TECHNISCHE UNIVERSITÄT DRESDEN

FACULTY OF COMPUTER SCIENCE
INTERNATIONAL CENTER FOR COMPUTATIONAL LOGIC

Master Thesis

Master Computational Logic

Translating Natural Language to SPARQL

Xiaoyu Yin
(Born 13. June 1994 in Zhumadian, Mat.-No.: 4572954)

Supervisor: Dr. Dagmar Gromann

Dresden, December 5, 2018

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tag dem Prüfungsausschuss der Fakultät Informatik eingereichte Arbeit zum Thema:

Translating Natural Language to SPARQL

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den December 5, 2018

Xiaoyu Yin

Acknowledgments

% write more

I would first like to thank my thesis supervisor Dr. Dagmar Gromann. I could not have completed this thesis without her encouragement and supervision. She consistently allowed this paper to be my own work, but steered me in the right the direction whenever she thought I needed it.

I would also like to acknowledge Dr. Dmitrij Schlesinger as the second reader of this thesis, and I am gratefully indebted to him for his very valuable comments on this thesis.

Abstract

Time to write some abstract

Contents

1 Introduction

1.1 Motivation

The World Wide Web has been quickly evolving and has now become a huge network containing various kinds of resources for billions of users to interact with. A majority of the documents on the Web are formatted texts in Hypertext Markup Language (HTML), serving as the content that can be rendered in computer browsers for humans to read and understand. In order to find a resource satisfying specific needs, a Web user normally needs to rely on the help of search engines to retrieve and filter results from innumerable documents on the Web. Even so, it takes quite more time for humans to distinguish useful materials from others than machines that can scan a large number of files almost simultaneously. However, finishing such kind of tasks requires the capability of understanding the documents to be scanned and the traditional file format like HTML has made it difficult for machines to do so.

The Semantic Web is the concept of a Web where data and information can be manipulated by machines automatically [SHBL06]. There has been a set of standards for altering the current Web to be more machine-readable and processable for artificial agents. In order to help achieve the potential of the current Web, a series of relevant technologies including mainly Resource Description Framework (RDF) [CWL14] and Web Ontology Language (OWL) have been introduced. With the help of these tools, an increasing number of documents containing uniform organized data have been published conveniently on the Web. One notable example of this is cross-domain Linked Datasets such as DBpedia [ABK+07]. DBpedia contains RDF documents that represent information extracted from Wikipedia articles, and all the documents with links to other datasets on the Web constitute an interlinked ontology model. DBpedia also provides an open API for users to submit complicated queries against those documents.

SPARQL is a language designed for humans to query and manipulate the information sources contained in an RDF store or online RDF graph, and is by far the recommended standard [HS13]. SPARQL has already been widely supported by large open datasets like DBpedia. Though the data queried by SPARQL is made to be publicly and openly available, the use of it has yet been focused to a group of experts with prior knowledge specific to some certain domain. For example, if a user wants to ask for a list of books belonging to some certain category, he or she needs to have prior knowledge about

the concepts and relations involved in describing books in RDF. The root of this problem is the gap between the natural language used by non-experts and the query language that consists of unique syntax, semantics and domain-specific vocabulary.

The motivation of this thesis is to bridge this gap between natural language and SPARQL. Since natural language and SPARQL can both be represented as sequences of pre-defined tokens, this can be treated as a translation problem. In recent years, the application of neural networks in machine translation has achieved greater improvements in translation results than previously applied statistical and phrase-based methods [MWN17]. Therefore, we want to transfer the success achieved by neural network models in translating from one natural language to another to the task of translating natural language to queries written in SPARQL. We look into the effects of such transfer by conducting several experiments, and comparing between different models with varying network configurations.

1.2 Thesis Outline

Chapter 2 presents the notion of Semantic Web and its corresponding technologies, research on the subject of neural machine translation in the area of deep learning, and past work closely related to this thesis.

Chapter 3 further describes the neural network models involved in this thesis and the evaluation metrics used for the experiments.

Chapter 4 specifies the experiments carried out, including the details of involved datasets, experimental setups for the training and testing of the models, and the environments of the software and hardware.

Chapter 5 demonstrates the experiment results in graphs and tables, and proposes some analysis and discussion based on various aspects of the results.

Chapter 6 gives a summary and provides an outlook for the future work.

2 Background

This chapter gives an introduction to the background technologies and subfields involved in this thesis. First, Semantic Web technologies are briefly introduced in Section 2.1, including the notion of RDF in Section 2.1.1, Linked Data in Section 2.1.3 and SPARQL in Section 2.1.2. Second, Section 2.2 describes the notion of neural machine translation and involved models and components. Finally, related research is discussed in Section 2.3.

2.1 Semantic Web Technologies

The original Web consisted largely of documents made up of hypertexts for rendering in the browsers, the meanings of the web page are not well conveyed, thus being difficult for computers to analyze, or users to make higher-level searches. As Berners-Lee et al. stated in [BLHL01], the Semantic Web is not an individual Web separate from the current one but an extension. In the Semantic Web, there is an important functionality that the machines are able to process data and information automatically. The semantics of web pages are well-encoded and displayed to the software agents owned by users or organizations to provide meaningful services. In the Semantic Web, agents from different sources, namely producers and consumers, are able to communicate with each other by exchanging an ontology which typically contains a taxonomy and a set of inference rules. With the ontology, the computers can define classes, subclasses, and relations among entites on the Web and perform automated reasoning as if they "understand" the information [BLHL01].

The World Wide Web has linked more than 10 billion websites, and the useful contents can be delivered almost instantaneously to the users through search engines. Meanwhile, it is evolving from the web of documents for humans to read to a web of data and information derived from shared semantics. There are various types of programs and intelligent agents around the Web handcrafted for particular tasks, however, they usually possess little ability to deal with heterogeneous types of information [SHBL06]. There is also a growing need for the integration of data and information, especially in areas that demand heterogeneous and diverse datasets originating from separate subfields [SHBL06].

A set of technologies are already here to provide a preliminary environment for transforming the current

Web into the Semantic Web. Figure 2.1 shows an illustration of the Semantic Web technology stack, where the language in each layer is dependent on the layers below it. These languages have provided a foundation for allowing shared semantics to be integrated into the current documents on the Web, and data to be connected in a more explicit and standardized way. Resource Description Framework (RDF) [CWL14], a language located at a lower layer, has provided a foundation for the standardization of the formats of common data. SPARQL, on the other end, is a query language that can be utilized to search and manipulate data in RDF format from diverse sources [HS13]. The details of RDF and SPARQL are respectively presented in Section 2.1.1 and Section 2.1.2.

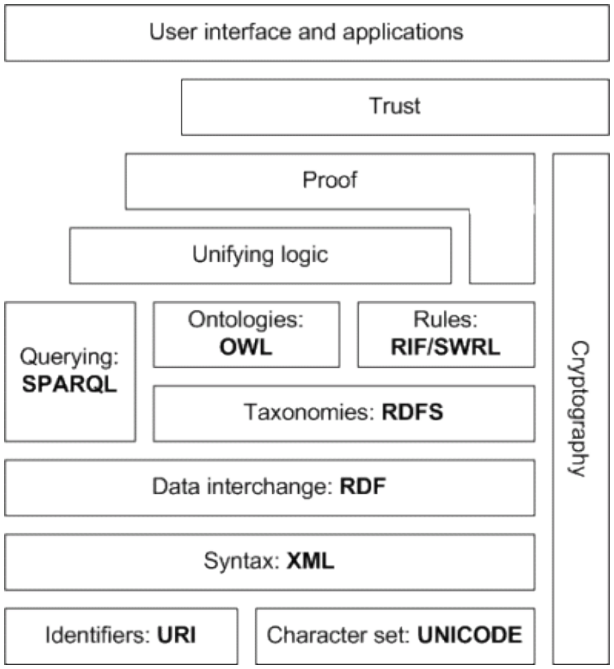


Figure 2.1: Semantic Web technology stack

The development of standardized technologies for the Semantic Web has promoted the integration of semantics into existing documents and linking of common data across different application domains and even regions. This enables a web of data where the data is connected with typed links, known as Linked Data [BHBL09]. It uses RDF to link arbitrary entites in the world by making typed statements, and allows complicated queries to be submitted in SPARQL. The applications of Linked Data are able to work upon a global and unbound data space, whereas the conventional web applications normally operate on top of a fixed set of data sources [BHBL09]. Further information on Linked Data is provided in Section 2.1.3.

2.1.1 RDF

The Resource Description Framework (RDF) is a representation language for defining information on the Web. The Semantic Web Technology Stack (Figure 2.1) provides the infrastructure to express the meaning of concepts and terms in an organized way that machines can easily process. In 1997, the first RDF specification was defined by the World Wide Web Consortium (W3C) and then it became a W3C recommendation in 1999. Currently, the latest version is RDF 1.1 [CWL14] published in 2014.

In RDF, meanings are expressed in triples. A set of triples constitutes an RDF graph, which can be visualized via a diagram containing nodes and directed arcs. Figure 2.2 demonstrates a simple RDF graph with merely two nodes and one arrow connecting them, which also indicates the three components of a triple: subject, predicate, and object. The subject and object represent some resource in the world, and the predicate denotes some relationship. Thus, an RDF triple makes assertions on some relationship of the things it identifies. An RDF graph claims the conjunction of statements encoded by its triples [CWL14].

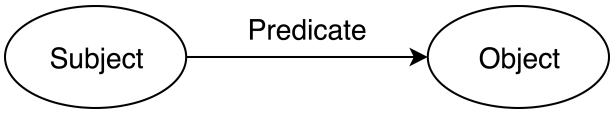


Figure 2.2: An RDF graph with one triple consisting of a subject, a predicate and an object [CWL14]

In the Semantic Web, arbitrary things in the world are denoted as resource and can be referred to by Internationalized Resource Identifier (IRI) or Literals. A datatype is used in addition to the Literal to specify its ranges of value. When no specific resources are identified but implicitly naming some relationship is needed, a blank node is used [CWL14]. In terms of an RDF triple, the following restriction is defined:

- the subject is an IRI or a blank node
- the predicate is an IRI
- the object is an IRI, a literal or a blank node

With the above-introduced notions, we show an example of how RDF can specify the concepts, properties, relations, and corresponding entities existing in the real world. We define the following IRIs[1]: <*http://example.org/bob#me*> referring to a human entity named Bob, <*http://www.w3.org/1999/02/22-rdf-syntax-ns#type*> representing "is a" relation, <*http://xmlns.com/foaf/0.1/Person*> specifying a concept denoting the class of person, <*http://schema.org/birthDate*> as a property representing date of birth, and <*http://www.w3.org/2001/XMLSchema#date*> referring to a data type of date. To express such in-

[1]The IRIs in this example are from `https://www.w3.org/TR/rdf11-primer`

formation: "Bob is a person who is born in 1990-07-04", the following triples are needed:

triple 1	subject	*<http://example.org/bob#me>*
	predicate	*<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>*
	object	*<http://xmlns.com/foaf/0.1/Person>*
triple 2	subject	*<http://example.org/bob#me>*
	predicate	*<http://schema.org/birthDate>*
	object	*"1990-07-04"^^<http://www.w3.org/2001/XMLSchema#date>*

It is noticeable that these IRIs do not share the same prefixes because in an RDF document concepts defined from different sources can be incorporated collaboratively. As a result, the existing knowledge sources are readily combined or extended with new information. Joining the above triples and replacing some IRIs with shorter prefixes we obtain a graph shown in Figure 2.3.

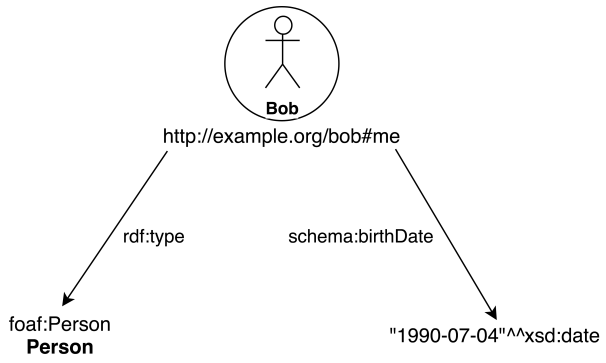


Figure 2.3: An example RDF graph representing "Bob is a person who is born in 1990-07-04". Some absolute IRIs are edited to be relative with prefixes.

RDF graphs provide an abstract representation for the knowledge. To write it down in a document, there exists a variety of serialization formats where each of them are designed for different scenarios and purposes. The most used ones are: Turtle [PC14], JSON-LD [SKL14], RDFa [HASB13], and RDF/XML [GS14]. Referring to the same RDF graph, the documents written in different serialization formats are logically equivalent [SR14]. The data owners can choose from these serialization formats to convert their existing documents into or publish new documents in RDF.

2.1.2 SPARQL

SPAQRL Protocol and RDF Query Language (SPARQL) is a structured language similar to SQL but for querying and manipulating RDF graphs [HS13]. This thesis primarily focuses on the query capability of SPARQL. A SPARQL query normally consists of a `SELECT` operator followed by queried objects which are usually denoted by variables or their combinations, a `WHERE` block containing graph patterns

for matching the RDF data store, and optionally extensional operators such as filtering and grouping depending on the requirements. A list of prefixes can be provided in the head for shortening the IRIs involved in the matching graph patterns. The results of SPARQL queries are usually result sets or RDF graphs.

For instance, to query the RDF graph exhibited in Figure 2.3, a simple SPARQL query with only one single variable and WHERE block can be formulated as in the Listing 2.1.

Listing 2.1: A SPARQL query asking for "the persons whose birthday is on 1990 July 4th"

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX schema: <http://schema.org/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?person
WHERE
{
  ?person rdf:type foaf:Person .
  ?person schema:birthDate "1990-07-04"^^xsd:date .
}
```

Note that in SPARQL, prefixes are mandatory in the head of the query to make IRIs resolvable into absolute forms if there are any abbreviations in the body. The query in Listing 2.1 simply asks for the persons whose birthday is on 4th of July 1990. Running it on Figure 2.3 leads to a result consisting of a single value which can be displayed in tabular form (see Table 2.1). The query results can also be exhibited in variable standardized formats including XML, JSON, CSV, and TSV for the benefits of exchanging results in miscellaneous environments [HS13].

person
<http://example.org/bob#me>

Table 2.1: Returned result of a SPARQL query on the RDF graph in Figure 2.3

Table 2.1 indicates that *<http://example.org/bob#me>* is the only entity that matches the conjunction of both triples in the query graph pattern. If it is likely to have multiple values in the result, one can control the sequence size of the result with `OFFSET` and `LIMIT` operators, and further specify the ordering by using `ORDER BY` operator. In addition, SPARQL supports more advanced operators[2] for expressing aggregation, negation, counting, etc. for performing queries with higher complexity against the given RDF data. Some of them are not mentioned here for the reason that it is so far difficult to include all the operators in this thesis. The operators covered in this thesis are listed in Section 4.1.

[2]Full capabilities of SPARQL is available at: `https://www.w3.org/TR/sparql11-query`

2.1.3 Linked Data

RDF as a common data format provides a sound foundation for the unification of information on the traditional Web. While a huge amount of data is available in a standard format, to create a Web of Data that is able to provide more meaningful services, the relationships between data are also important. The collection of interrelated datasets that contain links to each other on the Web are referred to as Linked Data [lin18]. Linked Data also refers to a set of best practices to publish, share, and connect the data, information, and knowledge on the Web [BHBL09].

Anyone can create Linked Data and publish it on the Web. There exists a number of ways to do so. One can simply put a static RDF file on a server, or publish relational databases with the help of dedicated tools. A list of tools for editing, publishing, and consuming Linked Data are already publicly available [ldt18]. For large datasets, a SPARQL query service is often needed as well to support broader use cases.

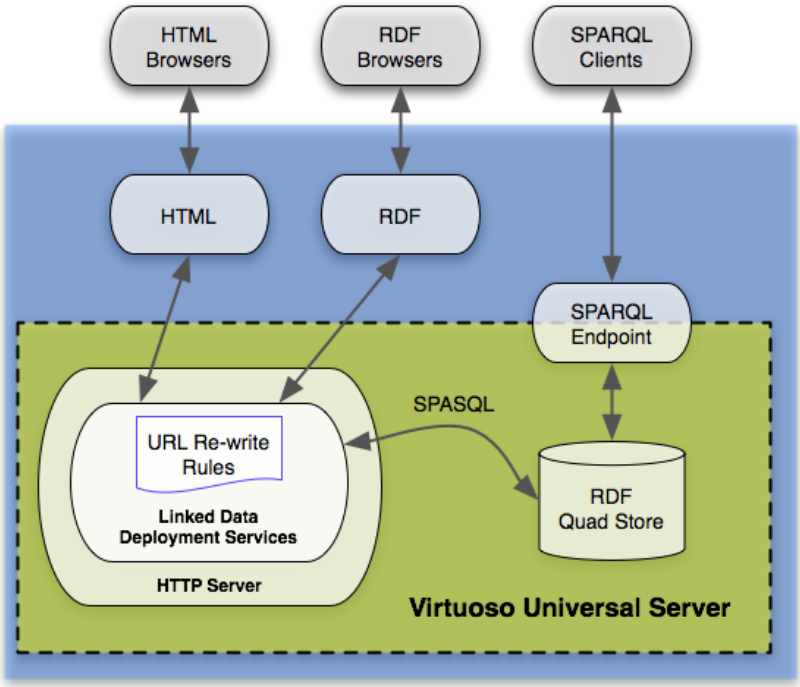


Figure 2.4: Current DBpedia data provision architecture [dbp18]

DBpedia is one of the most popular large-scale Linked Datasets online. It hosts datasets in RDF representing connected knowledge graphs that are generated by essentially parsing the articles of Wikipedia into structured information. The knowledge graph on DBpedia not only links the described concepts within Wikipedia but also connects the concepts with other external datasets existing on the Web. In addition, DBpedia is serving a public endpoint upon which applications can post queries in SPARQL to retrieve information in RDF triples for different kinds of purposes. Figure 2.4 illustrates the architecture

DBpedia is currently adopting for the provision of its RDF data set. In DBpedia, OpenLink Virtuoso Server[3] is used to provide support for SPARQL endpoint and HTTP hosting.

% Add some DBpedia use cases

2.2 Neural Machine Translation

As a global information space, the World Wide Web contains billions of web pages written in languages from various regions of the world. The problem arises when users face the contents they request in a different language from the one they are fluent in. Translation is therefore needed here to lower the language barrier. However, it is evident that human translation does not fit the requirements due to the large quantity of web documents. One alternative solution is Machine Translation (MT), which is a subcategory of Natural Language Processing (NLP). The goal of machine translation is to transform a text from an input language to a target language with the semantic meaning of the text being preserved.

However, due to the complexity of structures, semantics and vocabularies of natural languages, translation is considered a difficulty task for machines. According to [Pop12], the errors occuring in the machine translation output are mainly classified into five base categories: inflectional error, incorrect reordering, missing word, extra word, and lexical ambiguity. There is also debate whether fully automated high-quality machine translation systems can be achieved [BH64]. In addition, lack of context, incomplete common sense knowledge, and ineffectiveness in translating rare words have been major issues that affect the quality of the machine translation systems [Okp14] [WSC+16].

A large number of approaches have been developed in the area of machine translation over the last years. Currently, the architectures of existing MT systems can be divided into the following categories:

- **Rule-based Machine Translation** (RBMT). RBMT systems usually generate target language text based on an intermediary linguistic representation of the source text and a large set of rules that contain morphological, syntactic, and semantic bilingual mappings. They can be further subdivided into direct, transfer-based, and interlingua-based methods. The performance of RBMT systems, to a certain extent, relies on carefully designed linguistic rules and very large lexicons [MWN17].
- **Statistical Machine Translation** (SMT). SMT systems are developed on the basis of splitting a bilingual text corpus into respective source and translation text pairs. They apply machine learning algorithms that compute a statistical model from the given corpus and the model translates each phrase or word at a time based on a probability distribution [MWN17]. SMT approaches usually

[3]available at: `https://virtuoso.openlinksw.com/`

require an alignment between source sentence and several target sentences found in each text corpus and vice versa. Such methods suffer in performance when the languages involved have significantly different word orders [Okp14].

- **Example-based Machine Translation** (EBMT). EBMT utilizes bilingual corpora like SMT but they translate the text by example sentences. The major limitation of EBMT systems is translation of unknown words [MWN17].
- **Neural Machine Translation** (NMT). Some argue that NMT is also a statistical approach [MWN17]. NMT systems commonly consist of a model based on deep neural networks to perform end-to-end translation by words or characters in the given sentence [MWN17]. During the training of the model, the system steadily learns a representation of both languages in a continuous vector space and the ability to predict a combination of words with higher probability. The approaches in this category currently set the new state-of-the-art on several benchmark tests. Their relevant models are the primary focus of the investigation of this thesis.
- **Hybrid Machine Translation**. Hybrid approaches essentially leverage the advantages of the methods mentioned above to address their respective limitations and achieve better translation quality. In applications under this category, the hybridization of MT approaches are normally guided by either rule-based or corpus-based statistical systems [CJF15].

Among these categories, we focus primarily on the Neural Machine Translation methods. The development of NMT systems has gained more interests in recent years since deep neural networks have boosted extraordinary advancements in other areas of Artificial Intelligence such as computer vision [KSH12] and speech recognition [DYDA12]. NMT systems are usually superior in not needing hand-engineering of features that are one of the shortcomings of traditional machine learning systems [BGLL17]. However, some of the current NMT architectures have disadvantages in requiring large amounts of computation and time for training the deep model [BGLL17].

So far, many different architectures have been explored in NMT and new methods are constantly beating previous models in some benchmark datasets and achieving higher efficiency in computing. Among these architectures, primary works are listed here. Sutskever et al. [SVL14] and Cho et al. [CvMG$^+$14] proposed and deployed an encoder-decoder architecture that contains two models where Recurrent Neural Networks (RNN) (see Section 2.2.2) were used. The encoder encodes the input into a fixed-length vector, the decoder then decodes it into a translation. The two models are jointly trained to maximize the likelihood of a target sequence based on the given source sequence. However, the performance of this architecture drops when the length of the input sentence increases. To adress this issue, Bahdanau et al. and Luong et al. presented in [BCB14, LPM15] an attention mechanism which serves as an extension

to align the encoder and decoder. The acceptance of this mechanism increased the quality of translation significantly. Furthermore, there have been other improvement strategies like bi-directional RNN, beam search, etc. Some variants like Long-Short Term Memory (LSTM) [HS97] and Gated Recurrent Unit (GRU) [CvMG+14] versions of encoder-decoders have also been investigated. In the mean time, while RNN encoder-decoders consume a lot of computation time on their sequential learning, there are architectures based on Convolutional Neural Networks (CNN) that are able to achieve parallelized computations, thus outperform the former models at a faster speed [GAG+17a]. What's more, Vaswani et al. [VSP+17] proposed a self-attention model called the Transformer. This model shows both quality and speed advantages and has achieves state-of-the-art results on multiple translation tasks. In the past years, some novel paradigms for NMT have also emerged like the applications of Generative Adversarial Networks (GAN) in [WXZ+17, YCWX17].

2.2.1 Sequence to Sequence Learning

Sequence to sequence (Seq2Seq) learning [SVL14, CvMG+14] was an emerging area in supervised learning proposed for machine translation, and has been applied to many other sequential problems such as speech recognition, and text summarization. Learning the representation of one sequence and transformation of it into another sequence with different forms effectively and accurately is essentially the goal of Seq2Seq learning. The NMT approaches explored in this thesis belong to the category of sequence-to-sequence learning.

Regarding Seq2Seq learning on automated translation, encoder-decoder is one of the most applied architectures. In the architecture shown in Figure 2.5, the encoder receives the input as a sequence of tokens and turns it into a feature vector representing the input information in higher dimensionality space, the decoder then decodes that information from the vector in a way that enables generating another sequence of tokens as the output. When certain neural networks have been used in both encoder and decoder, this architecture has shown its superiority over the traditional phrase-based models in ease of extracting features, more flexibility with regard to the configuration of models, and better result accuracy [WSC+16].

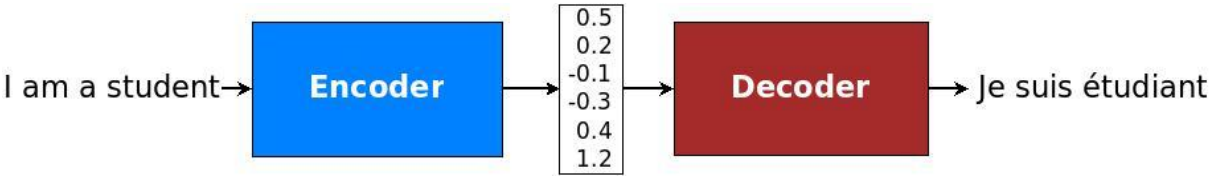


Figure 2.5: A conventional encoder-decoder architecture for machine translation [LBZ17]

In the above architecture, different kinds of neural networks in both encoder and decoder can be integrated to make combinations of models suitable for specific tasks. Recurrent Neural Networks are

mostly used in problems dealing with texts which normally have variable lengths. In the layers of RNN, various cell types, normally LSTM and GRU, can provide the model with capabilities of maintaining the past information and capturing long-range dependencies in texts. Convolutional Neural Networks are less commonly used in sequence modeling. However, they have started to appear in recently proposed models because of their inherent advantage in computation parallelization and effectiveness of building hierarchical representations.

Apart from the already mentioned components, a special attention mechanism is widely applied in Seq2Seq learning and even computer vision tasks that require models to attend to different parts of the input. In machine translation, an attention layer connects the decoder with the encoder hidden states to allow it search for parts of the source sequence instead of merely relying on a global vector. The attention mechanism has further improved the performance of NMT approaches by providing an effective mean to tackle the long-term dependency problem, although it also increases the computation.

2.2.2 Recurrent Neural Network

In sequence to sequence learning, Recurrent Neural Networks (RNNs) are the most widely used artificial neural networks. RNNs are specifically effective for processing sequential data. Given a sequence of inputs $x = (x_1, ..., x_{T_x})$, an RNN is able to compute a series of hidden states $h = (h_1, ..., h_{T_x})$ step-wise where at each step t the hidden state h_t is dependent on the previous hidden state h_{t-1} and input x_t by the following equation:

$$h_t = f(h_{t-1}, x_t) \tag{2.1}$$

where f is a non-linear activation function which may differ in distinct architectures (e.g. GRU and LSTM). Based on this set of hidden states h, a sequence of outputs $y = (y_1, ..., y_{T_y})$ can be generated. Given different configurations of input sequence length T_x and output sequence length T_y, RNNs can be adapted to cope with various tasks including image classification (one to one), image captioning (one to many), sentiment analysis (many to one), machine translation (many to many), and video frame labeling (many to many), all of which are shown in Figure 2.6.

In machine translation, the inputs and outputs are normally a sequence of words or characters from the source sentences and the target sentences. Therefore, RNNs are suited well in the encoder-decoder architecture (see Section 2.2.1). The RNNs served in the encoder and the decoder have different purposes. Figure 2.7 demonstrates a vanilla RNN encoder-decoder, where the encoder RNN reads the input words and encodes the information into the last hidden state vector, and then the decoder RNN receives this vector as the initial hidden state and samples a sequence word by word beginning from a starting symbol in the input until an ending symbol is produced in the output.

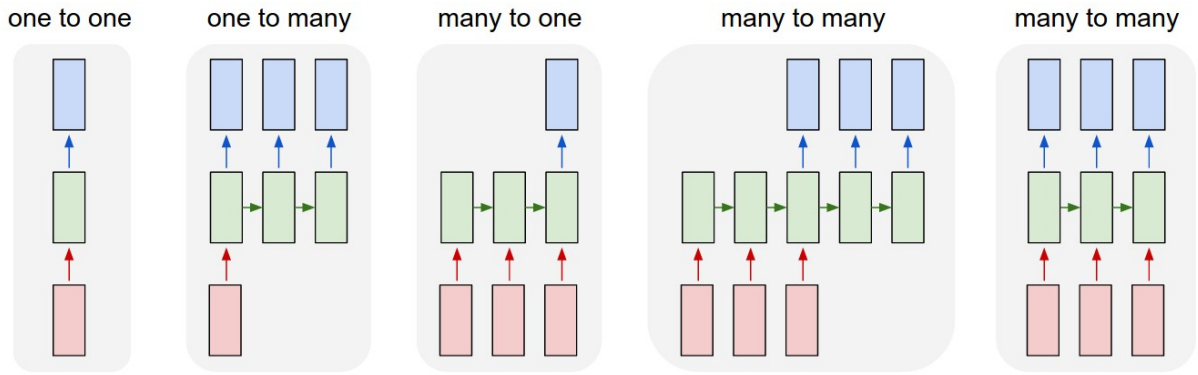


Figure 2.6: A list of RNNs (unfolded) mapping sequences to sequences with variant lengths [Kar15]. Each column of rectangles represents a step, the rectangle at the bottom, middle and top represent the input, hidden state, and output respectively at that step.

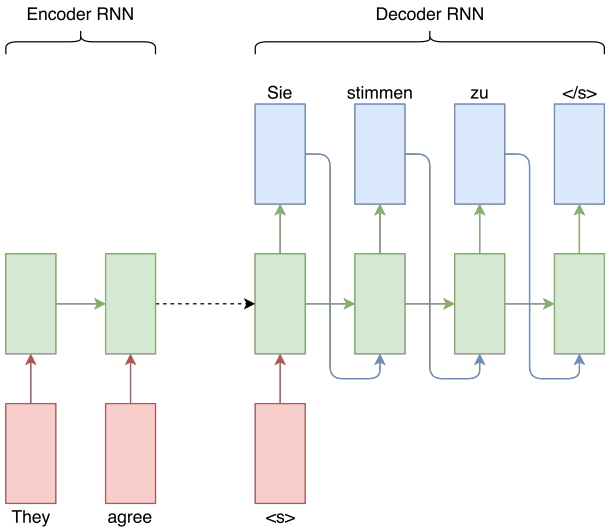


Figure 2.7: A vanilla RNN encoder-decoder

In addition to the primitive structure shown above, there are many techniques optimizing RNNs for various purposes. For example, the layers of hidden units can be stacked together, thus providing deeper representation of the information. Furthermore, residual connections [WSC+16] can be applied to deep layered RNNs to improve the gradient flow in terms of training. Bi-directional RNNs [SP97] were used in the encoder to reduce average dependency lengths.

2.2.3 Long Short-Term Memory

RNNs usually process information over a long duration. The ability of forgetting the old states once the information has been used becomes an important feature to RNNs. This is achieved by the so called gated RNNs, where the neural networks are able to gradually learn to decide when to clear the state relying

on the use of gated hidden units [GBC16]. Among different kinds of gated RNNs, Long Short-Term Memory (LSTM) [HS97] has been proved to be successful in many applications. LSTM-based models [WSC+16, SVL14] have been dominating the area of machine translation and achieving state-of-the-art results on many benchmark datasets.

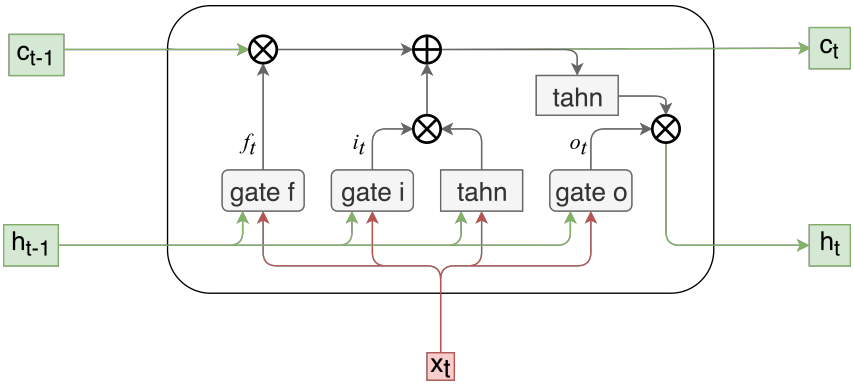


Figure 2.8: The diagram of the LSTM unit

Figure 2.8 reveals the structure of the LSTM unit in an RNN at time step t. Different from standard RNNs, a new vector which represents the cell state is used. There are also three gates, namely input gate i, forget gate f, and output gate o, each producing an activation vector. Suppose the number of input features is denoted by e and the number of hidden units is denoted by d. At step t, the input $x_t \in \mathbb{R}^e$, the previous cell state $c_{t-1} \in \mathbb{R}^d$, and the previous hidden state $h_{t-1} \in \mathbb{R}^d$ are already given information. In order to compute the current cell state $c_t \in \mathbb{R}^d$ and hidden state $h_t \in \mathbb{R}^d$, one needs to firstly compute three activation vectors:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

, then:

$$c_t = f_t \circ c_{t-1} + i_t \circ \tanh(W_c x_t + U_c h_{t-1} + b_c)$$
$$h_t = o_t \circ \tanh(c_t)$$

where the operator $\circ$ denotes the element-wise product and σ denote the sigmoid function. $W \in \mathbb{R}^{d \times e}$, $U \in \mathbb{R}^{d \times d}$, and $b \in \mathbb{R}^d$ denotes the internal weight matrices and bias vectors that are to be continuously updated during training phase. $f_t \in \mathbb{R}^d$, $i_t \in \mathbb{R}^d$, and $o_t \in \mathbb{R}^d$ are the activation vectors from corresponding sigmoid gates that control how much information is passed from the old states to the new ones.

2.2.4 Convolutional Neural Network

Compared to RNNs, Convolutional Neural Networks (CNNs) are more common for image processing tasks and have been less employed in machine translation despite their prominent advantages in feature extraction and faster training speed than the former.

In this thesis, CNNs are used in one of the investigated NMT models called Convolutional Sequence to Sequence (see Section 3.2.2), in which several convolutional layers are stacked upon the input and target sequences to extract hierarchical representations.

The convolutional operations in natural language processing are often one-dimensional instead of 2D when targeting the images. Figure 2.9 shows an example of a one-dimensional convolution layer, where

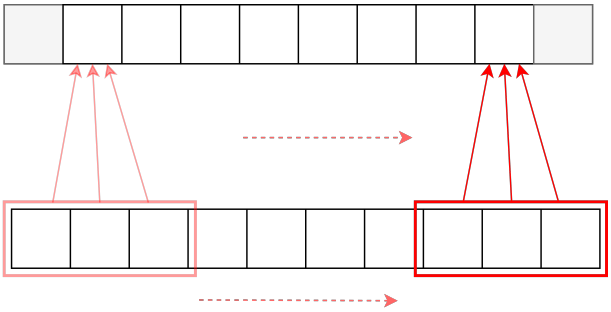


Figure 2.9: The illustration of a one-dimensional convolutional layer. The white squares are the input word embeddings or the output embeddings. The red rectangle and the dash arrows indicate that the convolution kernel is scanning over the sequence. The grey boxes placed in front of the output sequence represent the paddings which are usually zero vectors.

the size of the convolution kernel is $k = 3$, the length of the input is $m = 10$. The kernel maps a concatenation of k continuous input elements $x \in \mathbb{R}^{kd_x}$ to a single output element $y \in \mathbb{R}^{d_y}$, where d_x and d_y is respectively the dimension of the input embedding and output embedding. The parameters in the kernel are the weight matrix $W \in \mathbb{R}^{d_y \times kd_x}$ and bias $b \in \mathbb{R}^{d_y}$, and the output is simply computed by:

$$y = Wx + b$$

In practical situations, d_x and d_y are not essentially the same, as the output can be followed by a non-linearity to constitute a so-called gated convolutional neural network. In addition, in order for the CNN layers to be stacked to build deep hierarchical representations, before convolutions on each layer, the input of not enough length often needs to be padded.

2.2.5 Attention Mechanism

Attention mechanism [BCB14] is firstly proposed as an enhancement module for the basic RNN-based encoder-decoder architecture. In general, it gives the model abilities to conduct searching on parts of the source inputs at the time of inference. The conventional encoder-decoder models often perform worse when the length of the processed sentence is long, for the reason that the fixed-length vector produced by the encoder tends to lack some necessary information crucial for the decoder to predict accurate tokens. Attention mechanism addresses this issue by allowing the decoder to, at each decoding step, refer to a context vector calculated from weighted sum of a sequence of annotations to which the encoder maps the input sequence.

We hereby illustrate the concept of attention mechanism with a simplified version of the model proposed by Bahdanau et al. [BCB14] which utilizes GRU-based bi-directional encoder to obtain annotations from the input sequences instead of taking only hidden states along a single direction, which is shown in Figure 2.10.

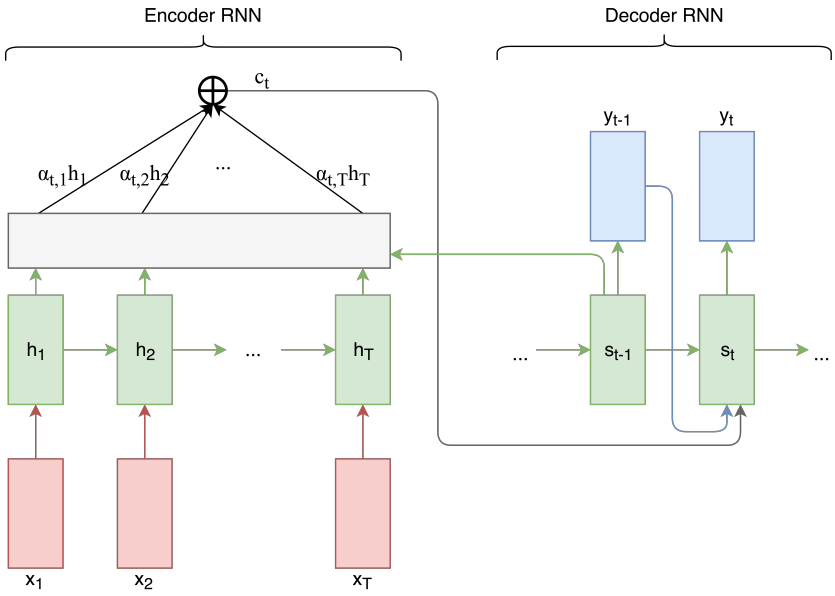


Figure 2.10: The t-th decoding step of a simplified encoder-decoder model with attention mechanism. The attention layer is placed on top of the hidden layer of the encoder.

Figure 2.10 demonstrates how a encoder-decoder model with attention generates the t-th target word y_t given a source sentence $x = (x_1, ..., x_T)$. The encoder maps x into a sequence of hidden states $h = (h_1, ..., h_T)$ by Equation 2.1. For the decoder, the hidden state s_t for step t is computed by

$$s_t = f(s_{t-1}, y_{t-1}, c_t)$$

where c_t is the context vector only for the t-th target word y_t. c_t is computed as a weighted sum of the

annotations of the input sequence (h in this case):

$$c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i$$

where $\alpha_{t,i}$ is the weight of each annotation h_i. It is based on a $score_{t,i}$ which is dependent on the previous decoder hidden state s_{t-1} and i-th annotation h_i. The equations for computing the score $e_{t,i}$ and the weight $\alpha_{t,i}$ is:

$$score_{t,i} = a(s_{t-1}, h_i)$$
$$\alpha_{t,i} = \frac{\exp(score_{t,i})}{\sum_{j=1}^{T} \exp(score_{t,j})}$$

where a is an alignment model usually parameterized as a feedforward neural network that can be jointly trained with other parts of the model. The $score_{t,i}$ is expected to evaluate how important the input around position x_i is with respect to the previous hidden state s_{t-1} in determining the next output y_t being generated. The final attention weights or the alignment model can be visualized into a graph that reflects the relationships between tokens of input sequence and output sequence as an example is shown in Figure 2.11.

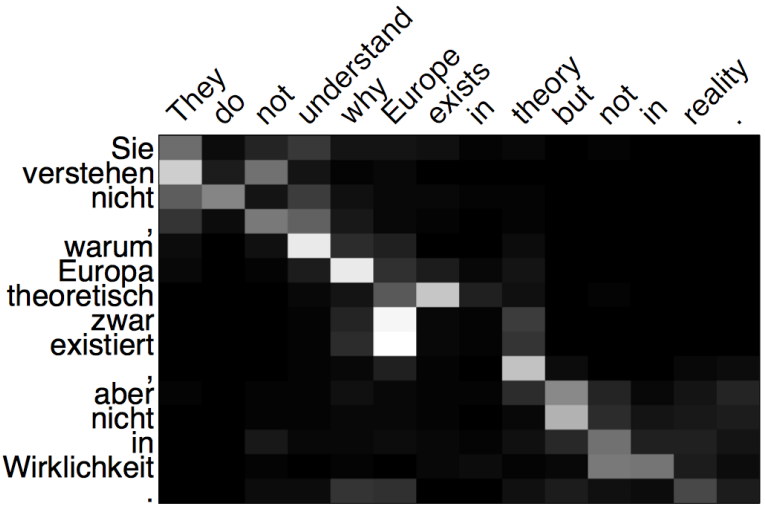


Figure 2.11: Visualization of the attention weights [LPM15]. Each block in the graph is related with the words positioned at the row and the column. Lighter blocks indicate stronger relations, i.e. bigger attention weights.

Although attention mechanism brings more preciseness to the models, it increases the training cost and may cause longer sequences impractical to translate. Therefore, in addition to the one described above which is referred to as global attention, there is local attention mechanism where in each decoding step only a subset of the whole source sequence is needed [LPM15].

2.3 Related Work

The primary focus of the investigation in this thesis is neural network models that can be used to map natural language statements to SPARQL expressions. Regarding the related work, we consider two main categories: the systems that integrate with NMT approaches and the systems not related to NMT. In the first category, we broaden the range on papers which have deployed machine learning methods to map unstructured sequences to structured sequences for the reason that we observe these methodologies sharing similar principles in learning the prediction of structured sequences like SPARQL. In terms of the second category, we narrow down the choices to those which translate natural language to SPARQL using merely non-NMT approaches on account of referencing their employed datasets, evaluation metrics, and experiment results for comparison.

2.3.1 Non-NMT Systems

In this section, the systems that did not involve neural computation are discussed. Most of the approaches investigated had a similar structure of two-step query composition, where the question provided by the user, either in a controlled natural language or not, needs to be adapted into an intermediary language representation which is lastly translated into a valid SPARQL expression.

SQUALL2SPARQL [Fer13] is a translator capable of producing SPARQL from a controlled English called Semantic Query and Update High-Level Language (SQUALL). Despite the limitation in the input side, this system is able to capture nearly full features of SPARQL 1.1. *SQUALL2SPARQL* achieved a high F-measure of 0.90 as well as very good precision and recall on QALD-3 challenge [CCL+13].

Pradel et al. [PHH13] presented an approach using an intermediary pivot query to interpret the natural language queries and then formalized into target formal queries like SPARQL. They implemented a system called *Semantic Web Interface using Patterns* (SWIP). The dataset they used primarily for evaluation was QALD-3 targeting the DBpedia knowledge base. An assessment of the system based on the measurements of precision, recall, and F-measure was conducted on it.

Xu et al. [XFZ14] proposed a question answering (QA) system that is able to recognize the intention and map the relevant semantic entities contained in a natural language query, which are used to further instantiate a formal structured query. They firstly extracted phrases from the query and then annotated the phrases with semantic labels in order to form a Directed Acyclic Graph (DAG) with phrase dependency relations. The DAG was lastly decoded into SPARQL query by mapping the phrases into corresponding entities in the knowledge base (KB). They experimented the system on 50 QALD-4 testing questions and achieved an F-measure of 0.71, which was a very competitive result at the QALD-4 challenge [UFL+14].

Dubey et al. [DDS+16] proposed a framework called *AskNow*, a QA system targeting DBpedia. This framework first transforms the questions in English to an intermediary common structure called Normalized Query Structure (NQS), which is later translated into a SPARQL query through a NQS parser and a SPARQL generator. The NQS plays a significant role in carrying the desire, the input information, and their mutual semantic relations in a query. From English query to NQS, they deployed the POS-tagger to retract tags of each tokens in the query and characterized the desire-input dependency relations through semantic analysis based on certain hypotheses. In NQS to SPARQL algorithm, they utilized DBpedia Spotlight [DJHM13] and WordNet [Mil98] for query annotation and entity matching, which essentially analyzes the tokens in the query and finds their synonymous entities in the vocabulary of DBpedia. In terms of evaluation, they tested four aspects including syntactic robustness, sensitivity to structural variation, semantic accuracy, and the accuracy of the whole system with precision and recall measurements, and the benchmark data sets QALD-4 [UFL+14] and QALD 5 [UFL+15].

2.3.2 NMT Systems

Cai et al. [CXY+17] proposed an enhanced encoder-decoder framework for the task of translating natural language to SQL, a similar query language to SPARQL but targeting relational databases instead of graph databases. They used not only BLEU [PRWZ02], but also query accuracy, tuple recall, and tuple precision for measuring the quality of output queries, and achieved good results.

Dong et al. [DL16] presented a method based on an encoder-decoder model with attention mechanism aimed at translating the input utterances to their logical forms with minimum domain knowledge. Moreover, they proposed another sequence-to-tree model that has a special decoder better able to capture the hierarchical structure of logical forms. Then, they tested their model on four different datasets and evaluated the results with accuracy as the metric.

Zhong et al. [ZXS17] proposed a framework called *Seq2SQL* for translating natural language questions to SQL. They took LSTM-based encoder-decoder networks as the core model. In order to leverage the structure of the SQL language, they augmented the input question with addition of column names of the queried table and split the decoder into three components, respectively predicting aggregation classifier, column names, and where clause part of a SQL query. As opposed to conventional teacher forcing, they trained the model with reinforcement learning to deal with the problem that queries that execute correct results but do not have exact string matches would be wrongly penalized. To address this issue in the evaluation, they performed analysis with measuring execution accuracy and logical form accuracy of generated queries.

Luz et al. [LF18] also used an LSTM encoder-decoder model but the purpose is to encode natural

language and decode into SPARQL. Furthermore, they employed a neural probabilistic language model to learn a word vector representation for SPARQL, and used the attention mechanism to associate a vocabulary mapping between natural language and SPARQL. For the experiment, they transformed the logical queries in the traditional Geo880 dataset into equivalent SPARQL form. In terms of evaluation, they adopted two metrics: accuracy and syntactical errors. They further compared their method with several other similar approaches [AlA15, KBZ06] and the comparison showed that they obtained better accuracy results. However, they did not deal with the "out of vocabulary" (OOV) problem and the issue of lexical disambiguation.

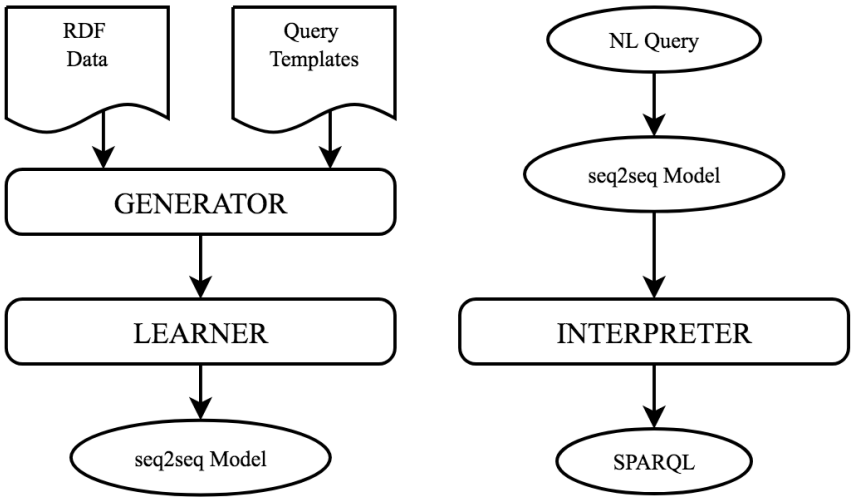


Figure 2.12: The generator-learner-interpreter architecture of *Neural SPARQL Machines* [SMM+18]

Soru et al. [SMM+18, SMV+18] proposed a generator-learner-interpreter architecture (see Figure 2.12), namely *Neural SPARQL Machines* (NSpM) to translate any natural language expression to encoded forms of SPARQL queries. They designed templates with variables that can be filled with instances from certain kinds of concepts in the target knowledge base and generated pairs of natural language expression and SPARQL query accordingly. After encoding operators, brackets, and URIs contained in original SPARQL queries, the pairs were fed into a sequence to sequence learner model as the training data. The model was able to generalize to unseen natural language sentence, and generate encoding sequence of SPARQL for the interpreter to decode.

3 Methodology

This chapter mainly describes the models of neural machine translation involved in this thesis for the task of translating natural language to SPARQL and related preliminaries. Then, a measurement BLEU that is typically utilized in automatic machine translation evaluation is described. We also clarify another metric called query accuracy which suits for our task specifically.

3.1 Preliminary

3.1.1 SPARQL Encoding

Unlike natural language, SPARQL queries are often written in structured forms instead of sequences that are easy to be tokenized. Therefore, the first step of training a Seq2Seq model with SPARQL is to convert SPARQL queries into sequences. To clarify, we encode the original SPARQL query into a sequential form which can be later decoded back.

We adopted the encoding approach in [SMM+18], which basically applies the following operations:

1. shorten the entities in the query with prefixes
2. replace the built-in symbols with dedicated words connected with underscores
3. shorten the query by replacing the built-in set phrases (e.g. ORDER BY) with one simplified word

These operations can be implemented as a set of replacements and applying them turns an original SPARQL query to a final sequence which contains tokens that are only formed of characters and underscores. An example is shown in Table 3.1. After training, as an encoded form of SPARQL translation is generated, it can be easily decoded back by reverse replacements.

3.1.2 Input

For the input of the NMT models, a sentence or paragraph can be normally separated into three kinds of sequences: characters, subwords, or words. Different methods of splitting sequences lead to differences in vocabularies that are seen by the model and its effectiveness of dealing with rare words. For the task

SPARQL	`SELECT DISTINCT ?uri`
	`WHERE {`
	`<http://dbpedia.org/resource/Sam_Loyd> <http://dbpedia.org/ontology/knownFor> ?uri .`
	`<http://dbpedia.org/resource/Eric_Schiller> <http://dbpedia.org/ontology/knownFor> ?uri .`
	`}`

Encoded	select distinct var_uri where brack_open dbr_Sam_Loyd dbo_knownFor var_uri sep_dot dbr_Eric_Schiller dbo_knownFor var_uri sep_dot brack_close

Table 3.1: A SPARQL query is encoded into a sequence of short tokens before treated as the input of the seq2seq model. Thus, the model learns how to translate from natural language texts into sequences of encoded form of SPARQL.

in this thesis, the texts are fed into the models in a word-based fashion on account of the reason that the words contained in the SPARQL vocabulary are mostly SPARQL operators and DBpedia entities that need to be treated as a whole.

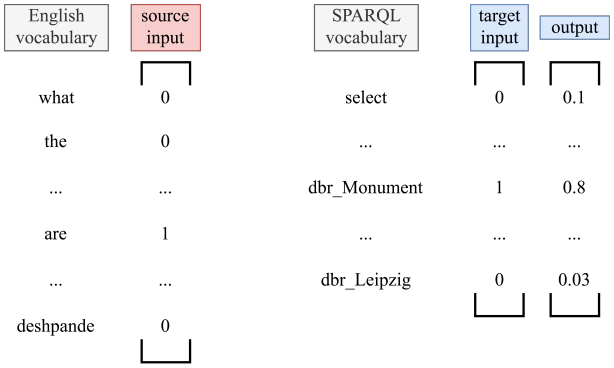


Figure 3.1: For our task, the source input (left) and target input (right) are one-hot encoding vectors representing word positions from respectively English and SPARQL vocabulary. The output vector is a probability distribution over all words in SPARQL vocabulary.

Each word in the sequences of both source input and target input is represented as an one-hot encoding vector, which is a sparse vector in which one element is set to 1 and all other elements are set to 0. The dimension of the one-hot encoding is equal to the size of the specific vocabulary. An example is shown in Figure 3.1.

Due to the use of one-hot encoding, every word in the vocabulary is orthogonal to each other. This does not reflect the relevance of some similar or distinct words such as man and king, king and queen. Therefore, in NMT models, the one-hot encoding vectors are usually further mapped into a low-dimensional

space where each word is represented as a dense vector called word embedding that holds floating-point values in each element. This mapping can be learned along with the training of the whole model or provided with a pre-trained word embedding matrix.

3.1.3 Output

The output of an NMT model is a sequence of vectors, where the dimension of each vector is the size of the vocabulary for the target language (see Figure 3.1). The sum of the entries in each vector is equal to 1, which means the vector is essentially a probability distribution over all the possible words in the target vocabulary.

The model usually generates the output vectors one by one, where each probability distribution is conditioned on its previous ones. During inference, one can then perform greedy or beam search over this sequence of probability distributions to generate a translation.

3.2 Models

In this thesis, we mainly focus on investigating three families of deep NMT models, including RNN-based models with an encoder-decoder architecture, the models entirely based on convolutional neural networks, and self-attention models relying on neither RNNs nor CNNs. We consider these three families covering the most advanced research of NMT and best-performing models so far, regardless of any other choices difficult to be classified such as hybrid models. Within each of these categories, one or more representative models are chosen and specified.

3.2.1 RNN-based Models

RNN is the most natural choice for the machine translation task because of its sequential structure. Prior research generally confirms that certain RNN-based models are superior to traditional statistical machine translation methods in translation quality, in spite of some weaknesses on expensive computation and issues of rare words.

There are many variants in RNN-based models that mostly differ in layer depth, unit type, etc. We choose a simple 2-layer LSTM network which is used in the learner phase of NSpM [SMM+18] (see Figure 2.12) as a baseline. On the basis of it, we apply two kinds of attention: global attention [BCB14] and local attention [LPM15] to see how much effects the attention mechanism have in our task.

Moreover, we adopt the model proposed by Luong et al. [LPM15] and the GNMT system proposed

by Wu et al. [WSC+16], both of which achieved state-of-the-art results on natural language translation benchmarks. The former is essentially a deeper LSTM with 4 layers and local attention mechanism. The latter is described in the following.

Google's Neural Machine Translation System

Google's Neural Machine Translation (GNMT) system [WSC+16] is proposed to address several issues existing in past NMT approaches such as lack of robustness and rare words in the input sentences. The model architecture of GNMT is an LSTM network with deep encoder and decoder which have both 8-layer depth as shown in Figure 3.2. The attention mechanism is used between the bottom layer of decoder and top layer of encoder. Starting from the third layer, a residual connection is employed between the input and the output of each LSTM cell.

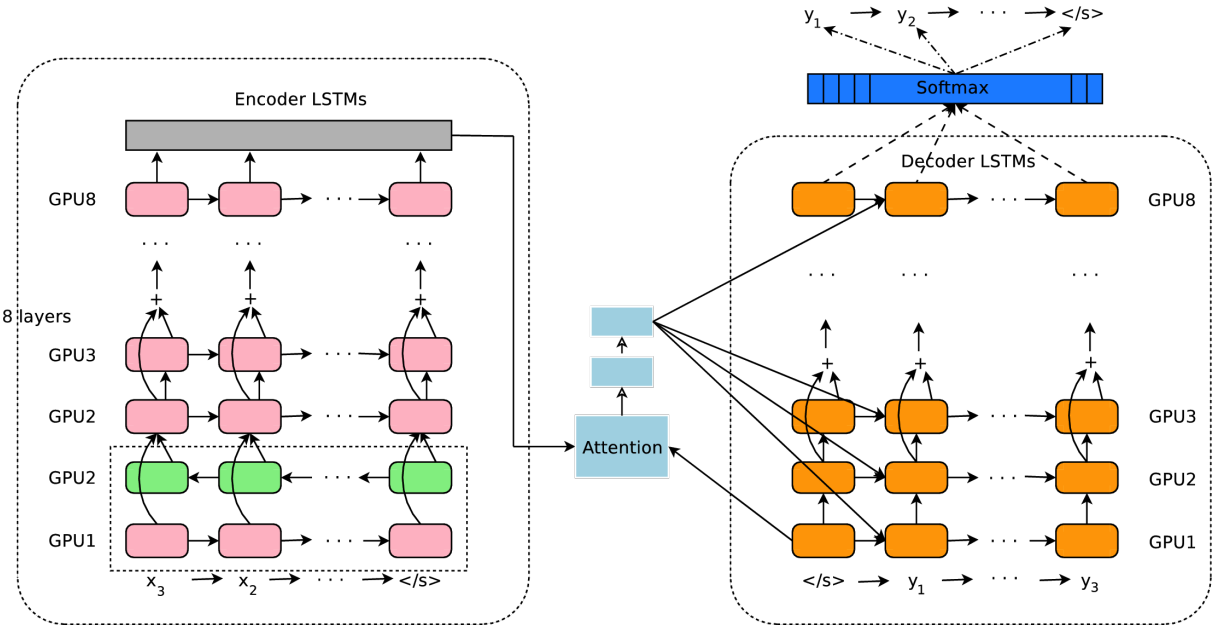


Figure 3.2: The model architecture of GNMT.

Specially, the first layer of the encoder in GNMT is a bi-directional RNN. It is based on the consideration that when output words are being translated left-to-right in the decoder, their closely related words in the source side may appear in distributed positions, which causes the distance between the correct dependency of target and source word to be longer. Although attention mechanism to a great extent has remedied this issue by allowing the decoder to refer to any position of the encoder while decoding, it is practically useful to encode the source information in both directions. The outputs of forward LSTM $\overrightarrow{x}$ and backward LSTM $\overleftarrow{x}$ are concatenated together before feeding into the next layer.

To address the issue of rare words, GNMT system adopts the wordpiece model (WPM) that segments the

words into wordpieces in the pre-processing stage and establishes a wordpiece vocabulary for the system. During decoding time, the decoder predicts wordpiece sequences which are subsequently converted back to word sequences. WPM is found to be beneficial for the translation accuracy and faster decoding speed [WSC+16]. However, due to the reason that preserving the integration of DBpedia entities in the target vocabulary is needed (mentioned in Section 3.1.2), WPM is not used in our experiments.

3.2.2 CNN-based Models

In the machine translation task, CNNs own several advantages over RNNs in the following aspects:

- Faster training speed because the computations of CNNs allow parallelization over every element in a sequence, whereas the computations in RNN are sequentially dependent on each other.
- Long-range dependencies have shorter paths when the inputs are processed in a hierarchical multi-layer CNN compared to the chain structure of RNNs. CNN is able to create representations for n continuous words in $\mathcal{O}(\frac{n}{k})$ convolutions with k-width kernels, while RNN needs $\mathcal{O}(n)$.

On the other hand, it also has certain limitations:

- The input needs to be padded before fed into the model for the reason that CNNs can only process sequences of fixed length.
- Additional position encoding is required to provide the model with a sense of ordering in the elements being dealt with.

Convolutional Sequence-to-Sequence

Convolutional Sequence to Sequence (ConvS2S) [GAG+17b] is a sequence to sequence modeling architecture that depends on CNNs instead of traditional RNNs.

Figure 3.3 depicts the general architecture of the ConvS2S model and how it can be trained. Actually, it still is the architecture of encoder-decoder, and the decoder is also aided by attention mechanism as the RNN-based models. The encoder (placed at top) and the decoder (placed at bottom left) both consist of several stacked convolutional blocks (only one block is drawn in the figure). Each convolutional block is composed of a one-dimensional convolutional layer (shown in Section 2.2.4) and a following Gated Linear Unit (GLU) as non-linearity. There is a residual connection between the input to the convolutional layer and the output of GLU.

As Figure 3.4 displays, the input of the convolutional block can be the output from the previous layer or the word embeddings, which is only the case at the bottom layer. Initially in the training phase, the input for the encoder is $\mathbf{e} = (w_1 + p_1, ..., w_m + p_m)$ where m is the length of an input sentence, $w_i \in \mathbb{R}^f$ and

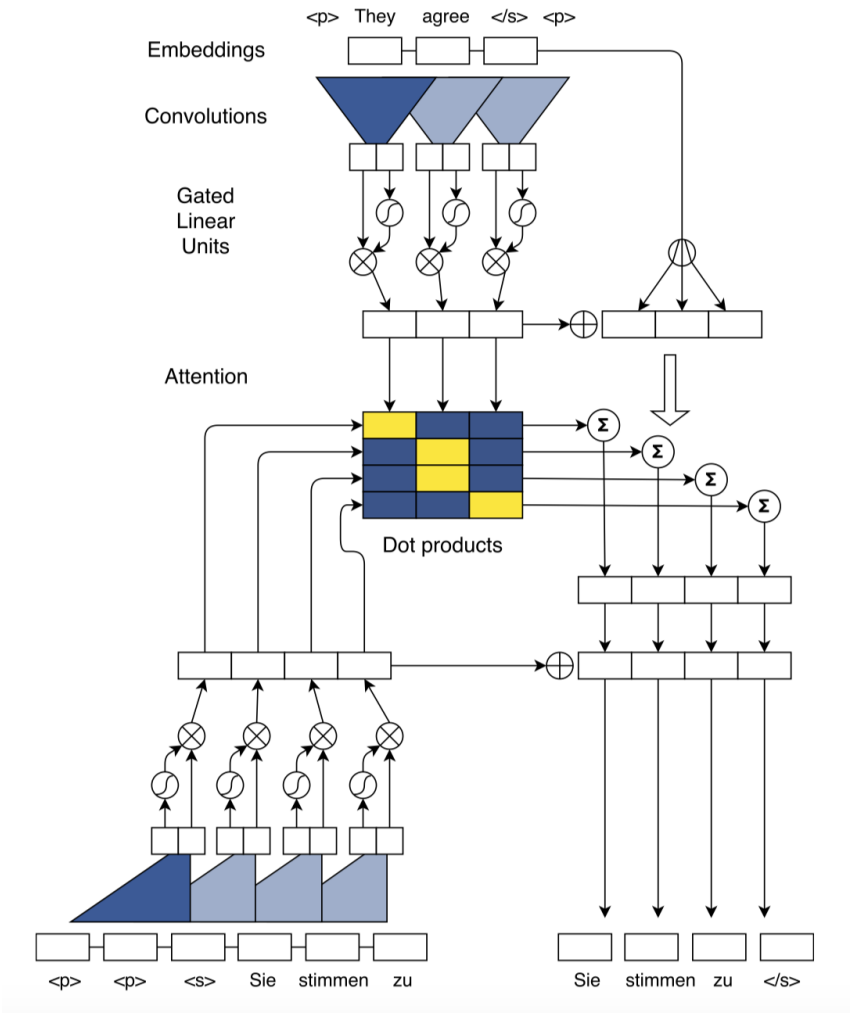


Figure 3.3: The demonstration of training the Convolutional Sequence-to-Sequence model [GAG+17b]

$p_i \in \mathbb{R}^f$ are the word embedding and the positional embedding of the i-th input element respectively (f is the embedding size). We denote the output of L' layers of the encoder convolutional block as $\mathbf{z} = (z^1, ..., z^{L'})$ where the output of l-th layer is $z^l = (z_1^l, ..., z_m^l)$ and l-th layer is stacked above the $l-1$-th layer. We denote the output of i-th convolution operation $Y = [A\, B] \in \mathbb{R}^{2d}$ where $A \in \mathbb{R}_d$ and $B \in \mathbb{R}_d$ takes each half of Y, then:

$$z_i^l = A \otimes \sigma(B) + z_i^{l-1}$$

where $\otimes$ is the point-wise multiplication and σ is the sigmoid function.

Note that because each convolutional block can only receive a fixed number of input elements where it is often a large number that covers the longest sentence in the corpora (i.e. m is often smaller than the input dimension), the input of each block before the convolution usually needs to be padded with zero vectors (like the grey squares shown in Section 2.2.4).

Compared to the encoder, each convolutional block in the decoder has one more attention module located

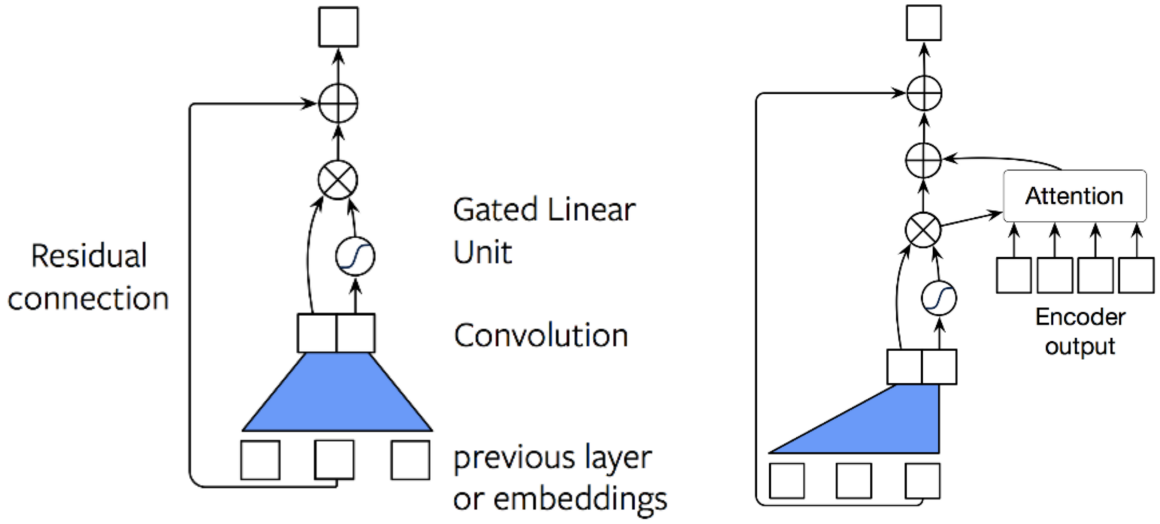


Figure 3.4: The illustration of convolutional blocks in the encoder (left side) and decoder (right side) of ConvS2S model [Aul17].

after the output of GLU and before the residual connection, which is shown on the right side of Figure 3.4. We denote the output of L layers of the decoder convolutional block as $\mathbf{h} = (h^1, ..., h^{L'})$, and the l-th layer's output as $h^l = (h_1^l, ..., h_n^l)$ where n is, during the training phase the length of target input elements, or during the generation phase the number of current decoding step. In l-th layer, before attention module, the computation is exactly like in the encoder blocks. We denote the i-th intermediate output of GLU as v_i^l, the final output of i-th decoder state h_i^l is then computed as follows:

$$d_i^l = W_d^l v_i^l + b_d^l + g_i \tag{3.1}$$

$$a_{ij}^l = \frac{\exp(d_i^l \cdot z_j^{L'})}{\sum_{t=1}^{m} \exp(d_i^l \cdot z_t^{L'})} \tag{3.2}$$

$$c_i^l = \sum_{j=1}^{m} a_{ij}^l (z_j^{L'} + e_j) \tag{3.3}$$

$$h_i^l = c_i^l + v_i^l + h_i^{l-1} \tag{3.4}$$

To compute the attention weights a_{ij}^l between the i-th decoder state and j-th source element, a decoder state summary d_i^l is firstly computed in a linear layer with v_i^l and the embedding of i-th target element g_i in Eq. 3.1. After that, the dot-product of d_i^l and j-th output of the top encoder block $z_j^{L'}$ is calculated as the attention weights in Eq. 3.2. To further let the model refer to the input elements, in Eq. 3.3, the conditional input c_i^l is computed as the weighted sum of both the encoder outputs and encoder input embeddings $\mathbf{e}$. At last, the output h_i^l is the addition of the conditional input, the output of GLU, and the input of this decoder layer from residual connection.

Since attentions exist in each decoder layer and are computed individually, higher layers have access to the information of which elements the lower layers attended to, which is called multi-step attention.

Given the output of the top decoder layer h^L, a distribution over the possible next target elements at i-th position can be retrieved with a softmax layer $softmax(W_o h_i^L + b_o)$ built upon a linear layer with weights W_o and bias b_o.

This architecture is able to parallelize the computations required during the training phase since the target elements are known beforehand and can be fed to the decoder once. However, during the inference stage where the target elements are not available, the computations in the decoder are still sequential. Nevertheless, full parallelization of the encoder is enough to make this model faster than most of its RNN rivals [GAG+17b].

3.2.3 Self-attention Model

Recently a new sequence to sequence model that is based neither on RNNs nor CNNs but only on the attention mechanism has emerged and quickly drawn plenty of attention. This kind of model shows a simpler architecture but brings less training cost and is currently claimed to achieve state-of-the-art results on several popular benchmarks such as English-German and English-French.

The Transformer

The Transformer model [VSP+17] is another novel neural machine translation model adopting the encoder decoder structure, without RNNs or CNNs as the primary units but entirely based on the attention mechanism. The traditional RNN models use attention mechanism to connect the encoder and decoder at some time steps. Differently, the Transformer model uses internal stacked self-attention in both encoder and decoder.

The attention mechanism proposed in the Transformer model is called Multi-Head Attention where each head performs Scaled Dot-Product Attention (see Figure 3.5) [VSP+17].

The scaled dot-product attention can be described as a mapping from queries Q and a set of key-value pairs K, V to an output attention matrix. Among these vectors, the queries and keys are of the same dimension d_k and the values are of dimension d_v. The output of dimension d_v is computed by:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.5}$$

where the dot products of the queries and all keys are computed firstly, then scaled by $\sqrt{d_k}$, and finally put through a softmax function to obtain weights on the different positions of values.

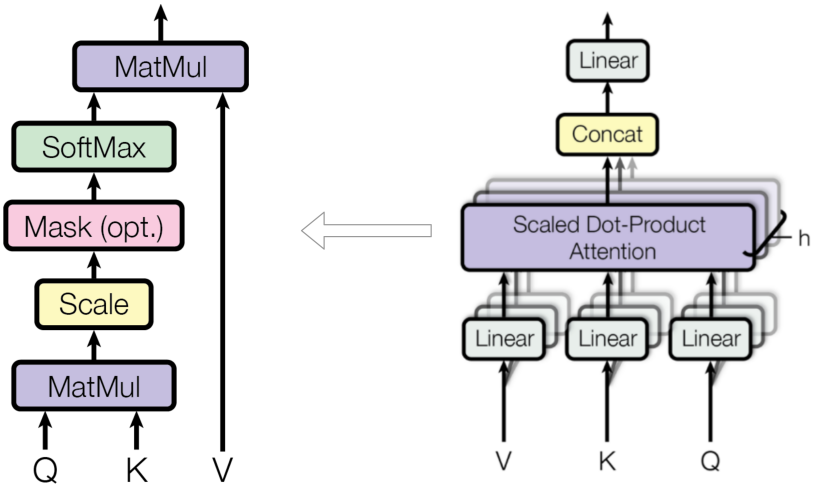


Figure 3.5: The special multi-head attention mechanism (right) in the Transformer model. Each head performs a scaled dot-product attention operation (left) [VSP+17].

Figure 3.6 displays the architecture of the Transformer model. The encoder and decoder both consist of N stacked layers where each layer of the encoder is shown on the left half and the decoder on the right half. The connection between them goes from the top layer of the encoder to each layer of the decoder. For all layers as well as their sub-layers in the encoder and decoder, the input and output have the same dimension d_{model}.

Each encoder layer consists of two sub-layers. The first is a multi-head attention sub-layer and the second is a point-wise fully connected feed forward sub-layer. Each sub-layer is employed with a residual connection, which means the input to the sub-layer is further passed to the tail and added to the output. After that, the output is normalized. In summary, the output of each sub-layer can be specified by $Norm(x + Sublayer(x))$ where x is the input and $Sublayer(x)$ indicates the operation performed by this sub-layer.

For multi-head attention sub-layers (yellow rectangles in Figure 3.6), the input x is composed of three parts $Q \in \mathbb{R}^{d_{model}}$, $K \in \mathbb{R}^{d_{model}}$, and $V \in \mathbb{R}^{d_{model}}$. In sub-layers that aim at performing self-attentions such as the ones in the encoder and the masked sub-layers in the decoder, Q, K, and V are identical and from the output of the previous layer. In sub-layers that connect both the encoder and decoder, $V = K$ and come from the encoder, Q is from the decoder and the one used in the residual connection. In each multi-head attention sub-layer, there are h heads that are equivalent to h parallel attention layers. Each head projects Q, K, and V to respectively d_k, d_k, and d_v dimensions and performs a scaled dot-product attention. The outputs gathered from all the heads are concatenated and projected again to a final output of dimension d_{model}. In an equation, it is represented as follows:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

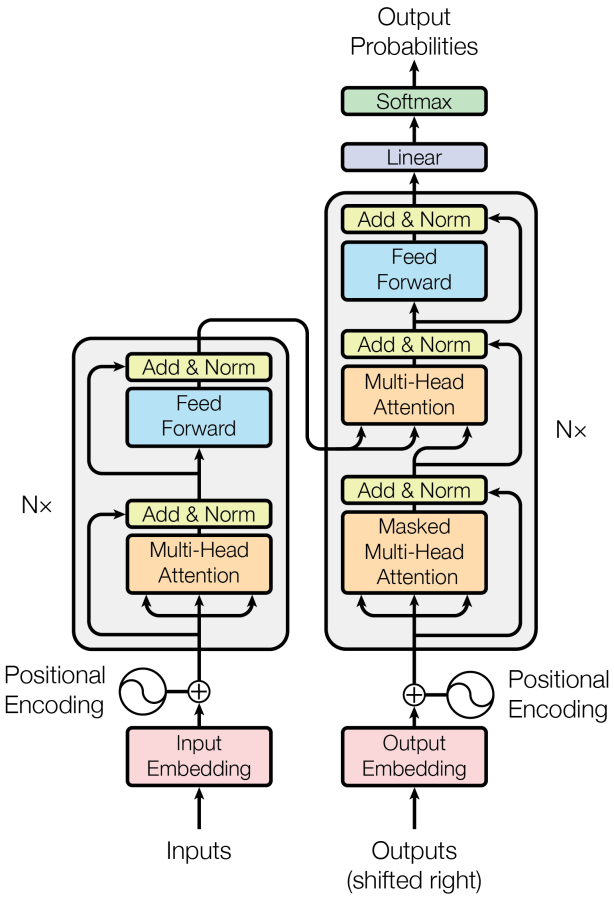


Figure 3.6: The architecture of the Transformer model [VSP+17]

The output of each head $head_i$ is defined the same as the Eq. 3.5:

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

where $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ are matrices for the corresponding projections.

In terms of fully-connected feed forward sub-layers, $Sublayer(x)$ means:

$$FeedForward(x) = max(0, xW_1 + b_1)W_2 + b_2$$

which sequentially propagates the input through one linear transformation, a ReLU activation function, and another linear transformation where the dimension of both linear transformations is d_{ff}. Therefore, the parameters here are $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$, $b_1 \in \mathbb{R}^{d_{ff}}$, $W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$, and $b_2 \in \mathbb{R}^{d_{model}}$.

3.3 Evaluation Metrics

Automatic machine translation evaluation is crucial for the development of machine translation system, since human evaluations are normally expensive, longer and subjective to some extents. In this thesis, we

use perplexity to see how well a model is being trained, and BLEU to assess how close the translations of the model are to the reference translations.

3.3.1 Perplexity

Perplexity is a common intrinsic measurement for evaluating the quality of a language model based on the cross entropy [Koe09]. A better language model is one that assigns higher probabilities to the words that actually occur. For a target probability distribution p and an estimated probability distribution q, the cross entropy is used to assess how close they are and defined as follows:

$$H(p, q) = -\sum_{x} p(x) \log q(x)$$

where x stands for the possible values in the distribution. The perplexity is then defined as the exponentiation of the cross entropy:

$$Perplexity(p, q) = 2^{H(p,q)}$$

It is possible to use e as the base instead of 2 and it depends on which one is used in the cross entropy.

Specifically for machine translation, the target distribution p is the one-hot encoding vector of the target vocabulary and q is obtained from the result of the output softmax layer. In practice, perplexity is calculated per batch or epoch where the cross entropy is averaged over all internal decoding steps beforehand. It is shown in related research [LKM15, WSC+16] that perplexity is a good measurement for MT and our results also confirm that source-conditioned perplexity strongly correlates with MT performance.

3.3.2 BLEU

BLEU is now one of the most popular automated metrics in the evaluation of neural machine translation systems. It is noted for its high correlation with human evaluation and low marginal cost for running [PRWZ02]. We choose BLEU as it is the most dominating metric for translation evaluation at present, which brings the advantage of more straightforward comparisons with other successful models and experiments.

N-gram precision is a measurement that is commonly used to reflect the adequacy and fluency of the translation sentence. In a sentence composed of multiple words, an n-gram refers to a contiguous sequence of n words within it. To compute the precision of n-grams, one just counts up the number of the n-grams which occur in any reference translation, and divide this number by the total number of n-grams in the candidate sentence. This does not check if the candidate translation contains too many duplicate words which merely exist fewer times in reference translation.

In order to prevent this issue which often occurs in NMT results, a modified version of n-gram precision is proposed in BLEU. To compute, firstly calculate the total number of n-grams in the sentence N. Then pick all the distinct n-grams and for each distinct n-gram w count the number of times it occurs in the sentence c_w and the maximum number of times it occurs in any single reference as m_w. Next, derive a clipped count for each w by $c\_clipped_w = min(c_w, m_w)$. Finally, the modified n-gram precision is computed by:

$$P = \frac{\sum_w c\_clipped_w}{N} = \frac{\sum_w min(c_w, m_w)}{N}$$

For example, two candidate translations in Table 3.2 are evaluated against the following references:

Reference 1: the cat is on the mat

Reference 2: there is a cat on the mat

When n equals 1 (unigram), the standard precision of the first candidate is 7/7 since all of the words "the" occur in the references. Whereas the modified version achieved only 2/7, because in this sentence there is only one distinct unigram "the" that has maximum reference count $m_w = 2$ which is derived from the first reference and total count $c_w = 7 = N$.

Table 3.2: The comparison between the standard n-gram precision and the modified version on two candidates. The modified n-gram precision reflects the duplicate words issue on candidate 1 while preserving the standard measurements on candidate 2.

Candidates	n-gram precision			modified n-gram precision		
	n=1	n=2	n=3	n=1	n=2	n=3
the the the the the the the	7/7	0	0	2/7	0	0
the cat sat on the mat	5/6	3/5	1/4	5/6	3/5	1/4

Although the modified n-gram precision addresses the duplicates issue, there is a problem that when the candidate translation length is too short, it is likely that the final precision is very high. Therefore, a brevity penalty is introduced as follows:

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$$

where c and r are the length of the candidate translation and the reference. Next, a set of positive weights w_n are assigned where $\sum_{n=1}^{N}$ equals 1 to take the geometric mean of the scores of the modified n-gram precision p_n with different n-gram sizes up to some N. Experimentally, $N = 4$ and uniform weights $w_n = 1/N$ performs encouraging results [PRWZ02].

Finally, the BLEU score is computed as:

$$\text{BLEU} = BP \cdot \exp(\sum_{n=1}^{N} w_n \log p_n)$$

BLEU is used to assess the accuracy of a translation corpus as well. For each translation, there might be multiple corresponding references one of which matches best with the translation in length. In this case, we replace c and r in the brevity penalty with the sum of the length of candidate translations and the sum of the best match lengths in references. p_n is replaced as well with the geometric average of the modified n-gram precisions of all the translations.

4 Experiments

This chapter includes the details of the experiments carried out in this thesis. We introduce the utilized English-SPARQL datasets in Section 4.1. The use of frameworks and experimental setups for each model and dataset are described in Section 4.2 and 4.3. The environments for training and testing the models are introduced in Section 4.4.

4.1 Datasets

To successfully train a neural machine translation model, a large-quantity bilingual parallel corpus is often needed. In natural language translation tasks, there are abundant choices where the most adopted ones are the multilingual datasets published for training and evaluating statistical machine translation models from the Workshop on Machine Translation (WMT) and the International Workshop on Spoken Language Translation (IWSLT), such as WMT14' English-German dataset. However, regarding translating natural language to SPARQL queries targeting at some specific knowledge base the choices are rather limited. In this thesis, three English-SPARQL datasets are selected: the monument dataset [SMM+18], LC-QUAD [TMDL17], and DBNQA [SMH18].

According to our research, different from a natural language translation corpus that can be established from crowdsourcing, the construction of an NL to query language (SPARQL in this thesis) dataset appropriate for NMT training has the following challenges:

- Creating correct language pairs usually requires expert knowledge of SPARQL which is not yet owned by a large number of people.
- The knowledge of corresponding knowledge bases is further required. Most of the knowledge bases such as DBpedia have vocabularies containing transformed words from automatic crawling of large quantities of articles online, for example, "located at" is represented by "dbo:location" in DBpedia. These words are not expected to be known by common users.
- The vocabularies of the online knowledge bases are likely to change along with the updates of the whole KB, which causes some pairs of the dataset invalid and hard to be tested again.

Because of the aforementioned difficulties and related issues, the most common adoption of construction

methods by the available datasets in this field is to first manually create a list of template pairs with placeholders inside and then replace the placeholders largely with extracted entities or predicates from the latest endpoint of online knowledge base.

4.1.1 Monument dataset

The monument dataset is generated and used by the Neural SPARQL Machine [SMM+18] system. It has 14,788 question-query pairs and the questions are only in English. The full vocabulary size is about 2,500 for English and 2,200 for SPARQL.

The range of entities in this dataset is restricted to the instances of a specific class dbo:Monument. The data is generated from a list of manually crafted template pairs and related assistant SPARQL queries that can be executed directly on DBpedia endpoint. For example, given a question template "Where is <A> ?" and a query template:

```
SELECT ?x
WHERE
{
  <A> dbo:location ?x .
}
```

where <A> belongs to the class dbo:Monument in DBpedia, then one can retrieve a list of entities and their corresponding English labels to replace <A> by executing the following SPARQL query on a DBpedia endpoint:

```
SELECT ?uri ?label
WHERE
{
  ?uri rdf:type <C> .
  ?uri dbo:location ?x .
  ?uri rdfs:label ?label .
  FILTER(lang(?label) = 'en') .
}
```

where the first triple imposes the class restriction and the second triple expresses the meaning of the template question. The returned values of $?uri$ and values of $?label$ are then used in pairs to replace the corresponding placeholders <A> in the given templates.

It is claimed that [SMM+18] 38 manually annotated templates have been used in generating the monument dataset. For each query template, 600 examples were generated with the aforementioned method. However, we found that out of these 38 template pairs there are some issues that may caused the whole dataset generated to be simpler than expected:

- Some template pairs have different English templates but same SPARQL query templates or very similar-structured query templates, which means the translation model may favor generating some certain kinds of SPARQL queries.
- Some English templates are partial phrases instead of full sentence (e.g. latitude of <something>).

4.1.2 LC-QUAD

Largescale Complex Question Answering Dataset (LC-QUAD) [TMDL17] is also an English-SPARQL dataset. It contains 5,000 pairs, in which about 7,000 English words and 5,000 SPARQL tokens are used. The SPARQL queries are for DBpedia.

The goal of LC-QUAD is to provide a large dataset with complex questions where the complexity of a question depends on how many triples its intended SPARQL query contains. To complete this goal, 38 unique templates as well as 5,042 entities and 615 predicates from DBpedia are involved in the generation workflow.

The generation of data in LC-QUAD is different from that in the monument dataset. Instead of allocating an executable SPARQL query for each English-SPARQL template pair to retrieve a list of entity instances, an entity seed list as well as a predicate whitelist are prepared beforehand. Next, each entity in the entity seed list is used as a seed to extract subgraphs from DBpedia through a generic SPARQL query. The triples in the subgraphs are then used to instantiate the SPARQL templates and the corresponding English templates which are called Normalized Natural Question Templates (NNQT). After that, the instances of NNQT are examined and paraphrased through peer reviews to ensure grammatical correctness. An example in LC-QUAD is shown in Table 4.1.

Table 4.1: An example question and its corresponding instantiation of the query template and NNQT in LC-QUAD generation [TMDL17].

Template	SELECT ?uri WHERE { ?x e_in_to_e_in_out e_in_out . ?x e_in_to_e ?uri . }
Query	SELECT ?uri WHERE { ?x dbp:league dbr:Turkish_Handball_Super_League . ?x dbp:mascot ?uri . }
NNQT Instance	What is the <mascot> of the <handball team> whose <league> is <Turkish Handball Super League >?
Question	What are the mascots of the teams participating in the turkish handball super league?

LC-QUAD has richer data variety than the monument dataset. However, limited size of LC-QUAD

makes it harder to be trained with deep neural network models. In addition, due to the pursuing of complexty of questions in LC-QUAD, grammar errors still exist, and some questions contain unusual punctuation (e.g. the u.n.i.t.y group) that leads to unnoticeable incorrect tokenizations during vocabulary building.

4.1.3 DBNQA

DBpedia Neural Question Answering (DBNQA) [SMH18] is the largest DBpedia-targeting dataset we have found so far. It is also based on English and SPARQL pairs and contains 894,499 instances in total. In terms of vocabulary, it has about 131,000 words for English and 244,900 tokens for SPARQL without any reduction.

DBNQA provides a remedy for the drawbacks of previous two datasets. A large number of generic templates are extracted from the concrete examples of two existing datasets LC-QUAD and QLAD-7-Train [UNH+17] by replacing the entities with placeholders. These templates can subsequently be used in the same approach as the one in the monument dataset (see Section 4.1.1) to generate a large dataset.

DBNQA has basically satisfied the data requirements of training a neural network model. However, the relatively large vocabulary of it needs to be coped with carefully otherwise the training is likely to suffer from memory shortages. Moreover, it is necessary to point out that the size of DBNQA is still incomparable to that of natural language datasets which commonly contain over millions of data.

4.2 Frameworks

There are a large number of available frameworks that implement the models described in Section 3.2 and provide integration of prompting internal training statistics as well as external evaluation scores. We chose two frameworks by taking into account the experiment requirements in this thesis, one of which is based on TensorFlow [AAB+15] and the other based on PyTorch [PGC+17].

TensorFlow Neural Machine Translation[1] (nmt) [LBZ17], as its name indicates, is a dedicated framework for neural machine translation based on TensorFlow. It provides a flexible implementation of the RNN-based NMT models. One can easily build and train variant RNN-based architectures by specifying the hyperparameters, e.g. number of encoder-decoder layers and type of attention, through designated Python program commands. This framework is used in our experiments for instantiating, training, and testing five different models including three baseline 2-layer LSTMs, a 4-layer GNMT, and an 8-layer

[1] available at `https://github.com/tensorflow/nmt`

GNMT.

Facebook AI Research Sequence-to-Sequence Toolkit[2] (fairseq) [GAG+17b] is another framework that implements various Seq2Seq models but based on PyTorch. It can also be used to perform other NLP tasks such as text summarization. Fairseq provides off-the-shelf models as well as packed hyperparameter sets for the users to configure their experiments. We used it to train and test three models including the 4-layer LSTM with attention proposed by Luong et al.[LPM15], the ConvS2S, and the Transformer.

It should be noted that there are some differences between the use of these two frameworks in terms of training in this thesis. With the nmt we use number of examples to determine the size of a mini-batch (i.e. batch size), whereas we use number of tokens with the fairseq. That means the batch size in the fairseq varies during the training according to the lengths of the examples in the mini-batch. In addition, the training and evaluation statistics are recorded based on epochs in the fairseq and steps in the nmt. The best checkpoint saving is based on the valid BLEU in the nmt and the valid loss in the fairseq because online BLEU measurements while training is only supported in the nmt.

4.3 Experimental Setup

We split each dataset in a ratio of 80%-10%-10% for training, validation, and test set. We further do two splits on the monument dataset. First is a ratio of 50%-10%-40% to evaluate the complexity of the dataset, and the second is using the splitting approach in [SMM+18] to directly compare our results with NSpM. The latter split essentially fixes 100 examples for both validation and test set and keeps the rest for the training set. In summary, we have 5 experimental datasets, namely:

- **MonumentNSpM**, **Monument50**, **Monument80**, **LC-QUAD**, and **DBNQA**

We set up 8 different models from three categories (see Section 3.2) and train them respectively on each of the aforementioned experimental datasets with a single GPU. The names of the models and their architectures as well as reported model settings and training hyperparameters are described as follows:

- RNN-based models
  1. **NSpM[3]**: an LSTM-based RNN model with 2 layers of both the encoder and decoder where the number of hidden units is 128. We use stochastic gradient descent (sgd) optimizer with a batch size of 128 and a fixed learning rate of 1 without decaying. A dropout of 0.2 is applied. Training is limited up to 50,000 steps.

  2. **NSpM+Att1**: NSpM plus a global attention mechanism module (see Section 2.2.5). Training hyperparameters are the same as NSpM.

[2]available at `https://github.com/pytorch/fairseq`

[3]from *Neural SPARQL Machines* (available at `https://github.com/AKSW/NSpM`)

3. **NSpM+Att2**: NSpM plus a local attention mechanism module (see Section 2.2.5). Training hyperparameters are the same as NSpM.
4. **LSTM_Luong**: an LSTM-based RNN model with 4 layers of both the encoder and the decoder where the number of hidden units is 1000.
5. **GNMT-4**: a GNMT model with 4 layers of both the encoder and decoder.
6. **GNMT-8**: a GNMT model with 8 layers of both the encoder and decoder. Training hyperparameters are the same as GNMT-4.

- CNN-based models
  7. **ConvS2S**: a Convolutional Sequence-to-Sequence model.
- Self-attention models
  8. **Transformer**: A small-sized Transformer model.

In summary, we have in total 40 experiments, each of which is training and testing one model on one dataset. We run each experiment one or more times until the model shows convergence on the dataset or the trend of its result remains unchanged. Unfortunately, due to the limitation of time and hardware resources, delicate fine-tuning of training hyperparameters has not been performed across different runs of each experiment. We only tuned the parameters for all of the experiments on MonumentNSpM dataset and then applied them on other experiments.

4.4 Runtime Environment

In our experiments, the primary runtime jobs include training and testing of the NMT models, where training is usually the most compute-intensive part. It is common to use GPU as the assistance to CPU to accelerate the calculation. Given that we have 40 different experiments where each experiment consumes various amount of memory according to the size of its model and dataset, we assign them to three GPUs with memory capacity from small to large running on a High Performance Computing (HPC) server. The configurations are listed in Table 4.2. The details of the assignment are displayed in Table 4.3. In terms of software, all of the training is completed using Linux operating system with Python 3.6.4, TensorFlow 1.8.0, and PyTorch 0.4.1 installed.

A local computer Macbook Pro manufactured in 2013 is also used for running dataset splitting, data preprocessing[4], testing of some models, and plotting of all the result graphs from the training statistics. The hardware is listed in Table 4.4. The software environment is macOS High Sierra 10.13.6 with Python 3.6.5, TensorFlow 1.8.0, PyTorch 0.4.1, and matplotlib 3.0.2 installed.

[4]English tokenization is done with `https://github.com/moses-smt/mosesdecoder/tree/master/scripts/tokenizer/mosestokenizer`

Table 4.2: Three hardware configurations on HPC server used in this thesis

	GPU Small	GPU Medium	GPU Large
CPU	Intel® Xeon® CPU E5-2450 @ 2.10GHz	Intel® Xeon® CPU E5-2680 @ 2.50GHz	POWER9
RAM	24 GB	16 GB	192 GB (approximately)
Cores	8	6	32
GPU	NVIDIA® Tesla® K20Xm	NVIDIA® Tesla® K80	NVIDIA® Tesla® V100-SXM2
GPU RAM	6 GB	12 GB	32 GB

Table 4.3: The hardware assignment of the experiments, where S represents GPU small, M means GPU medium, and L represents GPU large.

	MonumentNSpM	Monument50	Monument80	LC-QUAD	DBNQA
NSpM	S	S	S	S	M
NSpM+Att1	S	S	S	S	M
NSpM+Att2	S	S	S	S	M
GNMT-4	S	S	S	S	L
GNMT-8	M	M	M	M	L
LSTM_Luong	S	S	S	S	L
ConvS2S	S	S	S	S	L
Transformer	S	S	S	S	L

Table 4.4: Local Computer

CPU	Intel® Core™ i7-4960HQ @ 2.60GHz
RAM	16 GB
Cores	4
GPU	NVIDIA® GeForce® GT 750M
GPU RAM	2 GB

5 Results and Discussion

5.1 Results

In order to reflect how well our models have been trained on different datasets, we report the perplexity for each experiment along with the training steps (from nmt) or epochs (from fairseq). During the training, we store the model parameters of step or epoch which performs best[1] on the valid set in a model checkpoint file which is later used to perform decoding on the test set and report the BLEU scores.

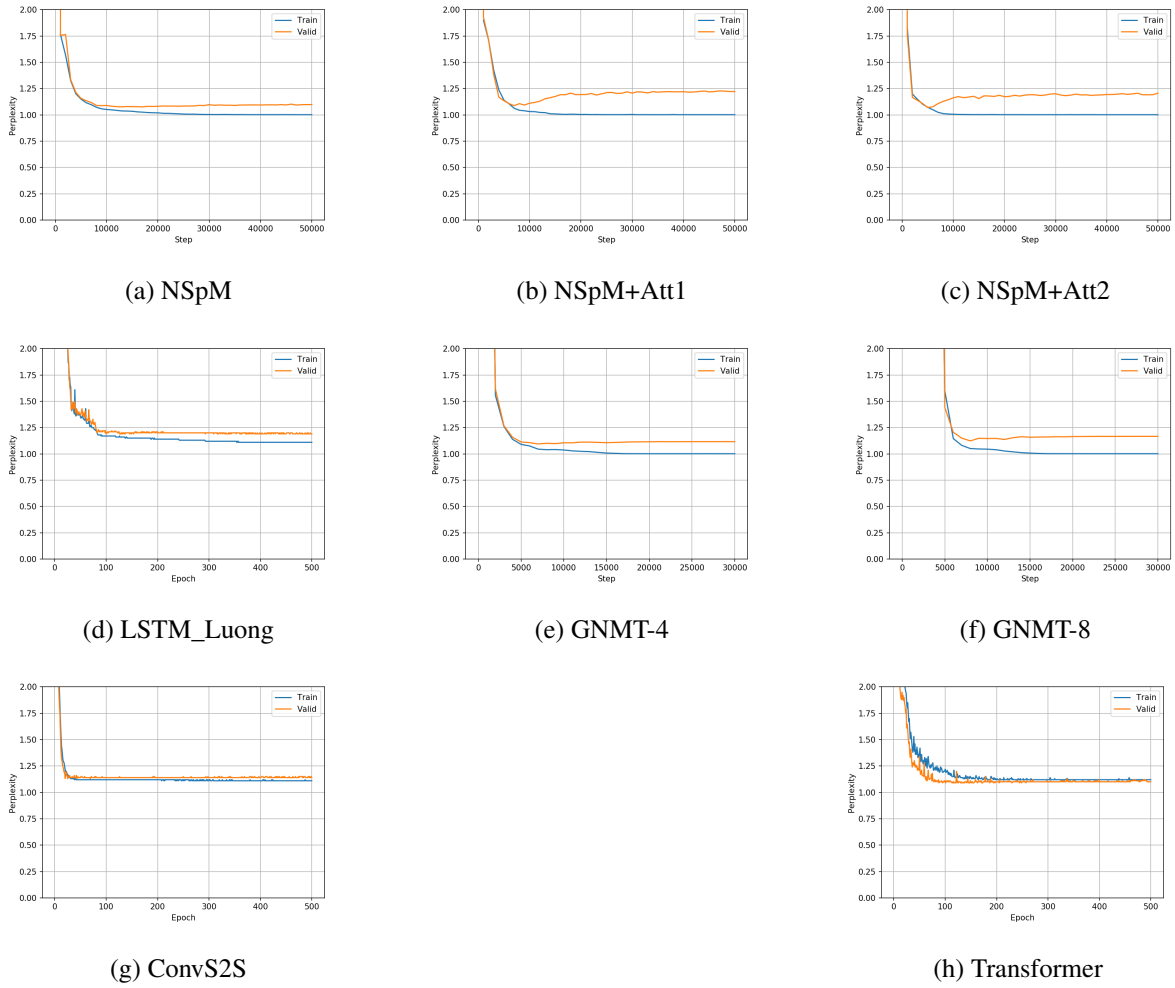
5.1.1 Training and Validation Perplexities

Figure 5.1 shows the graphs of the training and validation perplexity during the training of each model on the MonumentNSpM dataset. Compared to other two monument splits, this dataset has the largest number of training examples, where a full epoch for the fairseq is approximately equivalent to 120 steps for the nmt. After a sufficient number of iterations, all of the models are able to achieve a relatively low perplexity (close to 1) on the training set which means a nearly perfect fitting. On the valid set, all of the models converged at a perplexity between 1 and 1.25, although slight overfittings are observed on two attention-equipped NSpM models as shown in Figure 5.1b and 5.1c. In Figure 5.1h we noticed an unusual phenomenon that valid perplexity is lower than the training in early epochs.

Figure 5.2 shows the perplexity graphs on the Monument80 dataset. For this dataset, around 100 steps in the nmt are equivalent to an epoch in the fairseq. The trend of each model is similar with the MonumentNSpM dataset. Given the fact that this dataset contains more valid examples and less training examples than the MonumentNSpM dataset, the valid perplexities are mostly higher but still within the range 1 to 1.5. Slight overfittings are continued to be observed in Figure 5.2b, 5.2c, 5.2e, and 5.2f.

Next, in order to assess the complexity of the monument dataset, we experimented on the Monument50 by further reducing the number of training examples but keeping the same size of the valid set. Here, one epoch is equivalent to 60 steps. The perplexity graphs are shown in Figure 5.3. Although the training size

[1] Depending on the support of the frameworks, we store the one with best validation BLEU in nmt and the one with best validation loss in fairseq.



(a) NSpM (b) NSpM+Att1 (c) NSpM+Att2

(d) LSTM_Luong (e) GNMT-4 (f) GNMT-8

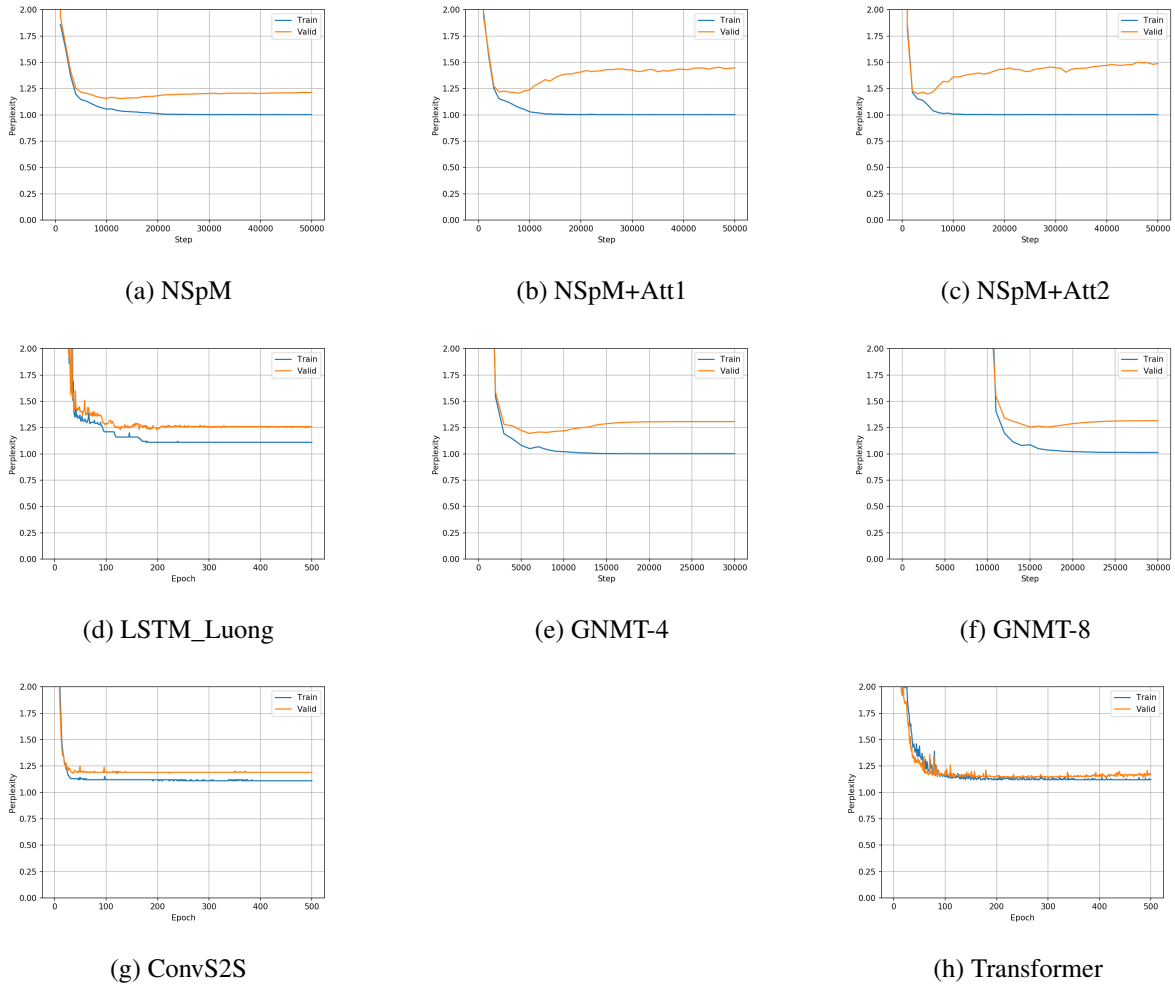
(g) ConvS2S (h) Transformer

Figure 5.1: Plots of perplexity on the training and validation set of MonumentNSpM for each model

is nearly half cut, it appears that the performance on the valid set are not much affected, especially for LSTM_Luong, ConvS2S, and Transformer which are all implemented in the fairseq. This phenomenon is also reflected on BLEU scores in Table 5.3 and Table 5.2.

In the perplexity graphs on the LC-QUAD dataset shown in Figure 5.4, we observed a noticeable difference between five models (Figure 5.4a, 5.4b, 5.4c, 5.4e and 5.4f have shown serious overfitting at the time of convergence) implemented by the nmt and three models (Figure 5.4d, 5.4g and 5.4h) implemented by the fairseq. LC-QUAD is much smaller but more complicated compared to other datasets. It only takes 34 steps to finish training an epoch inside the nmt framework. Most of the models have difficulty of providing low perplexity on the valid set, among which GNMT-8 performed the worst and ConvS2S the best.

Finally, Figure 5.5 demonstrates the perplexities on the DBNQA dataset. Across all of the models, no evident overfitting is spotted. In a batch size of 128, an epoch of DBNQA takes nearly 5600 training



(a) NSpM (b) NSpM+Att1 (c) NSpM+Att2

(d) LSTM_Luong (e) GNMT-4 (f) GNMT-8

(g) ConvS2S (h) Transformer

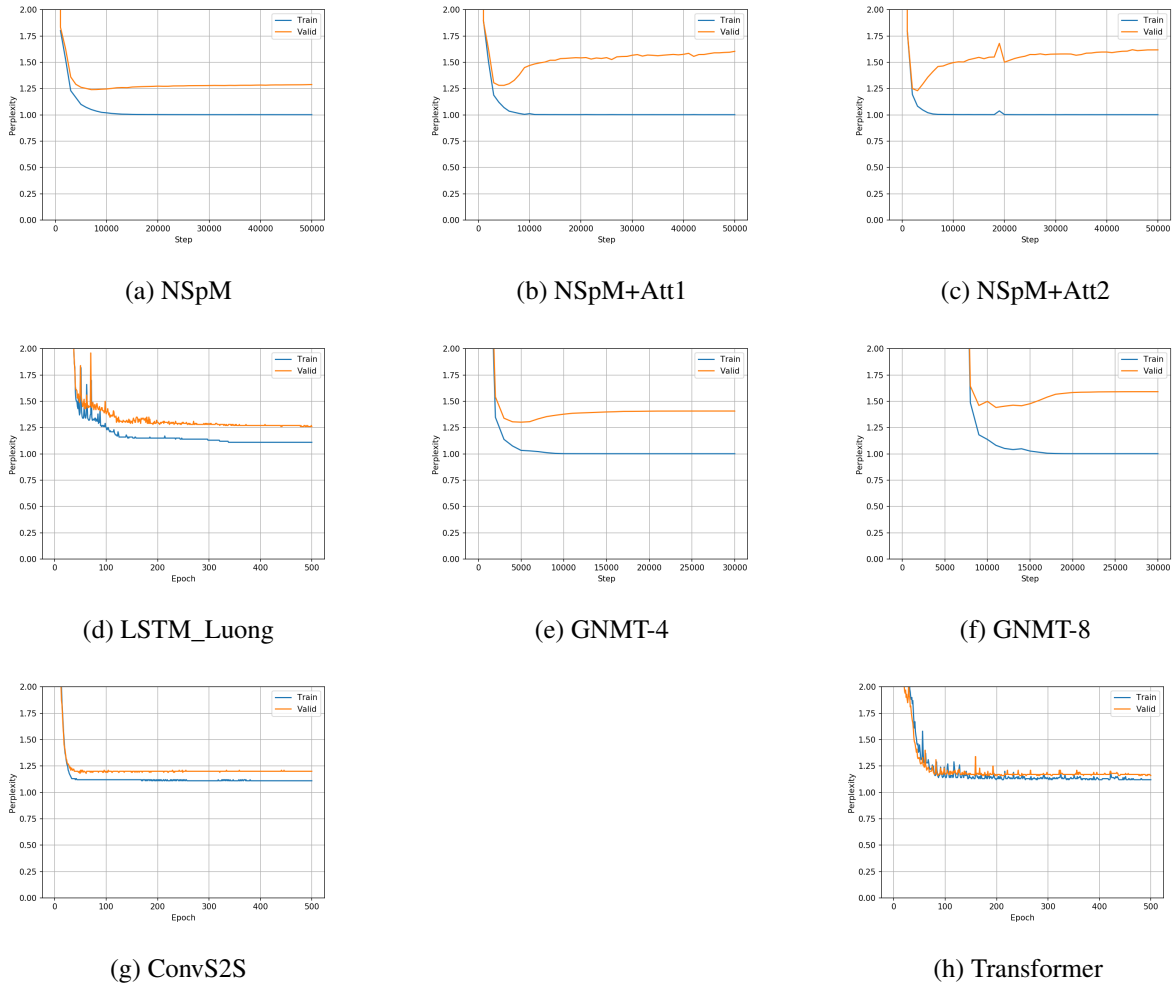
Figure 5.2: Plots of perplexity on the training and validation set of Monument80 for each model

steps. Due to the large size of DBNQA, some models like NSpM and GNMT-8 have shown rather slow or incomplete convergence since the maximum training steps for them (50k and 30k) are just equivalent to a small number (10 and 6) of epochs. Despite that, all of the models have reached a perplexity of at least 2 on the valid set, where NSpM+Att1, NSpM+Att2, and ConvS2S have achieved lower than 2 as shown in Figure 5.5b, 5.5c, and 5.5g.

5.1.2 BLEU Scores

The BLEU scores of the best checkpoint from each model are stated in Table 5.1-5.5. In each table, we primarily listed the BLEU scores on the valid and test set as well as the corresponding index of step or epoch of the selected model checkpoint. In order to refer to the state of the checkpoint at that step or epoch, we also reported the perplexities measured on the training and validation set.

In Table 5.1-5.3, we reported the BLEU scores on three splits of the monument dataset. In all of these



(a) NSpM (b) NSpM+Att1 (c) NSpM+Att2

(d) LSTM_Luong (e) GNMT-4 (f) GNMT-8

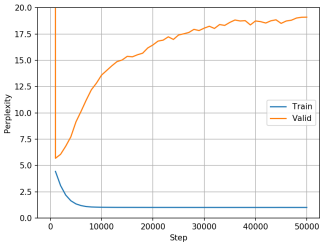
(g) ConvS2S (h) Transformer

Figure 5.3: Plots of perplexity on the training and validation set of Monument50 for each model

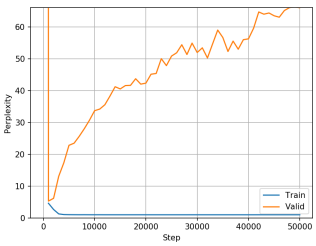
experiments, ConvS2S performed the best. It is also the case that LSTM_Luong, ConvS2S, and Transformer outperformed other models by a large margin (about 10-15 BLEUs), most evident on the MonumentNSpM experiments.

Table 5.4 shows the performance of each model on the LC-QUAD dataset. The models all achieved relatively lower BLEU scores on this dataset as well as higher perplexities compared to other datasets. Another thing to notice is that the best checkpoints selected here are all from the middle stages of the training, which correlates with the potential overfittings exhibited in Figure 5.4.

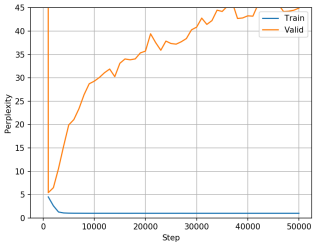
Table 5.5 displays the BLEU results on the DBNQA dataset. The ConvS2S model outperformed the NSpM model by around 30 BLEU points here, which is the largest model performance difference across all the datasets. It is also worth mentioning that the scores of the Transformer model declined to the third worst and the results of the NSpM with attentions raised to the second best in this dataset, which is usually the opposite in other datasets.



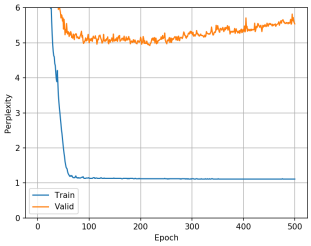
(a) NSpM



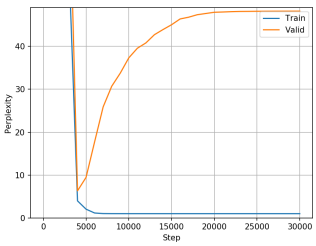
(b) NSpM+Att1



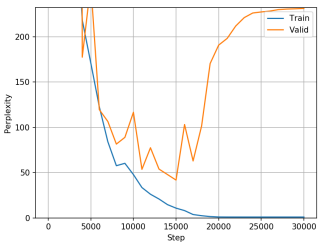
(c) NSpM+Att2



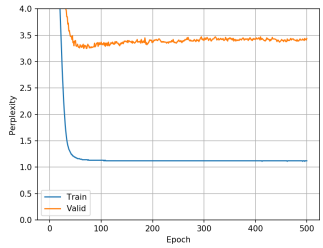
(d) LSTM_Luong



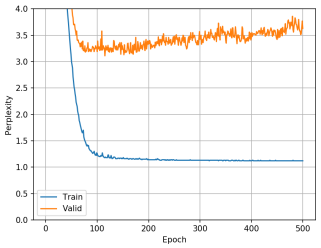
(e) GNMT-4



(f) GNMT-8



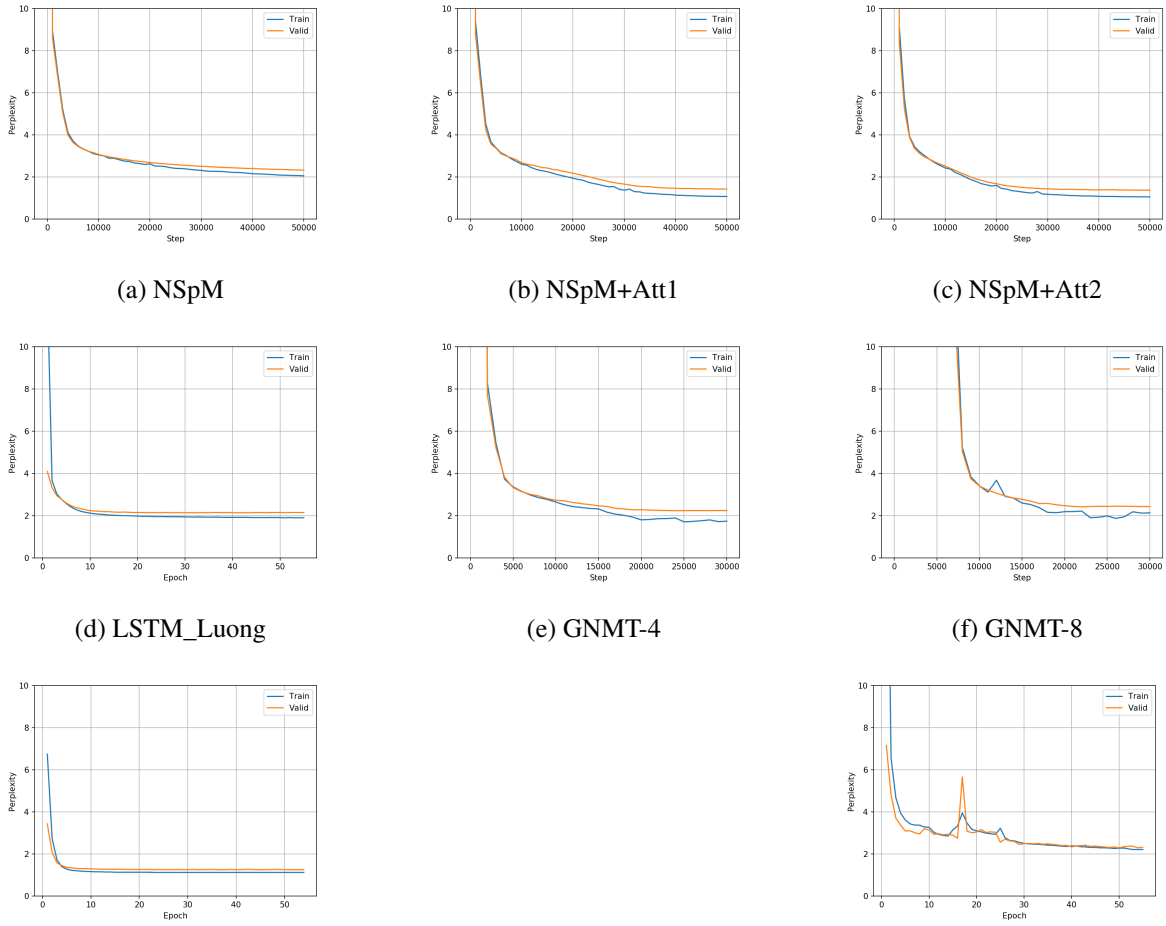
(g) ConvS2S



(h) Transformer

Figure 5.4: Plots of perplexity on the training and validation set of LC-QUAD for each model

5.2 Discussion



(a) NSpM (b) NSpM+Att1 (c) NSpM+Att2

(d) LSTM_Luong (e) GNMT-4 (f) GNMT-8

(g) ConvS2S (h) Transformer

Figure 5.5: Plots of perplexity on the training and validation set of DBNQA for each model

Table 5.1: BLEU scores and other information of the best checkpoint from each model on MonumentNSpM dataset

Models	Train ppl	Valid ppl	**Valid BLEU**	**Test BLEU**	Step / Epoch
NSpM	1.00	1.09	80.43	80.28	Step 29k
NSpM+Att1	1.01	1.16	80.36	80.58	Step 14k
NSpM+Att2	1.01	1.14	80.88	80.03	Step 9k
GNMT-4	1.03	1.11	80.12	79.53	Step 11k
GNMT-8	1.04	1.15	79.30	79.07	Step 10k
LSTM_Luong	1.11	1.19	92.39	91.67	Epoch 500
ConvS2S	1.11	1.14	**98.35**	**97.12**	Epoch 500
Transformer	1.13	1.09	95.25	95.31	Epoch 138

Table 5.2: BLEU scores and other information of the best checkpoint from each model on Monument80 dataset

Models	Train ppl	Valid ppl	**Valid BLEU**	**Test BLEU**	Step / Epoch
NSpM	1.00	1.21	87.55	87.03	Step 44k
NSpM+Att1	1.00	1.44	87.82	87.34	Step 44k
NSpM+Att2	1.00	1.44	87.99	87.37	Step 34k
GNMT-4	1.00	1.31	85.94	85.39	Step 30k
GNMT-8	1.01	1.32	84.94	84.14	Step 30k
LSTM_Luong	1.11	1.24	96.35	96.12	Epoch 200
ConvS2S	1.11	1.19	**96.74**	**96.47**	Epoch 500
Transformer	1.12	1.14	95.16	94.87	Epoch 267

Table 5.3: BLEU scores and other information of the best checkpoint from each model on Monument50 dataset

Models	Train ppl	Valid ppl	**Valid BLEU**	**Test BLEU**	Step / Epoch
NSpM	1.00	1.29	85.19	85.54	Step 50k
NSpM+Att1	1.00	1.60	85.98	86.17	Step 50k
NSpM+Att2	1.00	1.62	86.60	86.52	Step 50k
GNMT-4	1.00	1.41	82.92	83.01	Step 30k
GNMT-8	1.00	1.59	80.35	80.76	Step 30k
LSTM_Luong	1.11	1.26	94.05	94.75	Epoch 500
ConvS2S	1.11	1.20	**96.44**	**96.62**	Epoch 500
Transformer	1.14	1.17	93.80	93.92	Epoch 108

Table 5.4: BLEU scores and other information of the best checkpoint from each model on LC-QUAD dataset

Models	Train ppl	Valid ppl	**Valid BLEU**	**Test BLEU**	Step / Epoch
NSpM	1.00	16.46	43.91	43.50	Step 20k
NSpM+Att1	1.00	56.23	52.68	50.13	Step 40k
NSpM+Att2	1.00	43.20	53.03	50.86	Step 41k
GNMT-4	1.00	33.76	43.69	42.71	Step 9k
GNMT-8	1.01	229.96	44.32	43.91	Step 27k
LSTM_Luong	1.12	4.92	52.43	51.06	Epoch 218
ConvS2S	1.14	3.25	**61.89**	**59.54**	Epoch 71
Transformer	1.16	3.15	58.99	57.43	Epoch 163

Table 5.5: BLEU scores and other information of the best checkpoint from each model on DBNQA dataset

Models	Train ppl	Valid ppl	**Valid BLEU**	**Test BLEU**	Step / Epoch
NSpM	2.05	2.32	65.89	65.92	Step 50k
NSpM+Att1	1.07	1.42	89.87	89.87	Step 50k
NSpM+Att2	1.05	1.37	91.51	91.50	Step 50k
GNMT-4	1.74	2.24	69.65	69.61	Step 30k
GNMT-8	2.13	2.43	68.43	68.41	Step 30k
LSTM_Luong	1.90	2.15	77.64	77.67	Epoch 55
ConvS2S	1.12	1.25	**96.05**	**96.07**	Epoch 54
Transformer	2.21	3.34	68.68	68.82	Epoch 53

6 Conclusion

6.1 Summary

6.2 Outlook

Bibliography

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[ABK+07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A nucleus for a Web of open data. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4825 LNCS, pages 722–735. Springer, Berlin, Heidelberg, nov 2007.

[AlA15] Iyad AlAgha. Using linguistic analysis to translate arabic natural language queries to sparql. *arXiv preprint arXiv:1508.01447*, 2015.

[Aul17] Michael Auli. Sequence to sequence learning: Cnns, training and uncertainty, 2017. [Online; accessed 29. Nov. 2018].

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. sep 2014.

[BGLL17] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. Massive Exploration of Neural Machine Translation Architectures. 2017.

[BH64] Yehoshua Bar-Hillel. *Language and information: Selected essays on their theory and application*. Addison-Wesley Reading, 1964.

[BHBL09] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web, may 2001.

[CCL+13] Elena Cabrio, Philipp Cimiano, Vanessa Lopez, Axel-Cyrille Ngonga Ngomo, Christina Unger, and Sebastian Walter. QALD-3: Multilingual Question Answering over Linked Data. Technical report, 2013.

[CJF15] Marta R Costa-Jussa and José AR Fonollosa. Latest trends in hybrid machine translation and its applications. *Computer Speech & Language*, 32(1):3–10, 2015.

[CvMG+14] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. jun 2014.

[CWL14] Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax, 2014.

[CXY+17] Ruichu Cai, Boyan Xu, Xiaoyan Yang, Zhenjie Zhang, Zijian Li, and Zhihao Liang. An Encoder-Decoder Framework Translating Natural Language to Database Queries. 2017.

[dbp18] About DBpedia. `https://wiki.dbpedia.org/about`, Nov 2018. [Online; accessed 4. Nov. 2018].

[DDS+16] Mohnish Dubey, Sourish Dasgupta, Ankit Sharma, Konrad Höffner, and Jens Lehmann. AskNow: A framework for natural language query formalization in SPARQL. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016.

[DJHM13] Joachim Daiber, Max Jakob, Chris Hokamp, and Pablo N. Mendes. Improving efficiency and accuracy in multilingual entity extraction. In *Proceedings of the 9th International Conference on Semantic Systems (I-Semantics)*, 2013.

[DL16] Li Dong and Mirella Lapata. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*, 2016.

[DYDA12] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, 20(1):30–42, 2012.

[Fer13] S. Ferré. squall2sparql: a Translator from Controlled English to Full SPARQL 1.1. In Elena Cabrio, Philipp Cimiano, Vanessa Lopez, Axel-Cyrille Ngonga Ngomo, Christina Unger, and Sebastian Walter, editors, *Work. Multilingual Question Answering over Linked Data (QALD-3)*, Valencia, Spain, September 2013. See Online Working Notes at

http://www.clef2013.org/.

[GAG+17a] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional Sequence to Sequence Learning. 2017.

[GAG+17b] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional Sequence to Sequence Learning. In *Proc. of ICML*, 2017.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GS14] Fabien Gandon and Guus Schreiber. Rdf 1.1 xml syntax. *World Wide Web Consortium (W3C)*, 2014.

[HASB13] Ivan Herman, Ben Adida, Manu Sporny, and Mark Birbeck. Rdfa 1.1 primer - second edition. *World Wide Web Consortium (W3C)*, 2013.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HS13] Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. Technical report, 2013.

[Kar15] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. *Andrej Karpathy blog*, 2015.

[KBZ06] Esther Kaufmann, Abraham Bernstein, and Renato Zumstein. Querix: A natural language interface to query ontologies based on clarification dialogs. In *In: 5th ISWC*, pages 980–981. Springer, 2006.

[Koe09] Philipp Koehn. *Statistical machine translation*. Cambridge University Press, 2009.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[LBZ17] Minh-Thang Luong, Eugene Brevdo, and Rui Zhao. Neural machine translation (seq2seq) tutorial. *https://github.com/tensorflow/nmt*, 2017.

[ldt18] Tools | Linked Data - Connect Distributed Data across the Web. `http://linkeddata.org/tools`, Nov 2018. [Online; accessed 4. Nov. 2018].

[LF18] Fabiano Ferreira Luz and Marcelo Finger. Semantic parsing natural language into SPARQL: improving target language representation with neural attention. *CoRR*, abs/1803.04329, 2018.

[lin18] Linked Data - W3C Standards. `https://www.w3.org/standards/semanticweb/data`, Jul 2018. [Online; accessed 4. Nov. 2018].

[LKM15] Thang Luong, Michael Kayser, and Christopher D Manning. Deep neural language models for machine translation. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 305–309, 2015.

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective Approaches to Attention-based Neural Machine Translation. *EMNLP*, (September):11, 2015.

[Mil98] George Miller. *WordNet: An electronic lexical database*. MIT press, 1998.

[MWN17] Diego Moussallem, Matthias Wauer, and Axel-Cyrille Ngonga Ngomo. Machine Translation Using Semantic Web Technologies: A Survey. 2017.

[Okp14] MD Okpor. Machine translation approaches: issues and challenges. *International Journal of Computer Science Issues (IJCSI)*, 11(5):159, 2014.

[PC14] Ericand Prud'hommeaux and Gavin Carothers. Rdf 1.1 turtle: Terse rdf triple language. *World Wide Web Consortium (W3C)*, 2014.

[PGC+17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[PHH13] Camille Pradel, Ollivier Haemmerlé, and Nathalie Hernandez. Natural language query interpretation into SPARQL using patterns. In *CEUR Workshop Proceedings*, volume 1034. CEUR-WS, 2013.

[Pop12] Maja Popović. Class error rates for evaluation of machine translation output. *Proceedings of the Seventh Workshop on Statistical Machine Translation*, pages 71–75, 2012.

[PRWZ02] Kishore Papineni, Salim Roukos, Todd Ward, and Wj Zhu. BLEU: a method for automatic evaluation of machine translation. ... *of the 40Th Annual Meeting on* ..., (July):311–318, 2002.

[SHBL06] Nigel Shadbolt, Wendy Hall, and Tim Berners-Lee. The semantic web revisited, 2006.

[SKL14] Manu Sporny, Gregg Kellogg, and Markus Lanthaler. Json-ld 1.0. *World Wide Web Consortium (W3C)*, 2014.

[SMH18] Tommaso Soru, Edgard Marx, and Ann-Kathrin Hartmann. Generating a Large Dataset for Neural Question Answering over the DBpedia Knowledge Base. Technical report, 2018.

[SMM+18] Tommaso Soru, Edgard Marx, Diego Moussallem, Gustavo Publio, André Valdestilhas, Diego Esteves, and Ciro Baron Neto. SPARQL as a foreign language. In *CEUR Workshop Proceedings*, volume 2044, aug 2018.

[SMV+18] Tommaso Soru, Edgard Marx, André Valdestilhas, Diego Esteves, Diego Moussallem, and Gustavo Publio. Neural Machine Translation for Query Construction and Composition. 2018.

[SP97] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[SR14] Guus Schreiber and Yves Raimond. Rdf 1.1 primer. w3c working group note. *World Wide Web Consortium (W3C)*, 2014.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems (NIPS)*, pages 3104–3112, 2014.

[TMDL17] Priyansh Trivedi, Gaurav Maheshwari, Mohnish Dubey, and Jens Lehmann. Lc-quad: A corpus for complex question answering over knowledge graphs. In *Proceedings of the 16th International Semantic Web Conference (ISWC)*, pages 210–218. Springer, 2017.

[UFL+14] Christina Unger, Corina Forascu, Vanessa Lopez, Axel-Cyrille Ngonga Ngomo, Elena Cabrio, Philipp Cimiano, and Sebastian Walter. Question Answering over Linked Data (QALD-4). In Linda Cappellato, Nicola Ferro, Martin Halvey, and Wessel Kraaij, editors, *Working Notes for CLEF 2014 Conference*, Sheffield, United Kingdom, September 2014.

[UFL+15] Christina Unger, Corina Forascu, Vanessa Lopez, Axel-Cyrille Ngonga Ngomo, Elena Cabrio, Philipp Cimiano, and Sebastian Walter. Question Answering over Linked Data (QALD-5). *Working Notes of CLEF 2015 - Conference and Labs of the Evaluation forum*, 1391, 2015.

[UNH+17] Ricardo Usbeck, Axel-Cyrille Ngonga Ngomo, Bastian Haarmann, Anastasia Krithara, Michael Röder, and Giulio Napolitano. 7th open challenge on question answering over linked data (qald-7). In *Semantic Web Evaluation Challenge*, pages 59–69. Springer, 2017.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. jun 2017.

[WSC+16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva

Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *ArXiv e-prints*, pages 1–23, sep 2016.

[WXZ+17] Lijun Wu, Yingce Xia, Li Zhao, Fei Tian, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. Adversarial neural machine translation. *arXiv preprint arXiv:1704.06933*, 2017.

[XFZ14] Kun Xu, Yansong Feng, and Dongyan Zhao. Xser@QALD-4: Answering natural language questions via phrasal semantic parsing. In *CEUR Workshop Proceedings*, volume 1180, pages 1260–1274. Springer, Berlin, Heidelberg, 2014.

[YCWX17] Zhen Yang, Wei Chen, Feng Wang, and Bo Xu. Improving neural machine translation with conditional sequence generative adversarial nets. *arXiv preprint arXiv:1703.04887*, 2017.

[ZXS17] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017.

List of Figures

List of Tables