithm will learn by applying Gradient Descent iteratively until the parameter values converge. A sequence is tagged by mean of:

$$p(y|X, \lambda) = \frac{1}{Z(X)} exp\{\sum_{i=1}^{n} \sum_{j} \lambda_j f_j(X, i, y_{i-1}, y_i)\} \tag{3.2}$$

where Z(X) is the normalization to [0,1] since the output is expected to be a probability, and f_j is the feature function [66].

Chapter 4

SPARQL-QA system for KBQA

This chapter introduces *SPARQL-QA*, a system for open-domain question answering over knowledge bases that implements an NMT-based architecture which allows addressing the OOV problem by resorting to a novel combination of several tools. In detail, the architecture of *SPARQL-QA* is composed of three main modules: *Input preparation, Question Understanding, and Assembling*, as shown in Figure 4.1. In the architecture, the Input Preparation module applies several transformations to the question, preparing it to be used by the next module. Question Understanding analyses the semantics and syntax of the question to produce valuable information that the Assembling module uses to build the final result. In addition, an ensembled entity linking approach, called *Collision Named Entity Linking* (CNEL), is presented, which helps improve the overall performance of the KBQA system. The details of the overall system are described in the following.

4.1 Input preparation

In this phase, the input sentence is processed in such a way that it is polished to attenuate linguistic noise (e.g., shifts in spelling, grammar, and punctuation) and also recast to be used as input for the subsequent phase.

Acronyms normalization In this step, acronyms are converted to the corresponding names, i.e., *UK* becomes *United Kingdom*. Acronyms regularly refer to entities that must be linked to KB resources to use them inside the SPARQL query. By replacing the acronyms, the aim is to improve the entity resolution in the subsequent phases since the real names are more similar to the KB resource

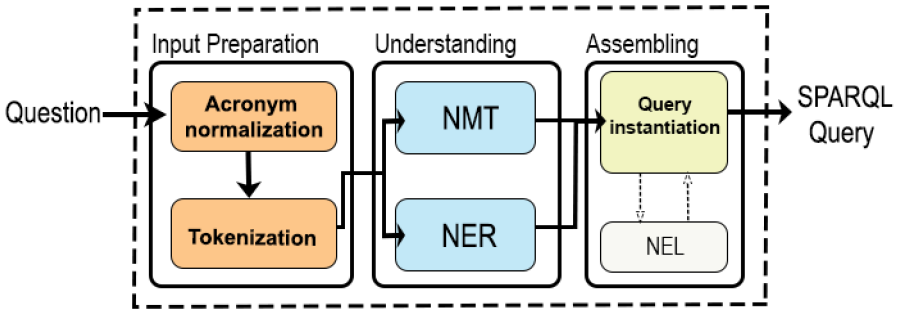


Figure 4.1: Architecture to translate questions into SPARQL.

names. In our approach, this is particularly useful for handling acronyms of countries. The Acronyms normalization task uses two main libraries to accomplish the objective: *spaCy* [55] and *Country Converter* (COCO) [106]. The former library allows for identifying the acronyms in a question thanks to its powerful NER mechanism. The latter, instead, returns the original country name given the acronym form. For example, the question *"How large is the area of UK ?"* becomes *"How large is the area of United Kingdom ?"*. This way, the question text is enriched with information about the full entity name to be used in the subsequent phases (Assembling in Section 4.3).

Tokenization The tokenization process is a fundamental step in almost all NLP approaches. It is the task of chopping a text up into pieces called *tokens*, which are indivisible units of text, usually representing words. Tokens are commonly separated by spaces or punctuation marks. When working with western languages, the tokenization process is relatively simple as these languages place clear word demarcations. In fact, most of the approaches used for natural language tokenization use regular expressions to identify tokens. During this process, Q_{nat} is cleaned by filtering out undesired characters such as punctuation marks and converted into a sequence of tokens $S = \{t_1, ..., t_n\}$. The tokenization process in this thesis used Keras preprocessing library. Figure 4.2 illustrates a tokenization example.

As neural networks work by processing numerical vectors, in this phase, the tokens in S are also converted into word embedding vectors. To this aim, the pre-trained word embeddings provided by $fastText$ [21] were used. The primary motivation for choosing fastText is that it can provide embeddings for those words out of the training vocabulary, thanks to its mechanism of decomposing unseen words into n-grams to obtain the word embedding (See Section 2.4). Furthermore, fastText has been shown to benefit many NLP models that use it as a basis for

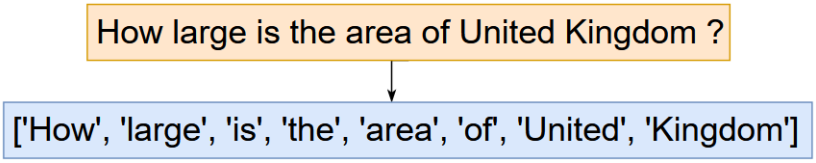


Figure 4.2: Example of sentence tokenization.

generating word representations [99, 36]. The use of fastText is, effectively, a first attempt to mitigate the impact of OOV words while processing the natural language questions.

4.2 Sequence Understanding

In this stage, the pre-processed Q_{nat} is analyzed to obtain both the SPARQL query template and the named entities that serve to build $S'_{Q_{nat}}$ in the subsequent phase. Note that this phase complies with the Question Understanding phase proposed for SP-based methods (see Section 3.2). For this purpose, the SPARQL-QA architecture uses two neural network models performing two NLP tasks: (*i*) Neural Machine Translation (NMT) and (*ii*) Named Entity Recognition (NER).

In an NMT learning task, given a source sequence of tokens, $X = (x_1, x_2, ..., x_n)$, and a target sequence of tokens $Y = (y_1, y_2, ..., y_m)$, the goal is to learn to model the conditional probability of Y given X [16]. During this step, the NMT model is fed with the tokenized version of Q_{nat} obtained in the previous stage to translate it into a *query template*. A *query template* is a SPARQL query skeleton in which the KB entities were replaced by placeholders (details on the template's format are given later in 4.2.1). This template constitutes a logical form that will be instantiated by combining it with the results obtained from the other neural network.

In the NER task, the entities present in Q_{nat} are identified and classified using a dedicated neural network. As in most of the literature, the NER implementation in the proposed architecture also adopts the *BIO* notation to tag a text, differentiating the beginning "*B*" and the interior "*I*" of the entities and leaving "*O*" for non-entity tokens (see Section 3.2.3). The NER model produces a tagging sequence in BIO notation by processing the tokenized version of Q_{nat} generated by the *Input preparation* (section 4.1) phase. The entities identified in Q_{nat} constitute candidate KB entities that can be part of the final query $S'_{Q_{nat}}$ during instantiation. It is important to note that the NER module employs a simplified version of the BIO notation, i.e. it does not distinguish between Person, Place, Organization, or others.

Table 4.1: $\langle Q_{nat}, Q_{sparql} \rangle$ for *Who painted the Mona Lisa?*

Question	**Query**
Who painted the Mona Lisa?	select ?a where { dbr:Mona_Lisa dbo:author ?a }

4.2.1 Training set format

The NMT and NER networks are trained together using the same input obtained by converting the original training set into a novel format called *QQT*. This preprocessing step has a two-fold objective. On the one hand, it aligns the inputs of both NMT and NER tasks, and on the other hand, it reduces the output vocabulary size and helps to mitigate the impact of the OOV words during the translation. Translating entities to URIs/Identifiers can be hard to learn from mere examples. A system can fail if the NER task is not simple and there are a lot of words out of the vocabulary of the training set. OOV words are a serious problem while querying large ontologies.

A QQT-format dataset is composed of a set of triples of the form $\langle Question, QueryTemplate, Tagging \rangle$. In a triple, $Question$ is a natural language question, and $Tagging$ is a labelling sequence indicating which parts of $Question$ are entities. $QueryTemplate$, on the other hand, is a SPARQL query template modified as follows: (i) The KB entities (resources identifier/URI) are replaced by variables; (ii) A new triple is added for each variable created in (i) in the form *"?var rdfs:label placeholder"*. In a straightforward approach, placeholders can be replaced by *Question* substrings depending on *Tagging* to obtain a SPARQL query.

Table 4.1 shows an example of a $\langle Q_{nat}, Q_{sparql} \rangle$ pair for the question: *Who painted the Mona Lisa?*, while Table 4.2 shows the corresponding $\langle Question, QueryTemplate, Tagging \rangle$ instance in QQT format. In Table 4.2, the term $1 denotes a placeholder where the number index 1 means that $1 has to be replaced by the first entity occurring in the question, *Mona Lisa*, in this case, as indicated by the BIO tagging notation. This representation builds on the idea that by replacing individuals in the SPARQL query with placeholders, the NMT network must not learn the complicated mapping between the question entities and the corresponding KB resource but is limited to the generation of placeholders, mak-

Table 4.2: QQT triple for *Who painted the Mona Lisa?*

Question	**QueryTemplate**	**Tagging**
Who painted the Mona Lisa ?	select ?a where { ?w dbo:author ?a . ?w rdfs:label $1 }	O O O B I O

Table 4.3: Comparison of the vocabulary sizes between the original dataset and the QQT format.

Dataset	Initial size	QQT Size	% Reduction
QALD-9	945	506	46.45
Mon600	1952	91	95.33
Mon300	1590	91	94.27

ing the model more tolerant to the OOV problem. For example, in Table 4.2, the query template does not contain any KB resource, so the NMT model does not need to understand that *Mona Lisa* stands for *dbr:Mona_Lisa* (in DBpedia). Remember that it is difficult to collect in a training set the millions of individuals present in a large cross-domain KB, so learning only the KB resources contained in the dataset will undoubtedly lead to the OOV problem when subjecting the model to unseen entities. Another advantage introduced by the QQT format is that the size of the output vocabulary decreases significantly, resulting in a reduction in model training times. A large output vocabulary means a considerable increase in the number of weights the model has to train on the final layers of the neural network and more computation effort to calculate the probability distribution over all vocabulary tokens using the Softmax function. Table 4.3 illustrates approximately how much a dataset output vocabulary is reduced using the QQT format.

4.2.2 The networks

As shown in Figure 4.1, the architecture has two critical parts, the NMT and NER models. It was decided to develop the NMT neural network using the standard *Encoder-Decoder* approach, with BiLSTM and Luong Attention [75], which has shown high performance in the literature [128, 37, 87]. The Encoder extracts semantic information from the question and encodes it into a fixed-length vector v. Instead, the Decoder attempts to decode v into a sequence in the output language. On the other hand, the NER network relies on the BiLSTM-CRF approach proposed by Huang et al. [58], which assigns a tag (in BIO notation) to each token in the input sequence.

It is important to note that the two models share the fact that they have a BiLSTM-based encoder to obtain the semantic information from the input question. For this reason, it was decided to do a joint training of the two models looking to improve the overall training times. This training technique also aims for the two networks to help each other during training by establishing a sort of transfer learning, which allows for improving the learning of each network.

The proposed network, depicted in Figure 4.3, has a single encoder composed

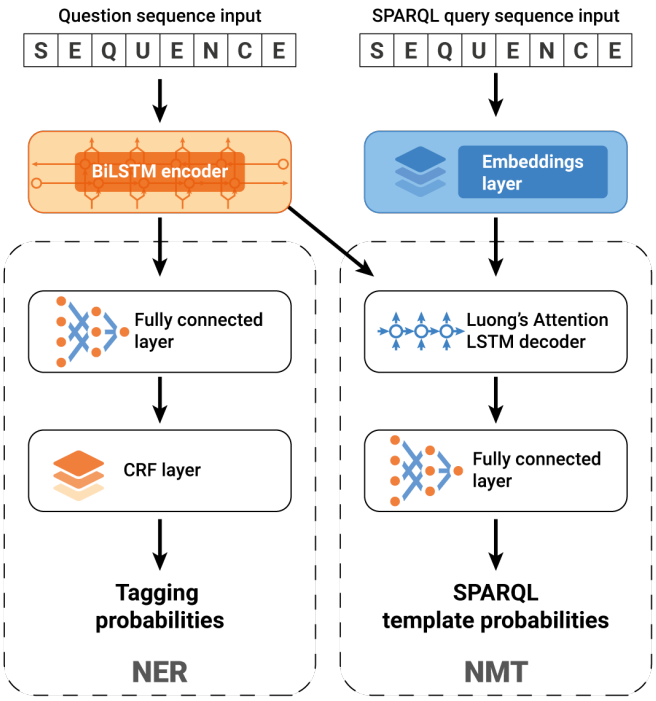


Figure 4.3: Network for joint training of NMT and NER.

of one BiLSTM layer providing information to two sub-branches. The first branch (NER) starts with a fully connected layer that receives the encoder's last hidden state and ends with a CRF layer responsible for determining $p(l|x)$, which refers to the conditional probability of a sequence l to be the tagging sequence of an input x. In this way, it is possible to obtain the most probable tagging sequence l' for x.

The second branch is an LSTM decoder with Luong's Attention initialized with the last cell state c and hidden state h from the encoder. Finally, a fully connected network with Softmax calculates the probability $p(y|x)$ that the sequence y correctly translates x into a $QueryTemplate$.

4.3 Assembling

At this point is where Q'_{sparql} is assembled using the information generated in the previous steps. The idea is to instantiate the query template with the entities

coming from the NER module by replacing the placeholders in the template with the proper entities, following the order provided by the placeholder indexes (see Section 4.2.1). Considering the data at hand, one could intuitively think of replacing the placeholders in Q'_{temp} with the text of the entities identified by the NER model for the question. For example, let us assume the following Q_{nat} question: *Which instruments does Cat Stevens play?* In this case, a system adopting a simple replacing approach, as mentioned before, might translate Q_{nat} into the following query template:

```
SELECT DISTINCT ?a WHERE { ?v dbo:instrument ?a . ?v rdfs:label $1 }
```

while the NER model will identify *Cat Stevens* as the named entity. Finally, the query instantiation step will produce:

```
SELECT DISTINCT ?a WHERE { ?v dbo:instrument ?a .
?v rdfs:label "Cat Stevens"@en }
```

which is a SPARQL query that produces the same answers as Q_{sparql} for DBpedia. This naive replacing approach has two major drawbacks that can negatively affect the quality of the query results, that is:

1. It relies heavily on the correct spelling of the question entities that have to match the value of the *rdfs:label* predicate in the KB. It is common for users to make spelling mistakes when writing or to write names of entities with missing parts (e.g. middle names or surnames);

2. There is no entity disambiguation, so the system cannot distinguish, for example, whether the Washington entity refers to the USA president or the USA state.

In order to address the mentioned issues and to produce higher-quality queries, this thesis proposes an ensemble Named Entity Linking (NEL) module called *Collision Named Entity Linking* (CNEL), which seeks to link the entities identified by the NER module to those in the KB.

4.3.1 Collision Named Entity Linking

The CNEL module comprises three entity linking systems: DBpedia Spotlight [34], WAT [92], and REL [116]. The first system was developed to address the linking task targeting DBpedia, while the other two systems disambiguate the named entities to Wikipedia. The excellent performance observed in the literature in tackling the NEL task and the availability of a public linking service through online

APIs [1] [2] [3] were the primary motivations for selecting these three systems to build CNEL. Due to the interconnection between DBpedia, Wikidata and Wikipedia, it is easy to convert entity links from one knowledge base to another, e.g. links identified by DBpedia Spotlight for DBpedia can be mapped to Wikipedia or Wikidata links.

Algorithm 1 summarizes the entity linking process. The input q is a sequence of tokens corresponding to the NL question. Input *ne* is the entity mention in q, identified by our NER module, that has to be linked against the KB. Finally, the input *els* is a list of entity links discovered by the EL systems for q, built by calling the public API of every single system and discarding the links whose confidence score is below a given threshold. The threshold is chosen by following the recommendation of the authors of each system. It is important to note that each link in *els* is a structure (see Listing 4.1) composed of the actual entity in the KB, the *"spot"* in q, the start and end positions of the spot in q, and a confidence score. The *"spots"* are meaningful substrings in a text segment.

```
{
   uri: ‘‘http://dbpedia.org/resource/Washington,_D.C.’’,
   spot: ‘‘Washington’’,
   start: 3’,
   end: 13’,
   confidence: 0.9569326
}
```

Listing 4.1: Example of the structure of an *els* link element produced for the question: *Is Washington the capital of the United States?*

In particular, to link a NER entity to the corresponding entity in the KB, CNEL starts by selecting from all the links found by the NEL systems those whose spot has an overlap with the NER entity in the question (line 2 of Algorithm 1). The selected candidates are ranked according to a *collision index CI*, which is given by the total number of non-matching tokens between the two substrings so that an exact match has a collision index equal to 0 (line 5 of Algorithm 1). CI is calculated by:

$$CI = |\#TE + \#TS - (2 \times \#OT)| \tag{4.1}$$

where $\#TE$ is the number of the NER entity tokens, $\#TS$ is the number of the link spot tokens, and $\#OT$ is the number of overlapping tokens. In the case of a tie between candidates, CNEL relies on the confidence score provided by the

[1] https://www.dbpedia-spotlight.org/api
[2] https://sobigdata.d4science.org/web/tagme/wat-api
[3] https://github.com/informagi/REL

Algorithm 1 Link a NER entity to the corresponding KB entity

Input: Input question, $q = \{w_1, w_2, ..., w_n\}$, NER entity, $ne = \{w_1, w_2, ..., w_n\}$ substring of q, EL links for q, $els = [l_1, l_2, ..., l_n]$
Output: $l \in els \cup \varnothing$ ▷ An entity link or empty if no link exists

1: **procedure** NE_LINK(q, ne, els)
2: $coll\_els \leftarrow remove\_no\_overlapping\_links(q, ne, els)$
3: **if** $len(coll\_els) == 0$ **then**
4: **return** $\varnothing$
5: $coll\_els \leftarrow rank\_by\_ci\_and\_ce(q, ne, coll\_els)$
6: **return** $coll\_els[0]$

NEL systems for each link as a second ranking criterion. Finally, CNEL takes the highest-ranked link as the final candidate to be used in the query (line 6 of Algorithm 1). If it is impossible to link the NER entity with this approach (no candidates after selecting overlapping links, see line 3 of Algorithm 1), CNEL returns an empty set to indicate the lack of a proper link.

4.3.2 Query Instantiation

In *SPARQL-QA*, the Query Instantiation process consists of trying to cover as many placeholders as possible using the NER entities. For this purpose, the QI module starts an iterative process over the query's placeholders. At each iteration, the NER entity associated with the placeholder index is selected and linked to the KB using the CNEL system. If there is a link for the entity, the placeholder is substituted directly with the KB identifier; otherwise, *SPARQL-QA* falls back to the basic technique of resorting to the output of the NER module. In this way, the combined collision named entity linking is not employed in cases where a better substitution cannot be found. In case the iterative process ends, and there are still unfilled placeholders, i.e. the number of placeholders in the query is greater than the number of available NER entities, the QI module relies on the entities identified by the NEL systems to try to remedy this problem, knowing that unfilled placeholders may result in syntactically invalid queries.

Thus, the QI module tries to fill the empty placeholders one by one, using the NEL systems links that were not used in the previous steps. The problem with this *"greedy"* approach is that it does not follow a KB link selection criteria as there is no NER entity to guide the process. One can think about using the current placeholder index to pick the link at that position from the links array, but this is not a good solution since there is no direct relationship between placeholders and

NEL entities. NEL systems are trained without considering the SPARQL query entities, while in *SPARQL-QA*, the NER model learns by seeing questions where the annotated entities correspond to those in the query, as established by the QQT format. Furthermore, NEL systems often produce more entities than those obtained by the NER model, breaking any possibility of using a position-based approach to pick a suitable linked entity. However, this aggressive technique can be effective in cases where the question is quite simple (1 or 2 entities) and the NER model fails.

Chapter 5

Experimental Analysis

This chapter reports on the experiments carried out to assess the effectiveness of the introduced approach and its components. In particular, *SPARQL-QA* was executed on several KBQA benchmarks, and the results were compared with other KBQA approaches. Most of the systems compared are not publicly available, so for the sake of comparison, the scores (whenever available) for the compared systems on the same benchmarks are presented as reported in the literature.

5.1 Implementation details and benchmark setup

The deep learning models were implemented using the Keras[1] library, a well-known machine learning framework built on top of Tensorflow. Keras abstracts, to some extent, the complex implementation mechanism proposed by Tensorflow, minimising the number of user actions required for everyday use cases. The framework supports the development of convolutional and recurrent neural networks in addition to the standard neural networks. It also provides implementations for commonly used neural network building blocks such as layers, loss and activation functions, optimisers, and others. Another important tool is the Keras Preprocessing package which brings utilities for preprocessing images, sequences and text. This package is used during the Input Preparation (Section 4.1) stage to perform the input tokenisation.

The training process for the NMT and NER models was carried out using Google Colaboratory[2], which is a Jupyter Notebook environment running entirely

[1] https://keras.io/

[2] https://colab.research.google.com/

in the cloud. Colaboratory provides an environment with 12 GB of RAM and the possibility to run the code using GPU and TPU configurations. The programming language selected to develop the system was Python version 3.7.

For the development of the experiments, the system was trained and executed using the well-known and publicly available Monument, QALD-9, QALD-10 and LC-QuAD v1 datasets. The macro precision (P), recall (R), and F1-score (F1) was the choice to assess the performance of the system. In the case of QALD-9 and QALD-10, the assessment also considered the F1 QALD measure [83].

5.1.1 Evaluation measures

The most common measure when evaluating systems for machine translation is the *Bilingual Evaluation Understudy* (BLEU) score. This measure is based on precision-based features and works by comparing the n-grams of the candidate translation with those of one or more reference translations. The BLEU score takes a value between 0 and 1, where a value close to 1 means the translation is more similar to the reference translation. Although BLEU is a robust measure widely used to evaluate machine translation systems, it is unsuitable for evaluating the system presented in this thesis. There are several reasons for not considering BLEU as an evaluation measure for the proposed system. The first one is that *SPARQL-QA* is not only based on NMT but also uses the results of the NER model to compose the final translation, so having good BLEU values for NMT does not guarantee the quality of the final answers. The second reason is that BLEU does not consider the word order in the candidate translation with respect to those in the reference sentences, which can be detrimental when evaluating a system that works with regular languages like SPARQL, which follows a well-established syntax and token ordering. Finally, most KBQA systems are evaluated based on the answers obtained from executing the logical form in the KB, regardless of how the query was built, so it makes sense to adopt this kind of method to enable comparisons with other systems.

KBQA systems are generally evaluated based on the quality of the answers to a given question. Unlike Text-based systems that provide one or several text segments as answers, KBQA systems extract a list of answers from the underlying KB that can come in different formats, i.e. resource identifiers, text, numbers, and boolean values, among others. The most common measures to evaluate the performance of these systems are Accuracy, Recall and F1-score.

Precision is a widely used measure in classification and information retrieval problems. It focuses on measuring the correct detection of relevant instances out of the total instances the model retrieved. In the KBQA context, this can be seen

as the fraction of correct retrieved answers among all answers the system gave for a given question. The Precision is calculated by mean of:

$$Precision = \frac{correct\ retrieved\ answers}{all\ retrieved\ answers} \tag{5.1}$$

Recall is another common measure often used in conjunction with Precision to evaluate KBQA systems. It measures the proportion of instances the model correctly identified as relevant out of the total relevant instances, that is, the model's ability to identify the relevant instances. For a KBQA system, it is the fraction of correctly identified answers among all correct answers and is calculated as follows:

$$Recall = \frac{correct\ retrieved\ answers}{all\ correct\ answers} \tag{5.2}$$

F1-score is the third measure considered to evaluate the system. Usually, Precision and Recall scores are given together and are not quoted individually. The reason is that it is easy to vary the sensitivity of a model to improve Recall at the expense of Precision or vice versa. For example, assuming a model that tends to extract many answers from the KB, it is possible to hit a larger number of *correct retrieved answers* (dividend part on 5.2) and thus increase the Recall; however, the Precision decreases as *all retrieved answers* (divisor part on 5.1) increase. So, to measure the performance of a model, it is possible to rely on the F1-score, which is the harmonic mean of the Precision and Recall. The measure is calculated as:

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5.3}$$

This allows for combining Precision and Recall into a single number. Generally, the best KBQA systems are those with the highest F1 score.

5.1.2 Dataset preparation

Before training the NMT and NER models, the datasets are subjected to a series of transformations to allow for better training and to obtain the QQT dataset version. The first transformation applied to the data looks to improve the query tokenisation process since a SPARQL query is a combination of tokens defined by the language syntax and the KB resources and predicates identifiers, including punctuation marks, arithmetic symbols, and natural language words, among others. This transformation follows the encoding approach suggested by Soru et

al. [104]. With this approach, the resource URIs are shortened using the RDF prefixes, and column characters (":") are replaced with underscores to keep each URI as a single token. Punctuation marks and other non-alphabetic symbols such as brackets, dots, wildcards, and arithmetic operators are replaced by verbal descriptions. For example, the character *"{"* is replaced by the token *"brack_open"*. By applying this transformation, the query will contain only alphabetical characters and underscores so that the tokenisation will be much easier, and consequently, the training phase will benefit. Listing 5.1 shows how a SPARQL query is encoded.

```
--Original query--
SELECT DISTINCT ?uri WHERE { dbr:Colosseum dbp:location ?uri .
<http://dbpedia.org/resource/Trevi_Fountain> dbp:location ?uri . }

--Encoded query--
select distinct var_uri where brack_open dbr_Colosseum
dbp_location var_uri sep_dot dbr_Trevi_Fountain dbp_location var_uri
sep_dot brack_close
```

Listing 5.1: Example SPARQL query encoding using the approach Soru et al. [104]. Note that there are no characters other than letters and underscores.

In the second step, the questions are annotated with the BIO notation, providing the necessary information to train the NER model. Most of the large KBQA datasets proposed in the literature are created employing hand-crafted templates that are instantiated using entities and predicates extracted from the knowledge base to create the dataset question-query pairs. This template mechanism ensures the existence of a correspondence between the entities found in the query and the entities in the natural language question, which allows for automating the entity annotation process. The annotation process follows four main steps to annotate each question in the dataset:

1. Take a question-query pair in the original dataset.
2. Look for the entities in the query, i.e., tokens starting with *"dbr"* for DBpedia or *"wd"* for Wikidata, and create a human-readable representation of them.
3. Look for substrings in the question that match the human-readable representations obtained in point 2 and mark the respective tokens with the BIO notation.
4. Mark the other tokens in the question as O (non-entity token).

The human-readable representation created at point 2 is a critical piece of the annotation process and is slightly different between DBpedia and Wikidata datasets.

Table 5.1: Comparison on Monument.

	Mon300			Mon600		
	P	**R**	**F1**	**P**	**R**	**F1**
NSpM	0.860	0.861	0.852	0.929	0.945	0.932
SPARQL-QA	0.824	0.826	0.823	0.826	0.831	0.828

In the case of DBpedia, the entity URI in the query already contains the entity name, e.g. for the URI *dbr:Albert_Moss_(cricketer)*, all the text following *"dbr:"* part constitutes the name. So to create the human-readable representation, the name is cleaned by removing the parts that are less probable to be part of the question text. For instance, in the previous example, everything after the comma (,) is removed, leaving *Albert Moss* as the human-readable representation. In Wikidata, on the other hand, entities are identified using an alphanumeric identifier, for example, *wd:Q4710870*. Therefore, to create the human-readable representation of an entity, it is necessary first to query the KB asking for the entity name (rdfs:label predicate value) and then clean it using the same approach as for DBpedia.

The annotation process also applies in the case of manually created datasets, where there is no clear correspondence between the entities in the query and the question. Manually created datasets are usually small, so it is possible to check the generated annotations manually and fix possible mistakes. Finally, the SPARQL queries are modified following the steps proposed in section 4.2.1 to create the placeholders, thus concluding the creation of the QQT dataset.

5.1.3 Evaluation on Monument dataset

The Monument dataset was created to evaluate the Neural SPARQL Machines (NSpM) [104] approach introduced by Soru et al. The dataset contains 14,778 question-query pairs concerning instances of the class Monument *(dbo:Monument)* present in DBpedia. For the sake of comparison with the NSpM system, its Learning Module was trained following the setup found in [104], where the authors proposed two instances of the dataset denoted as Monumet300 and Monument600 containing 8,544 and 14,788 pairs, respectively. In both cases, the dataset splitting fixes 100 pairs for both the validation and test set, keeping the rest for the training set. All the data is publicly available on GitHub[3].

On the other hand, to train *SPARQL-QA*, a grid search hyperparameter tuning was performed centred on three metrics: the embedding size of the target language, batch size, and LSTM hidden units. The number of epochs was set to 5, shuffling

[3]`https://github.com/LiberAI/NSpM/tree/master/data`

Table 5.2: Comparison on OOV entities dataset.

	Mon300 model			**Mon600 model**		
	P	**R**	**F1**	**P**	**R**	**F1**
NSpM	0.097	0.123	0.101	0.110	0.110	0.110
SPARQL-QA	0.818	0.820	0.816	0.828	0.830	0.826

the dataset at the end of each one. After the tuning, the hyperparameters were set to 300, 96, and 64 for embedding size, LSTM hidden units, and batch size, respectively. The results in Table 5.1 show that the presented approach performs reasonably well, reaching F1-score values greater than 0.80, while NSpM performs slightly better.

The cases where *SPARQL-QA* did not provide an optimal answer were investigated, and it was found that the performance of the proposed approach is mainly affected by errors in entity linking. In particular, the dataset contains several questions that lack context for correctly determining the entities' URIs. For example, for the question *"What is Washington Monument related to?"* the system links the entity to the URI: *dbr:Washington_Monument*, but the gold query uses the specific URI: *dbr:Washington_Monument_(Baltimore)*. Note that there is no reference to *Baltimore* in the question text, and there are Washington Monuments also in other places like Milwaukee and West Point, according to DBpedia. However, NSpM often uses the specific URI of the gold query.

To better understand this issue, it was decided to perform a tougher experiment. For this purpose, the templates of NSpM and a randomly selected set of unseen monument entities extracted from DBpedia were used to create a new test set of 200 pairs with OOV words. Subsequently, NSpM and *SPARQL-QA* answered the questions contained in this dataset. As a result, Table 5.2 shows that the proposed approach confirms the same good performance (F1 score greater than 0.80), demonstrates a better generalizing power being resilient to unseen entities, and also performs better than NSpM.

5.1.4 Evaluation on QALD-9

Question Answering over Linked Data (QALD)[4] is a series of challenges that aim to assess and compare QA systems for KBQA. This experiment considered QALD-9, the benchmark released as part of the ninth edition of the challenge. QALD-9 is a standard dataset among the KBQA systems, so it can be possible to establish a comparison between *SPARQL-QA* and those systems. The dataset contains 558 question-query pairs, targeting DBpedia, and is divided into 408 and 150 questions

[4] `https://qald.aksw.org`

Table 5.3: Comparison on QALD-9.

	P	**R**	**F1**	**F1 QALD**
Elon	0.049	0.053	0.050	0.100
QASystem	0.097	0.116	0.098	0.200
TeBaQA	0.129	0.134	0.130	0.222
wdaqua-core1	0.261	0.267	0.250	0.289
gAnswer	0.293	0.327	0.298	0.430
NSQA	0.314	0.321	0.308	0.453
SPARQL-QA	0.330	0.336	0.327	**0.478**

for training and testing, respectively. The queries produced by *SPARQL-QA* for QALD-9 were executed using the official DBpedia SPARQL Endpoint running the DBpedia version 2022-03. For the compared methods, this section reports the results available in the literature that were obtained for QALD-9 by using DBpedia 2016-10[5]. It is important to note that this dataset is very challenging to be approached using learning techniques, given the small training set that does not cover all question types of the test set.

During the training phase with QALD-9, a cross-validation process with five splits (k=5) was performed together with the hyperparameter tuning process, motivated by the small number of examples in the training set. The settings were the same as with the Monument dataset, but setting the number of epochs to 30. Table 5.3 shows the performance of the QALD-9 challengers, the NSQA [62] system, and the previous version of *SPARQL-QA*. As can be seen, *SPARQL-QA* performs indicatively well.

Given the small size of the dataset, it was decided to expand the training set to understand the behaviour of the proposed system properly. In order to expand the dataset, several question templates from the gold questions were created by annotating all the named entities with the spaCy NER tool first and then checking the annotations manually. Subsequently, the new dataset was created by replacing the annotated entities with others randomly selected from the KB until creating a total of 1816 pairs. This is the *expanded* training set. Further, the same query generation process was applied to the pairs of the test set. The new pairs were added to the previously expanded dataset to create a new benchmark (labelled *expanded w/test*) containing 2331 examples. No pair or gold query from the original

[5]This old version of the DBpedia endpoint (2016-10) is no longer available online. For the same availability reason, the compared methods cannot be run on the new version of DBpedia (2022-03). Thus, this is the best picture that *SPARQL-QA* could obtain. Since there is no difference in the dataset of questions, the results are sufficiently indicative of the performance of the systems.

Table 5.4: Evaluation on QALD-9 expanded datasets.

	P	**R**	**F1**	**F1 QALD**
expanded	0.317	0.325	0.315	0.462
expanded w/test	0.687	0.688	0.682	0.794

test set was added to this dataset. As shown in Table 5.4, expanding the dataset with the same question types (*expanded dataset*) does not improve the results too much; rather, it could be harmful because there is more repeatability in a training set that is not representative of the test set, leading to less generalisation. Moreover, when the training set is expanded with patterns from the original test set (see the expanded w/test row), the system reaches an excellent performance (F1 QALD of 0,75). This latter experience confirms that the training set of QALD-9 is not fully representative of the test set.

5.1.5 Evaluation on QALD-10

QALD-10 is a Wikidata-based dataset composed of 412 pairs for training and 394 human-curated questions for testing, presented for the tenth edition of QALD, where *SPARQL-QA* was one of the participants. In order to execute the queries, the challenge organisers provided a fixed endpoint[6] to ensure replicability. In the case of this dataset, it is important to mention that the *Transfer Learning* technique was applied in addition to cross-validation to train the models. Transfer learning is the task of reusing the knowledge acquired by a model trained using another dataset to solve a similar problem [86]. In these cases, the pre-trained model can either be used as a starting point for a model in a second task or be fully trained using another dataset. For QALD-10, the idea was to train the *SPARQL-QA* models using the LC-QuAD v2.0 [39] dataset, which has about 30,000 question-query pairs and then reuse the knowledge acquired in this vast dataset to learn from the dataset proposed for the challenge. LC-QuAD v2.0 was not considered for the experimental study due to its major quality problems, mainly related to its use of crowdsourcing platforms to generate and diversify the question texts. More information about the LC-QuAD v2.0 issues can be found in [37]. Beyond these problems, using this dataset to pre-train the model helps to learn additional question and translation patterns and, above all, allows learning about the use of certain Wikidata predicates that are difficult to learn with QALD-10. Table 5.5 shows the results obtained by the proposed system, which ranked 1st in the competition. The challenge results are available at [9]; the pa-

[6] `http://sems-vm-1.informatik.uni-hamburg.de:443/api/endpoint/sparql/`

Table 5.5: QALD-10 challenge scores.

	P	R	F1	F1 QALD
Gavrilev et al.	0.1421	0.1400	0.1403	0.1948
Baramiia et al.	0.4289	0.4272	0.4277	0.4281
Steinmetz et al.	0.3206	0.3312	0.3215	0.4909
SPARQL-QA	0.4538	0.4574	0.4538	**0.5947**

per fully describing the competition will be published as part of the ESWC 2022 proceedings. This result demonstrates that the proposed approach can be applied to different knowledge bases while maintaining the same good performance.

Challengers autopsy Although *SPARQL-QA* obtained good results in the challenge, it is always helpful to analyse how opponents perform in the competition by understanding the main reasons that affect or improve the systems' performances compared to *SPARQL-QA*.

The first challenger in Table 5.5, Gavrilev et al.[7], presented a multilingual ensembled system combining two existing approaches, the well-known DeepPavlov [27] KBQA and Knowledge Graph Embedding Based Question Answering (KEQA) [57]. The first step of this system is to translate questions in other languages to English, thus supporting multilingual question answering by simply using the English-based QA approaches mentioned above. Depending on the question language, the authors used different pre-trained versions of the T5 model for translation. KEQA and DeepPavlov then process the questions in search of an answer; if both systems can answer a certain question, priority goes to DeepPavlov. Gavrilev et al. carry out some pitfalls that can explain its performance in the competition (see Table 5.5 first row). The first problem is that the authors used the pre-trained models given by DeepPavlov and KEQA without retraining or fine-tuning them on the QALD-10 training set; this could affect the overall system as possible new patterns in the test set may not be treated correctly. Two other inconveniences come from DeepPavlov; this QA system does not support boolean questions, which represent around 16% of the test set, and the entity linking mechanism uses the Levenshtein Distance [130] among words to disambiguate the entities, thus not considering the question context. KEQA, instead, only supports simple questions[8] that represent less than 12% of the test set. Furthermore, it is trained on the Simple Questions

[7]The details about this system were provided directly by the authors, as there is no published paper.

[8]Simple Questions are answered by a SPARQL query with a single triple pattern and no counting, filtering or grouping, or other complex syntax.

dataset based on Freebase [22] (discontinued since 2016), so the answer has to be matched to Wikidata, with no guarantee that such a match exists. To conclude, the Gavrilev et al. approach only considers single-element answer sets, which means that for the question *"Who are the children of Barack Obama?"* the system will always answer with just one Wikidata entity when it should provide two.

Baramiia et al. presented a Ranking Approach to Monolingual Question Answering over Knowledge Graphs [17] that take advantage of the state-of-the-art transformers and the Wikidata knowledge base embeddings provided by PyTorch-BigGraph [69]. The knowledge base embeddings are feature vectors for entities and predicates in a knowledge graph where adjacent entities are supposed to be close to each other in the vector space. The authors fine-tuned a BART model to produce word embeddings as close as possible to those provided by PyTorch-BigGraph. The idea is then to feed the model with a question and predict two embeddings, one for the entity and another for the predicate. Given these two vectors, the authors find the nearest ones (precisely 3 for the entity and the predicate) in the PyTorch-BigGraph embeddings using the Scalable Nearest Neighbors (ScaNN) method. The entities and predicates associated with the nearest embeddings act as candidates to produce the final SPARQL query. Even when this approach builds on state-of-the-art techniques, the fact that it has been developed to support only simple questions undoubtedly affected its performance in the competition. This limitation also caused the BART model to be fine-tuned with only 145 (queries with a single triple pattern) out of 412 samples, making it complicated to generate good embeddings for those combinations of entities and predicates not present during training but included in the test set. This approach, like Gavrilev et al., only considers single-element answer sets.

The approach by Steinmetz et al. [101] takes advantage of the AMR graphs to create a SPARQL query capable of retrieving the question answer. An AMR is a graph-based representation of the semantic information of a sentence and has already been used in the field of KBQA [62], showing good results. In the first step, the authors obtain an AMR representation of the question using a pre-trained multilingual AMR parser and then align the graph to the question by associating to each node/edge the word or words in the sentence that evoke them. In the second step, the system generates candidate queries from the paths of a simplified version of the AMR graph obtained earlier. For entity and property identification, the authors used Falcon 2.0 [97]; if the tool does not identify a property, then a fuzzy search is performed on the properties of the training dataset. Finally, the queries are executed, and if they produce results, the categories of these results are compared with an answer type category obtained in precedence and accepted if they match. The answer type category categories are predicted using a pre-trained model for this task. This approach has the advantage that it does not

require end-to-end training on the provided dataset. According to the authors, at the current stage of the approach, they do not properly support boolean questions and questions requiring operators like LIMIT, ORDER or FILTER in the SPARQL query. Also, they manifest to have problems during the entity and relation linking, something that was marked to solve in the future. These pitfalls, together with the complexity of the QALD-10, certainly impacted the system's performance.

As with any system based on machine learning, the *SPARQL-QA* performance depends significantly on the amount and quality of the data available for training. This is the motivation why LC-QuAD v2.0 was employed to train the models, i.e. to make the models learn as many patterns as possible. Even by doing this, the system cannot correctly translate several questions, such as comparison questions or questions with superlatives not found in any of the datasets used for training. Another problem affecting the system is the incorrect identification of some predicates, which motivated the future integration of a relation-linking system that helps to close this gap. On the other hand, one of the advantages of our system is the ability to learn complex operators such as COUNT, LIMIT, GROUP BY and others, as long as they are well represented in the dataset. Moreover, it is able to learn boolean queries.

5.1.6 Evaluation on LC-QuAD v1

Large-Scale Complex Question Answering Dataset (LC-QuAD v1) [114] (`https://github.com/AskNowQA/LC-QuAD`) was created in 2017 to provide a sufficiently large, complex, and varied dataset for the application and evaluation of machine learning-based QA approaches. The benchmark has 5000 question-query pairs, the training set has 4000 questions, and the test set has the remaining 1000 pairs. LC-QuAD v1 is based on DBpedia, and more than 80 % of the questions contain two or more relationships, increasing its complexity. In order to evaluate the system in LC-QuAD v1, the same procedure as with the QALD datasets was used to adjust the model parameters, and the official DBpedia SPARQL Endpoint with DBpedia 2022-03[6] was employed to execute the queries.

The results in Table 5.6 show that the approach performs well, something expected in a large dataset like LC-QuAD v1. Diving into the details, it was observed that sometimes the SPARQL template generation underperforms. Indeed, the system might produce wrong predicates, especially in large queries, because of the ambiguity problems in DBpedia. For instance, to refer to the leader of a country, city, or organization of any kind, DBpedia can use predicates such as *dbp:leader*, *dbp:leaderName*, or *dbp:mayor* interchangeably, making it difficult for a system to select the correct predicate, which depends on how an entity has been defined in the KB. In this particular case, *SPARQL-QA* tends to use *dbp:leader* (the most

Table 5.6: Comparison on LC-QuAD v1.

	P	**R**	**F1**
WDAqua	0.220	0.380	0.280
QAMP	0.250	0.500	0.330
NSQA	0.448	0.458	0.444
SPARQL-QA	**0.545**	**0.542**	**0.541**

frequent in the training set). However, this phenomenon affects a limited number of cases, and the overall performance is good.

5.1.7 Ablation study

For completeness, this section reports the result of an ablation study measuring the impact of the various modules of the system. To this aim, one module at a time was removed, and then the *SPARQL-QA* scores were recalculated for each dataset. For QALD-9, the ablation study considered the original and the *expanded w/test* datasets in order to provide a more extensive analysis.

Acronym normalization.

As shown in Table 5.7, the lack of the acronyms normalization module basically does not affect the final results for the two Monument dataset variants. This low impact is due to the lack of acronyms in the Monument test sets. The phenomenon also applies to the LC-QuAD v1 dataset, where the results are the same after removing the analyzed module. However, in the case of QALD datasets, Table 5.8 shows that the F1 QALD measure for the models without acronym normalization is lower than the previous values. This behaviour is due to the presence of acronyms the system without the module under assessment was not able to deal with. All in all, acronyms normalization improves the overall performance of the system in case acronyms are part of the input. It is important to note that the task of the CNEL can sometimes adjust the acronyms.

Table 5.7: Ablation: F1 scores Monument, Monument OOV.

	Mon300 model		**Mon600 model**	
	test set	**OOV**	**test set**	**OOV**
SPARQL-QA	0.823	0.816	0.828	0.826
w/o Acro	0.823	0.816	0.828	0.826
w/o CNEL	0.811	0.734	0.779	0.739

Collision Named Entity Linking.

Results in Table 5.7 show that the lack of entity linking has a negative impact on the system results for the Monument datasets, either in the original test set or in the OOV version. On the other hand, after removing the CNEL module for LC-QuAD v1, the changes in the scores are small. The low impact of CNEL in this dataset is because the entities found in the questions frequently coincide with the text value of the *rdfs:label* KB predicate used by the proposed approach to compose $S'_{Q_{nat}}$. Concerning the more involved QALD-9 dataset, the results show a drop in the F1 QALD measure without CNEL (see Table 5.8). The performance decreases by about 3% while using the model trained on the *expanded w/test* dataset (center column). At the same time, the performance on QALD-10 is also affected by the lack of the CNEL module. This analysis shows the usefulness of CNEL on more involved questions.

Joint training.

Joint training of NER and NMT networks is a novelty of the proposed approach. During the experiments, each network was trained separately, and then the results obtained with a system using these models were compared with the results using joint training. The first benefit of the joint training is to save execution time since it is not necessary to wait for the NMT network training to start with the NER network and vice versa. Saving time is mainly useful when the availability of computational resources is a problem. The experiments showed a reduction in training time of about 40% when applying the joint training technique. The second and expected benefit is to obtain a better generalization from the data since the networks for NER and NMT might help each other. In the Monument dataset, the system performance remains mostly the same when training the models separately; however, with a more involved dataset like QALD-9, the impact of the joint training is evident. By training the networks separately, the system could not obtain an F1 QALD value greater than 0.406, much less than the 0.478 obtained with the joint training. For recall, precision, and F1-score, the obtained values were 0.281, 0.275, and 0.271, respectively, which are also smaller than the

Table 5.8: F1 QALD scores for the ablation study on QALD-9.

	QALD-9	**Expanded w/test**	**QALD-10**
SPARQL-QA	0.4781	0.7948	0.5947
w/o Acro	0.4670	0.7810	0.5883
w/o CNEL	0.4614	0.7674	0.5594

best results with joint training. This analysis confirms the positive impact of the joint training technique.

Chapter 6

Related Work

Several approaches have been proposed to address the KBQA problem in the last few years. This chapter analyses some SP-based systems, highlighting the main differences against the current proposal. The analysis divides the approaches into two main groups: *pattern-based* and *deep-learning-based*.

6.1 Pattern-based

The idea of using patterns to map natural language queries to SPARQL queries is not new in this field and has been exploited by several authors in the literature [93, 107].

The approach presented by Pradel et al. [93] draws on the postulate that, in real-life applications, all user questions are variations of a few typical question families. This postulate finds its roots in the results of a research [105] that analysed large question logs from English-speaking users on a real Web search engine. Consequently, the core of the proposed system, called SWIP, relies on capturing and writing query patterns representing these question families. The question processing in SWIP consists of two main steps, the *natural language query interpretation* and the *pivot query formalisation*. The former step starts by identifying the named entities present in the question text that can correspond to entities in the underlying KB. Then, in a second moment, a question dependency tree is built using a part-of-speech (POS) [78] tool, taking into account the previously identified named entities, so the entity text is seen as a single token by the tool. The third stage of the query interpretation aims at identifying the question focus, which allows knowing the type of SPARQL query to be created. Finally, with this information, the authors create what they call a *pivot query*, a structure halfway between the NL question and the targeted formal query. This pivot query is written using the

pivot language defined by the authors and explicitly represents extracted relations between keywords of the NL question.

In the *pivot query formalisation* step, each element of the user question expressed in the pivot language is matched to an element (either class, property, instance or literal) of the knowledge base. Then, predefined query patterns are mapped to those elements in the pivot query to obtain a list of potential interpretations of the question that are ranked according to their relevance and proposed to the user in the form of reformulated questions. Thus, the sentence selected by the user allows the final SPARQL query to be built. The patterns are modular structures containing optional and repeatable subpatterns. In this approach, the patterns are defined as a 4-*tuple* $(G, \mathcal{Q}, \mathcal{SP}, \mathcal{S})$ targeting the underlying KB, which avoids exploring the ontology to link the semantic entities identified from the keywords. In a query pattern p, G is a connected RDF graph that describes the pattern's general structure and represents a family of queries. $\mathcal{Q}$ is a subset of elements of G, called *qualifying elements* and are taken into account during the mapping of the user question and the considered pattern. $\mathcal{SP}$ is a set of sub-patterns for p, and $\mathcal{S}$ is a template of a descriptive sentence containing substrings for each sub-pattern and qualifying element that serves to build the reformulated questions presented to the user. All patterns and sub-patterns are created manually by experts.

Even though the above approach is interesting, it may be affected by the fact that they do not use an entity-linking system to deal with the entities, so they rely on the *rdfs:label* property, which requires the correct spelling of the entity name. However, beyond this, the main weakness of this approach is the number of person-hours that must be devoted to creating the patterns. Experts must be familiar with the domain and the underlying KB and have to analyse the available data carefully to identify possible patterns.

Steinmetz et al. [107] presented a pattern-based approach whose main novelty is the low coupling to an underlying KB. To achieve this low dependency, the KB must be constructed in RDF(S)/OWL, provide terminological knowledge about the used vocabulary/ontology, and make the actual facts available as assertional knowledge. For efficient search and lookup reasons, the authors create lookup stores (tables in a relational database) containing the URI and the original and alternative labels of classes/categories, properties and actual entities in the knowledge base. The original and the alternative labels correspond to the values of the *rdfs:label* and *skos:altLabel* properties, respectively. The labels in the lookup stores allow the correct mapping of type information, verb relations and entities in the question to the KB. For DBpedia, the lookup stores also store the labels of redirects and disambiguation pages of each entity, thus increasing the probability of finding the correct entity mentioned in the natural language question. The ap-

proach starts by parsing the question using the Stanford lexical parser [1], resulting in a parse tree that provides the question type, part-of-speech information, and word relations. According to the question type, the authors identify the question focus and, therefore, the subject of the question, which is used as the result variable in the SPARQL query. In the second step, the system creates general RDF triples, where the subject, property and object are represented by the phrases extracted from the question as placeholders. These general triples are created by analyzing the patterns of POS tags in the parse tree of the question. Each POS tag combination (a pattern) is identified manually and is associated with a set of RDF triples. Subsequently, the general triples are transferred to real RDF triples in a mapping process that considers the underlying knowledge base. To this end, each term in the general triples is matched to one or several labels in the lookup stores, allowing to obtain the RDF URIs to be used in each position of the KB-specific triple. A triple mapping can result in multiple versions of the same triple and, thus, in multiple SPARQL queries for the same question. Each term mapping has an associated relevance score that allows for providing an overall mapping score for each of the queries generated. Finally, the resulting SPARQL queries are executed and ranked depending on the expected answer type and the mapping score. The best-ranked query is used to answer the user's question.

As a pattern-based approach, this system requires a lot of pre-processing time to identify and write the possible patterns. On the other hand, looking for a proper mapping for each component in a triple can be very expensive, especially in large KBs with millions of individuals and thousands of properties and classes. In addition, using text labels to determine the KB properties may lead to lexical gaps in cases where the ontology does not provide information about synonyms, as in the case of DBpedia. For instance, the fact that two people are married is represented by the DBpedia property *dbo:spouse*, while in NL, several other expressions are used, such as "wife/husband of", "married", and "in a relationship", among others. This problem is intrinsically solved in deep-learning-based systems, as word embeddings for synonyms are supposed to have similar representations.

Although the systems introduced above proved to work reasonably well on selected benchmarks, the human effort required to identify, create and maintain the patterns makes the pattern-based systems an unpopular choice for dealing with the KBQA problem. In the deep learning approaches, this limitation is not present, as they are supposed to learn the patterns from the training data and, in the case of new question forms, are expected to provide valid SPARQL query approximations.

[1] https://nlp.stanford.edu/software/lex-parser.shtml

6.2 Deep Learning-based

Currently, most of the proposed systems that constitute the state-of-the-art for the KBQA task use deep neural networks in some way as part of their architecture. This fact is in line with the excellent results achieved in several areas with the application of deep learning, especially NLP, where tasks such as translation, Part-of-speech tagging, and NER, among others, have achieved performance never seen before.

The *Neural SPARQL Machines (NSpM)* [104] approach proposed the idea of treating SPARQL queries as a foreign language that can be translated using machine translation tools. NSpM uses an end-to-end architecture that heavily relies on a Neural Machine Translation (seq2seq) model to translate NL questions into SPARQL. The system architecture comprises three main components: a *generator*, a *learner* and an *interpreter*. The generator aims to produce a training set by instantiating a set of query templates with entities extracted from the knowledge base. The query templates are defined in advance and constitute an alignment between a natural language question and its respective SPARQL query, with entities replaced by placeholders. The learner implements an LSTM-based seq2seq architecture with two layers modelling a machine translator that learns from the previously generated dataset how to translate questions in natural language into a sequence that encodes a SPARQL query. Before training, the authors propose to encode the query by replacing the SPARQL-specific characters and operators with predefined tokens to prevent the tokenisation tools from removing important tokens. Also, URIs are shortened using prefixes, and column characters are replaced with underscores to keep each URI as a single token. Indeed, *SPARQL-QA* adopts this encoding process due to its positive impact on correctly making the model learn the SPARQL syntax. In the last step of the NSpM architecture, the interpreter decodes the encoded query representation obtained by the learner into the final SPARQL query. The interpreter basically performs the reverse operation of the encoding process explained above.

To demonstrate the approach's applicability and as a research contribution, the authors created the Monument dataset, which contains information concerning instances of the *dbo:Monument* class in DBpedia. NSpM was run on this dataset and evaluated using the purely syntactic BLEU score, showing it performs well in reproducing the syntax of the target queries. Unfortunately, this approach underperforms on unseen questions with OOV words, as no effort is made to address this issue. The model is left to learn from the data the mapping between the entities in the question and the URIs in the query, so entities not present in the training set will not be translated correctly. An exhaustive analysis of this problem was made in section 5.1.3 of this document. Beyond this issue,

NSpM demonstrated that NMT-based systems are a valid proposition to address the KBQA problem and paved the way for approaches such as the one presented in this thesis. In addition, the *generator* component was the basis for creating DBNQA [50], the largest DBpedia-targeting dataset so far and a superset of the Monument dataset.

The application of Neural Machine Translation to address the KBQA problem was also studied in depth by Yin et al. [128], who used and compared eight different NMT models to translate NL questions into SPARQL on well-known datasets. The compared models rely on three main types of architectures, RNN-based, CNN-based and Transformers models, since those represented the best-performing NMT architectures in the field at the time of the publication. For the RNN-based models, the authors decided to use the NSpM model [104] and two other variants of the same model with Bahdanau Attention [16] and Luong Local Attention [75] to investigate the effect of attention mechanisms. Moreover, Yin et al. [128] also adopted the model proposed by Luong et al. (LSTM_Luong) [75] and the GNMT system proposed by Wu et al. [122]. The former is a four-layer LSTM model with Luong Local attention. GNMT, on the other hand, was divided into two models, GNMT-4, which has four stacked LSTM layers, and GNMT-8, which has eight layers. The GNMT architecture proposes using residual connections to deal with the loss of information caused by the use of many layers. In addition, GNMT utilizes a bi-directional RNN on the first layer of the encoder that reads the input sequence from left to right and right to left. Concerning the CNN-based models, the paper considered the well-known ConvS2S [42] model, which proposes an encoder-decoder architecture with attention, where both consist of stacked convolutional blocks. The blocks are composed of a single-dimensional convolutional layer followed by a Gated Linear Unit (GLU). The last considered model was the one by Vaswani et al. [117], which proposed tackling the NMT problem using self-attention, thus avoiding convolution and recurrence. The models based on this architecture are commonly called Transformers and have achieved state-of-the-art results on multiple translation tasks. It is important to note that for the experiments, the SPARQL queries were encoded following the approach by Soru et al. [104]. All eight models were trained on the Monument [104], LC-QuAD v1 [114] and DBNQA [50] datasets and evaluated using the Perplexity, BLEU and Accuracy scores. As a result, the ConvS2S approach showed to be the best, slightly outperforming the competitors in all the measures. On the other hand, the LSTM models with attention and Transformers also demonstrated to perform well. Incredibly, the GNMT models proved to be, by a considerable margin, the worst solution to the task. The authors assume that this lack of performance may be associated with the complicated architecture of this model since reducing the number of layers reflects a slight performance improvement. Although this research

does not propose a system for KBQA per se, it provides valuable information that can serve as a reference point for approaches that aim to use NMT to address this problem, as in the case of this thesis.

Luz et al. [76] also used an encoder-decoder architecture based on LSTM with Bahdanau's attention [16] to address the task of translating natural language into SPARQL queries. The first part of this research concentrated on finding a good vector representation (word embeddings) for tokens in both the source and target language. The vector representations of the natural language were calculated using the well-known vector-based GloVe [89] model. On the SPARQL side, the authors tested different ways of calculating the word embeddings, i.e., random initialization, TF-IDF/PCA [13, 10], W2V10 [80] and a self-created vector representation approach. The *"self-created"* approach proposes to train an LSTM-based seq2seq model with an attention mechanism and use the alignment table generated by the attention to match source and target tokens. With this matching, the vector representation of a target language token is computed by first looking up the natural language word it matches and then obtaining the corresponding GloVe vector. The second part of the paper consisted of training a seq2seq model with and without attention using the four target vector representations mentioned before to translate from natural language to SPARQL. For the experiments, Luz et al. transformed the logical queries in the former Geo880[2] dataset into equivalent SPARQL form and created an OWL ontology of the Geo880 domain. The models were evaluated using the Accuracy and the Syntactical Errors. The latter is calculated as the number of queries with syntactical errors over the total number of queries. As a result, the models using the self-created vector representation approach highly outperformed the other models and performed well in addressing the translation task. In any case, it would be interesting to let a model learn the target vector representations by itself and then compare it with the previous proposals. Concerning the system presented in this thesis, the Luz et al. approach does nothing to deal with the OOV words and lexical ambiguities; however, it still demonstrates that NMT constitutes a valid proposition in this field.

The approach by Chen et al. [28] belongs to the family of query graph ranking approaches that treat the KBQA problem as that of generating a set of query paths on the KB and ranking them w.r.t. the given question. These approaches first construct several query graphs with respect to the KB, which constitute structured representations of a natural language question. Subsequently, the candidates are ranked according to the similarity with the question's lexical and semantic structure to select the most suitable one to create the final SPARQL query. Chen et al. retain that the query graph generation process used by the existing systems has

[2]https://www.cs.utexas.edu/users/ml/nldata/geoquery.html

a significant drawback: it produces a considerable number of noisy query graphs. The problem with these noisy graphs is that they have incorrect structures, but their components have high similarity to the question, making existing query ranking models often make erroneous judgments when dealing with them. Building on this issue, the authors proposed an approach that features two stages. The first stage uses a generative neural-based model to produce an *Abstract Query Graph* (AQG) that models the query graph. An AQG is a tree which consists of a combination of n labelled vertices $v \in \{\text{“}Ent''$, “$Type''$, “Num'', “$Var''\}$, connected with $n-1$ labelled, undirected edges $e \in \{\text{“}Rel''$, “Ord'', “Cmp'', “Cnt'', “$Isa''\}$ [28]. The AQG serves as a constraint to generate the candidate query graphs whose structures match it, thus mitigating the generation of many potentially noisy graphs. The generative model is a sort of seq2seq architecture that takes a question and an empty graph as the inputs and outputs a sequence of graphs where the last one is the completed AQG. The model first encodes the question using a standard Bi-LSTM encoder. Then, the encoded question supports a Graph Neural Network (GNN) [65] that aims to learn the vector representation of the graph generated at timestep $t-1$. The GNN is followed by a decoder implementing a Bi-LSTM network that translates the question and graph information at $t-1$ into a vector at each time step. The final component of the generative model, the Executor, uses the decoder output to predict the element label that serves to perform one of three operations: (i) add a fresh vertex, (ii) select a vertex to be connected to the fresh vertex, and (iii) add an edge. The operations are executed one per timestep in the order specified before. The final step of the first stage is generating the graph candidates by grounding on AQG according to the entity linking results obtained using an existing system. The second stage of this approach leverages existing models to perform a candidate query ranking to select the most suitable one. This final candidate allows for producing the final question results. The experiments show that the approach reached state-of-the-art results on LC-QuAD while obtaining comparative results on ComplexQuestions and WebQuestions datasets. It is curious how the system performs well in a complex dataset like LC-QuAD but not so well on a simpler one like WebQuestions. Also, it can be interesting to evaluate the impact of other work embedding approaches like fastText or BERT since GloVe cannot deal with OOV words.

This approach is substantially different from the one proposed in this thesis from a philosophical point of view. As explained above, the system relies on generating and ranking query graphs. However, the two approaches still have a point in common: they encode the semantic information in the question using a Bi-LSTM network.

A well-known issue of the graph ranking approaches is the effort involved in creating a query since they generate a considerable number of candidate graphs

(204 on average for LC-QuAD in this case) that have to be successively sorted by similarity. This complexity is supposed to be avoided by employing an NMT-based approach like *SPARQL-QA*. On the other hand, Chen et al. confirm that semantic information extraction using RNN (LSTM/Bi-LSTM) with attention architectures is still a valid approach.

Kapanipathi et al. [62] proposed a system called NSQA that combines Abstract Meaning Representation (AMR) graphs and Logical Neural Networks (LNN) to face the problem of KBQA. In this system, the *question understanding* phase relies on semantically parsing the question into an AMR graph that helps to reduce the complexity and noise of natural language questions. An AMR parse is a rooted, directed, acyclic graph where each node represents a concept, while edges represent relations between concepts [62]. The concepts may include normalized surface forms, Propbank frames [63] and other AMR-specific constructs to handle named entities. The edges, instead, include OntoNotes[3] roles and AMR-specific relations such as polarity or mode. The AMR parser used by Kapanipathi et al. utilizes a stack-Transformer transition-based model, which also allows for aligning explicit question text to AMR nodes. The *logic parsing* phase in this approach converts the AMR graph into a query graph aligned with the underlying knowledge base. The first step in the conversion is to enrich the AMR graph with the links to entities in the KB. To link the entities, the authors trained a BERT-based NER model and used BLINK [121] for entity disambiguation. Based on the AMR node-text alignment information, the linked entities are then associated with the AMR. In the second step, NSQA transforms the AMR into its corresponding query graph. Although the AMR is similar to the KB structure, some challenges must be addressed during the transformation. The first one is that AMR can contain nodes that are concepts or PropBank predicates, which can correspond to both entities and relationships, while in the query graph, nodes primarily correspond to entities, and edges correspond to relationships. The second one arises because it is possible for multiple predicates and concepts in the AMR to jointly represent a single binary relation in the query graph due to the underlying KB uses an entirely different vocabulary. To address such challenges, the authors proposed a Path-based Graph Transformation algorithm. Finally, the system uses an existing relation-linking system to link the predicates in the graph to those in the KB. The relation linking system takes in the question text and AMR predicate as input and returns a ranked list of KB relationships for each triple, allowing the creation of a list of candidate query graphs. The top-ranked query graph is selected as the final graph and represents the WHERE clause of the final SPARQL query. Kapanipathi et al. decided to translate the obtained graph to an intermediate representation

[3]https://catalog.ldc.upenn.edu/LDC2013T19

using existential First-Order logic (FOL) instead of using SPARQL, arguing that it enables the use of any FOL reasoner and makes NSQA compatible with reasoning techniques beyond the scope of typical KBQA. In the FOL representation, the non-logical symbols consist of the binary relations, entities in the KB and some additional functions to represent SPARQL query constructs such as ASK, SELECT and COUNT. Finally, the authors used the neuro-symbolic reasoner Logical Neural Network (LNN), which takes the FOL forms as input and has access to the KB to provide an answer to the question. During the experimental assessment, NSQA was executed on QALD-9 and LC-QuAD datasets and demonstrated to perform well, reaching state-of-the-art results.

This approach distinguishes itself by being the first system to employ AMR graphs to deal with the KBQA problem and by proposing a modular architecture that does not require end-to-end training data since each module is trained for its specific subtask. However, the accuracy of producing the AMR is still not satisfying on complex questions, especially for those with long-distance dependency; thus, the errors can propagate through the system pipeline and cause mistakes in generating the correct answer. Moreover, this approach may be unable to deal with implicit predicates, as the parsing relies on the question surface. Like the approach by Chen et al., NSQA is conceptually different from the system presented in this thesis but still incorporates some good ideas that can be included in *SPARQL-QA*, such as a Relation Linking system to help improve predicate detection.

The approach ElNeuQA [37] presented by Diomedi et al. is currently the most similar to *SPARQL-QA*. Both systems share the idea that pure NMT approaches suffer from the problem of OOV words, mainly related to the vast number of possible entities not covered in the training datasets. Thus, ElNeuQA and *SPARQL-QA* leverage an NMT model to predict a SPARQL query template with entity URIs replaced by placeholders and then use a NER model to detect entity mentions in the question. The mentions are linked to the KB and used to instantiate the former template and obtain the final query. The main differences rely on how the modules in each system are implemented and interact with each other.

Diomedi et al. experimented with three different NMT architectures, i.e., ConvS2S, LSTM, and Transformer, with the ConvS2S model showing the best performance. For the NER part, named *Sequence Labelling* in ElNeuQA, the authors decided to use a BiLSTM-CRF model similar to the one used in this thesis. At this point, there are two main differences between these two approaches. The first one is that *SPARQL-QA* uses a BiLSTM-based architecture for the NMT model that enables a joint trained with the NER model by sharing the sentence encoder, while in ElNeuQA, the models learn separately. The second difference concerns the NER approach, i.e., the Diomedi et al. approach produces labels that indicate entities and their disambiguated roles, e.g. object or subject in the

SPARQL triples; *SPARQL-QA*, instead, uses the BIO notation without the notion of the role. The roles of the entities in ElNeuQA match the placeholders found in the query template; for instance, the role obj_2 will match a placeholder $\langle obj\_2 \rangle$ in the query template.

ElNeuQA also proposes an ensemble Entity Linking system that relies on four different EL systems. The ensemble EL system includes a voting system where, for a given mention, the output of each EL system lends a vote to the top-scored entity for that mention. In the case of *SPARQL-QA*, the linking is made by considering the overlapping among the mentions and the entities detected by the individual EL systems (see Section 4.3.1). During instantiation, ElNeuQA looks for the placeholders in the template and tries to find a mention in the question that matches the placeholder role. Then, the ensemble EL system links the mention to the KB, and the obtained URI is used to replace the corresponding placeholder in the query template. *SPARQL-QA*, on the other hand, follows an order-oriented approach, as described in section 4.3. The system was tested on the LC-QuAD 2.0 dataset and a self-created dataset called WikidataQA, demonstrating the usefulness of the entity linking system for dealing with the OOV problem in NMT-based approaches for KBQA. As Diomedi et al. considered a different set of datasets to those investigated in this thesis, it is impossible to establish a direct comparison between the two approaches. However, it can be an aspect to be considered in the future.

Other related lines of work that aim to translate natural language queries into SQL queries and BASH commands were also studied for completeness.

In the Seq2SQL [134] approach, an LSTM Seq2Seq model is used to translate from natural language to SQL queries. In this approach, Zhong et al. [134] introduced the interesting idea of guiding the learning process using a *Reinforcement Learning* agent. Yu et al. [129] introduce a large-scale, complex and cross-domain semantic parsing and text-to-SQL dataset to train different models to convert text to SQL queries. Most of the models used to establish a baseline for this dataset were based on a Seq2Seq architecture with attention, demonstrating an adequate performance. Another interesting approach for text-to-SQL generation was introduced by Zhang et al. [132]. They implement a Seq2Seq model with Luong's attention, using BiLSTMs and BERT embeddings. The approach performs well on the SParC and Spider datasets, outperforming the related work in some cases.

Lin et al. [72] address the topic of natural language to BASH commands translation. They propose NL2Bash, a new dataset in the ambitious domain of controlling the operating system using natural language, containing more than 9000 instances. To demonstrate that NL2Bash is challenging and establish a baseline for it, the authors applied and evaluated three approaches, (*i*) Seq2Seq model, (*ii*) CopyNet, and (*iii*) Tellina, demonstrating decent performances. In particular, the OOV problem is not taken into account in this approach.

Summary In the literature, there are many strategies to address the problem of Knowledge Base Question Answering, implementing a variety of approaches ranging from symbolic to sub-symbolic ones. State-of-the-art was recently advanced to an impressive level by the latest neural machine translation approaches, and the architecture for QA evolved to incorporate modules and solutions to many subproblems. The performance of systems is, thus, improving, and it still makes sense to build on the idea of KBQA based on NMT. In particular, it can be observed that many existing approaches based on NMT basically do nothing special to deal with out-of-vocabulary words. Although it is a very sensitive issue when dealing with large and evolving knowledge bases, the problem of adequately handling words that are out of the vocabulary of the training set received less or no attention. On the other hand, this thesis presents an approach that builds on the ideas used by NMT-based systems and adds techniques specifically conceived for approaching the OOV problem. Notably, this thesis proposes an architecture that addresses many of the issues connected with translation (e.g., acronyms, entity linking) by combining dedicated tools in a novel way. The capability of handling OOV words, resulting in a better capability of generalizing what it learns from the training set, makes the proposed approach innovative with respect to existing works.

Chapter 7

Conclusion

Significant advances in many areas of computer science, such as the Semantic Web, have led to the rise of new ways of storing and organising data, providing the information with semantic meaning and making it easier to share. As a result, it has been possible to gather large volumes of information through the interconnection of thousands of knowledge bases from different domains, giving rise to what is known as Linked Data. Knowledge bases such as DBpedia and Wikidata are evidence of the potential of the semantic web, making a priceless amount of knowledge freely available to society. Unfortunately, accessing these sources of information has been a privilege reserved for specialists in the field who know the appropriate languages for interrogating knowledge bases. For this reason, several research works have proposed to apply Question Answering techniques to interrogate knowledge bases through natural language, thus allowing non-expert users to access information easily.

In order to contribute to addressing the knowledge base accessibility issues, this thesis presented a novel QA system based on deep neural networks to query knowledge bases using natural language. The system builds on practical ideas presented in the literature but addresses an issue that can hinder the application of state-of-the-art KBQA implementations in real-world scenarios: the inability to deal with the OOV words. To this end, the proposed approach resorts to several well-known NLP tools like *Neural Machine Translation*, *Named Entity Recognition* and *Entity Linking*, combining them in an original way to overcome the issues. Specifically, the novelties of the presented system include:

- a novel RNN architecture where translation and NER are co-trained, thus saving training time and improving performance.
- a training set format that reduces the output vocabulary and is essential for tackling OOV words.

- a collision-named entity linking that supports the correct entity identification during the assembly phase.

In *SPARQL-QA*, the process of answer a given user question consists of three main phases. The *Input Preparation* phase applies several transformations to the question, allowing it to be analysed better by the deep learning models. Subsequently, during the *Understanding* phase, the system translates the question into a non-instantiated SPARQL query template and identifies the entities involved in the text. Finally, using both the query template and the identified entities, the *Assembling* process creates a query that can be executed on the KB to obtain the answers to the user's question.

During the experiments, the system showed effective results on the Monument, QALD-9, QALD-10, and LC-QuAD v1 publicly available datasets and demonstrated a more general and robust behaviour on unseen questions among the compared systems. Moreover, it became clear how crucial it is to have a quality data set in order to achieve good results in machine learning tasks. A small dataset such as QALD-9 can compromise the ability of a machine learning model to generalise correctly, as could be seen during the experiments (see Section 5.1.4). On the other hand, many large datasets (e.g., LC-QUAD v2) are very noisy, affecting the pre-processing and tokenisation phases and, consequently, the training of the models. In these cases, correcting these problems requires a great deal of time and effort.

It is worth mentioning that the approach presented in this thesis won the tenth Question Answering over Linked Data (QALD) Challenge held in May 2022, demonstrating that it can achieve state-of-the-art performance on renowned datasets.

Future work Despite the results obtained during the experimentation, there is still room for improvement; for this reason, it is planned to experiment and integrate other NLP tools that may lead to more accurate answers to users' questions. Thus, some studies are ongoing to integrate a Relation Linking system, specifically the one presented by Barbara et al. [18], to support the NMT model to find the appropriate KB predicates since this represents one of the most frequent problems during translations. Similarly, it is intended to use the Transformer architecture in the future, although this would mean some substantial changes to the current proposed architecture.

Bibliography

[1] OWL. https://www.w3.org/OWL/, 2004. Web Ontology Language (OWL).

[2] OWL 2 Web Ontology Language. https://www.w3.org/TR/owl2-overview/, 2012. OWL 2 Web Ontology Language.

[3] SPARQL 1.1 Query Language. https://www.w3.org/TR/sparql11-query/, 2012. W3C Recommendation 21 March 2013.

[4] RDF 1.1 N-Triples. https://www.w3.org/TR/n-triples/, 2014. W3C Recommendation 25 February 2014.

[5] RDF 1.1 XML Syntax. https://www.w3.org/TR/rdf-syntax-grammar/, 2014. W3C Recommendation 25 February 2014.

[6] RDF Schema 1.1. https://www.w3.org/TR/rdf-schema/, 2014. W3C Recommendation 25 February 2014.

[7] Resource Description Framework (RDF). https://www.w3.org/RDF/, 2014.

[8] Terse RDF Triple Language. https://www.w3.org/TR/turtle/, 2014. W3C Recommendation 25 February 2014.

[9] 10th Question Answering over Linked Data (QALD) Challenge @ ESWC 2022. https://www.nliwod.org/, 2022.

[10] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[11] Abdalghani Abujabal, Rishiraj Saha Roy, Mohamed Yahya, and Gerhard Weikum. Never-ending learning for open-domain question answering over knowledge bases. In *Proceedings of the 2018 World Wide Web Conference*, pages 1053–1062, 2018.

[12] Abdalghani Abujabal, Mohamed Yahya, Mirek Riedewald, and Gerhard Weikum. Automated template generation for question answering over knowledge graphs. In *Proceedings of the 26th international conference on world wide web*, pages 1191–1200, 2017.

[13] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003.

[14] Juan A. Alonso and Gregor Thurmair. The comprendium translator system. In *Proceedings of Machine Translation Summit IX: System Presentations*, New Orleans, USA, September 23-27 2003.

[15] Franz Baader, Diego Calvanese, Deborah McGuinness, Peter Patel-Schneider, Daniele Nardi, et al. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[16] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[17] Nikita Baramiia, Alina Rogulina, Sergey Petrakov, Valerii Kornilov, and Anton Razzhigaev. Ranking approach to monolingual question answering over knowledge graphs. In *NLIWoD7 2022*, volume 3196, pages 32–37. CEUR-WS.org, 2022.

[18] Vito Barbara, Manuel Borroto, and Francesco Ricca. A sequence to sequence approach for knowledge base relation linking. In *NL4AI@ AI* IA*, 2021.

[19] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. on Neur. Net.*, 5(2):157–166, 1994.

[20] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227. IGI global, 2011.

[21] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.

[22] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250, 2008.

[23] Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 858–867, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[24] Eric Brill, Susan Dumais, and Michele Banko. An analysis of the askmsr question-answering system. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 257–264, 2002.

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

[26] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[27] Mikhail Burtsev, Alexander Seliverstov, Rafael Airapetyan, Mikhail Arkhipov, Dilyara Baymurzina, Nickolay Bushkov, Olga Gureenkova, Taras Khakhulin, Yurii Kuratov, Denis Kuznetsov, et al. Deeppavlov: Open-source library for dialogue systems. In *Proceedings of ACL 2018, System Demonstrations*, pages 122–127, 2018.

[28] Yongrui Chen, Huiying Li, Yuncheng Hua, and Guilin Qi. Formal query building with query structure prediction for complex question answering over knowledge base. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 3751–3758. ijcai.org, 2020.

[29] David Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd annual meeting of the association for computational linguistics (acl'05)*, pages 263–270, 2005.

[30] Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Frederick Reiss, and Shivakumar Vaithyanathan. Domain adaptation of rule-based annotators for named-entity recognition tasks. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1002–1012, 2010.

[31] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. Association for Computational Linguistics, 2014.

[32] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2014.

[33] Francois Chollet. *Deep learning with Python*. Manning Publications Company, 2017.

[34] J. Daiber, M. Jakob, C. Hokamp, and P. N. Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *ICSS (I-Semantics)*, 2013.

[35] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, June 2019.

[36] EDDY MUNTINA Dharma, F Lumban Gaol, H Leslie, HS Warnars, and B Soewito. The accuracy comparison among word2vec, glove, and fasttext towards convolution neural network (cnn) text classification. *Journal of Theoretical and Applied Informtion Technology*, 100(2):31, 2022.

[37] Daniel Diomedi and Aidan Hogan. Question answering over knowledge graphs with neural machine translation and entity linking. *CoRR*, abs/2107.02865, 2021.

[38] Bonnie J Dorr, Pamela W Jordan, and John W Benoit. A survey of current paradigms in machine translation. In *Advances in computers*, volume 49, pages 1–68. Elsevier, 1999.

[39] Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In *International semantic web conference*, pages 69–78. Springer, 2019.

[40] David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

[41] Mikel L Forcada, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim O'Regan, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez, and Francis M Tyers. Apertium: a free/open-source platform for rule-based machine translation. *Machine translation*, 25(2):127–144, 2011.

[42] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR, 2017.

[43] Abbas Ghaddar and Philippe Langlais. Robust lexical features for improved neural network named-entity recognition. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1896–1907. Association for Computational Linguistics, 2018.

[44] Y. Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[46] Bert F Green Jr, Alice K Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, pages 219–224. ACM, 1961.

[47] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing? *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1995.

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

[49] Armin Haller, Axel Polleres, Daniil Dobriy, Nicolas Ferranti, and Sergio J Rodríguez Méndez. An analysis of links in wikidata. In *European Semantic Web Conference*, pages 21–38. Springer, 2022.

[50] Ann-Kathrin Hartmann, Edgard Marx, and Tommaso Soru. Generating a large dataset for neural question answering over the DBpedia knowledge base. 2018.

[51] Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. Scalable modified kneser-ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 690–696. Association for Computational Linguistics, August 2013.

[52] S. Hochreiter. Recurrent neural net learning and vanishing gradient. *Intern. Jour. Of Uncert., Fuzz. and KB Systems*, 6(2):107–116, 1998.

[53] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. Knowledge graphs. *ACM Computing Surveys (CSUR)*, 54(4):1–37, 2021.

[54] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4), jul 2021.

[55] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd. spaCy: Industrial-strength Natural Language Processing in Python, 2020.

[56] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible sroiq. *Kr*, 6:57–67, 2006.

[57] Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. Knowledge graph embedding based question answering. In *Proceedings of the twelfth ACM international conference on web search and data mining*, pages 105–113, 2019.

[58] Z. Huang, W. Xu, and K. Yu. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991, 2015.

[59] Yufan Jiang, Chi Hu, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. Improved differentiable architecture search for language modeling and named entity recognition. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3585–3590, Hong Kong, China, November 2019. Association for Computational Linguistics.

[60] James Joyce. Bayes' Theorem. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2019 edition, 2019.

[61] Dan Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2009.

[62] Kapanipathi et al. Leveraging Abstract Meaning Representation for knowledge base question answering. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3884–3894, Online, August 2021. Association for Computational Linguistics.

[63] Paul R Kingsbury and Martha Palmer. From treebank to propbank. In *LREC*, pages 1989–1993, 2002.

[64] Philipp Koehn, Franz J Och, and Daniel Marcu. Statistical phrase-based translation. Technical report, University of Southern California Marina Del Rey Information Sciences Inst, 2003.

[65] Rik Koncel-Kedziorski, Dhanush Bekal, Yi Luan, Mirella Lapata, and Hannaneh Hajishirzi. Text generation from knowledge graphs with graph transformers. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2284–2293. Association for Computational Linguistics, 2019.

[66] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, 01 2001.

[67] Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. A survey on complex knowledge base question answering: Methods, challenges and solutions. In Zhi-Hua Zhou, editor, *Proceedings of the IJCAI-21*, pages 4483–4491. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Survey Track.

[68] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[69] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. PyTorch-BigGraph: A Large-scale Graph Embedding System. In *Proceedings of the 2nd SysML Conference*, Palo Alto, CA, USA, 2019.

[70] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880. Association for Computational Linguistics, July 2020.

[71] Jimmy Lin. An exploration of the principles underlying redundancy-based factoid question answering. *ACM Transactions on Information Systems (TOIS)*, 25(2):6–es, 2007.

[72] Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D Ernst. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the LREC 2018*. European Language Resources Association (ELRA), 2018.

[73] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[74] Adam Lopez and Philip Resnik. Word-based alignment, phrase-based translation: What's the link? In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 90–99, 2006.

[75] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421. Association for Computational Linguistics, 2015.

[76] Fabiano Ferreira Luz and Marcelo Finger. Semantic parsing natural language into sparql: improving target language representation with neural attention. *arXiv preprint arXiv:1803.04329*, 2018.

[77] Gaurav Maheshwari, Priyansh Trivedi, Denis Lukovnikov, Nilesh Chakraborty, Asja Fischer, and Jens Lehmann. Learning to rank query graphs for complex question answering over knowledge graphs. In *International semantic web conference*, pages 487–504. Springer, 2019.

[78] Christopher D Manning. Part-of-speech tagging from 97% to 100%: is it time for some linguistics? In *International conference on intelligent text processing and computational linguistics*, pages 171–189. Springer, 2011.

[79] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Workshop Track Proceedings*, 2013.

[80] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.

[81] Alexander H Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1400–1409, November 2016.

[82] Mark A Musen. The protégé project: a look back and a look forward. *AI matters*, 1(4):4–12, 2015.

[83] Ngonga Ngomo. 9th challenge on question answering over linked data (qald-9). In *Semdeep/NLIWoD@ISWC*, volume 2241, pages 58–64, 2018.

[84] Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alexander Fraser, Shankar Kumar, Libin Shen, David A Smith, Katherine Eng, et al. A smorgasbord of features for statistical machine translation. In *Proceedings of the HLT-NAACL 2004*, pages 161–168, 2004.

[85] Christopher Olah. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Online accessed: 2022-12-02.

[86] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[87] Anand Panchbhai, Tommaso Soru, and Edgard Marx. Exploring sequence-to-sequence models for SPARQL pattern composition. In *Iberoamerican Knowledge Graphs and Semantic Web Conference*, pages 158–165. Springer, 2020.

[88] Marius Paşca. Open-domain question answering from large text collections, 2003.

[89] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[90] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018. cite arxiv:1802.05365Comment: NAACL 2018. Originally posted to openreview 27 Oct 2017. v2 updated for NAACL camera ready.

[91] Ruggero Petrolito and Felice Dell'Orletta. Word embeddings in sentiment analysis. In *CLiC-it*, 2018.

[92] Francesco Piccinno and Paolo Ferragina. From tagme to wat: a new entity annotator. In *Proceedings of the first international workshop on Entity recognition & disambiguation*, pages 55–62, 2014.

[93] C. Pradel, O. Haemmerlé, and N. Hernandez. Natural language query interpretation into sparql using patterns. In *Fourth International Workshop on Consuming Linked Data-COLD 2013*, volume 1034 of *CEUR Workshop Proceedings*, 2013.

[94] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[95] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[96] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[97] Ahmad Sakor, Kuldeep Singh, Anery Patel, and Maria-Esther Vidal. Falcon 2.0: An entity and relation linking tool over wikidata. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, CIKM 20, page 3141–3148, New York, NY, USA, 2020. Association for Computing Machinery.

[98] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[99] Igor Santos, Nadia Nedjah, and Luiza de Macedo Mourelle. Sentiment analysis using convolutional neural network with fasttext embeddings. In *2017 IEEE Latin American conference on computational intelligence (LA-CCI)*, pages 1–5. IEEE, 2017.

[100] Taihua Shao, Yupu Guo, Honghui Chen, and Zepeng Hao. Transformer-based neural network for answer selection in question answering. *IEEE Access*, 7:26146–26156, 2019.

[101] Kanchan Shivashankar, Khaoula Benmaarouf, and Nadine Steinmetz. From graph to graph: Amr to sparql. In *NLIWoD7 2022*, volume 3196, pages 38–42. CEUR-WS.org, 2022.

[102] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[103] Qi Song, Yinghui Wu, Peng Lin, Luna Xin Dong, and Hui Sun. Mining summaries for knowledge graph search. *IEEE Transactions on Knowledge and Data Engineering*, 30:1887–1900, 2018.

[104] T. Soru et al. SPARQL as a foreign language. *SEMANTiCS 2017 - Posters and Demos*, 2017.

[105] Amanda Spink, Dietmar Wolfram, Major BJ Jansen, and Tefko Saracevic. Searching the web: The public and their queries. *Journal of the American society for information science and technology*, 52(3):226–234, 2001.

[106] K. Stadler. The country converter coco - a python package for converting country names between different classification schemes. *The Journal of Open Source Software*, 2(16), aug 2017.

[107] N. Steinmetz, A. Arning, and K. Sattler. From natural language questions to SPARQL queries: A pattern-based approach. In *BTW*, volume P-289 of *LNI*, pages 289–308. GfI, Bonn, 2019.

[108] Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William W Cohen. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4231–4242. Association for Computational Linguistics, 2018.

[109] Yawei Sun, Lingling Zhang, Gong Cheng, and Yuzhong Qu. Sparqa: skeleton-based semantic parsing for complex questions over knowledge bases.

In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8952–8959, 2020.

[110] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[111] Tech With Tim. What is a neural network? `https://www.techwithtim.net/tutorials/python-neural-networks/what-is-a-nn/`. Online accessed: 2022-12-02.

[112] Peter Toma. Systran as a multilingual machine translation system. In *Proceedings of the third european congress on information systems and networks, overcoming the language barrier*, pages 569–581, 1977.

[113] Hayssam N Traboulsi. *Named Entity Recognition: A local grammar-based approach*. University of Surrey (United Kingdom), 2006.

[114] Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. Lc-quad: A corpus for complex question answering over knowledge graphs. In *International Semantic Web Conference*, pages 210–218. Springer, 2017.

[115] Dmitry Tsarkov and Ian Horrocks. Fact++ description logic reasoner: System description. In *International joint conference on automated reasoning*, pages 292–297. Springer, 2006.

[116] Johannes M van Hulst, Faegheh Hasibi, Koen Dercksen, Krisztian Balog, and Arjen P de Vries. Rel: An entity linker standing on the shoulders of giants. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2197–2200, 2020.

[117] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[118] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[119] Bin Wang, Angela Wang, Fenxiao Chen, Yuncheng Wang, and C-C Jay Kuo. Evaluating word embedding models: methods and experimental results. *APSIPA transactions on signal and information processing*, 8, 2019.

[120] William A Woods. Semantics and quantification in natural language question answering. In *Advances in computers*, volume 17, pages 1–87. Elsevier, 1978.

[121] Ledell Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. Scalable zero-shot entity linking with dense entity retrieval. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6397–6407. Association for Computational Linguistics, 2020.

[122] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[123] Yingce Xia, Tianyu He, Xu Tan, Fei Tian, Di He, and Tao Qin. Tied transformers: Neural machine translation with shared encoder and decoder. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 5466–5473, 2019.

[124] Wenhan Xiong, Mo Yu, Shiyu Chang, Xiaoxiao Guo, and William Yang Wang. Improving question answering over incomplete KBs with knowledge-aware reader. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4258–4264, Florence, Italy, July 2019. Association for Computational Linguistics.

[125] Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. LUKE: Deep contextualized entity representations with entity-aware self-attention. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6442–6454, Online, November 2020. Association for Computational Linguistics.

[126] Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 523–530, Toulouse, France, July 2001. Association for Computational Linguistics.

[127] Michihiro Yasunaga, Hongyu Ren, Antoine Bosselut, Percy Liang, and Jure Leskovec. QA-GNN: Reasoning with language models and knowledge graphs for question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 535–546, Online, June 2021. Association for Computational Linguistics.

[128] Xiaoyu Yin, Dagmar Gromann, and Sebastian Rudolph. Neural machine translating from natural language to sparql. *Future Generation Computer Systems*, 117:510–519, 2021.

[129] Yu et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921. Association for Computational Linguistics, 2018.

[130] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1091–1095, 2007.

[131] Hamid Zafar, Giulio Napolitano, and Jens Lehmann. Formal query generation for question answering over knowledge bases. In *European semantic web conference*, pages 714–728. Springer, 2018.

[132] Zhang et al. Editing-based SQL query generation for cross-domain context-dependent questions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5338–5349, November 2019.

[133] Biao Zhang, Deyi Xiong, Jinsong Su, Hong Duan, and Min Zhang. Variational neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 521–530, 2016.

[134] V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017.

[135] Shuguang Zhu, Xiang Cheng, and Sen Su. Knowledge-based question answering by tree-to-sequence learning. *Neurocomputing*, 372:64–72, 2020.