Neural Machine Translating from Natural Language to SPARQL

Xiaoyu Yin[a], Dagmar Gromann[b,*], Sebastian Rudolph[a]

[a]*TU Dresden, Nöthnitzer Str. 46, 01187 Dresden, Germany*
[b]*University of Vienna, Gymnasiumstr. 50, 1190 Vienna, Austria*

Abstract

SPARQL is a highly powerful query language for an ever-growing number of resources and knowledge graphs represented in the Resource Description Framework (RDF) data format. Using it requires a certain familiarity with the entities in the domain to be queried as well as expertise in the language's syntax and semantics, none of which average human web users can be assumed to possess. To overcome this limitation, automatically translating natural language questions to SPARQL queries has been a vibrant field of research. However, to this date, the vast success of deep learning methods has not yet been fully propagated to this research problem. This paper contributes to filling this gap by evaluating the utilization of eight different Neural Machine Translation (NMT) models for the task of translating from natural language to the structured query language SPARQL. While highlighting the importance of high-quantity and high-quality datasets, the results show a dominance of a Convolutional Neural Network (CNN)-based architecture with a Bilingual Evaluation Understudy (BLEU) score of up to 98 and accuracy of up to 94%.

Keywords: SPARQL, Neural Machine Translation, natural language queries, learning structured knowledge

*Corresponding author
Email address: `dagmar.gromann@univie.ac.at` (Dagmar Gromann)
URL: `https://transvienna.univie.ac.at/` (Dagmar Gromann)

1. Introduction

SPARQL [1] is the standard query language for retrieving and manipulating information contained in RDF [2] graphs. In contrast to regular search engines, where natural language (NL) queries can be posed, using SPARQL requires knowledge about the entities in the domain to be queried as well as an understanding of the syntax and semantics of the language. For this reason, its use is generally limited to a group of Semantic Web experts proficient in the query language. To spread its accessibility to a wider audience and thereby making knowledge stored in formats queried with SPARQL available, ways to automatically translate from NL questions to SPARQL queries have been investigated over the last decades. This paper contributes to this line of work by evaluating the use of NMT to automatically translate NL to SPARQL.[1]

Approaches to produce SPARQL queries from a controlled NL [3] or using an intermediary format (e.g. [4]) have obtained good results on highly complex SPARQL queries. Many such approaches exist, however, this paper focuses on comparing the utilization of NMT models to address this task and we do not consider approaches that do not use neural networks. In view of the success of NMT approaches, it comes as a surprise that very few of them have been utilized to address this particular problem [5, 6, 7]. Luz et al. [8] combine an attention-equipped Long Short-Term Memory (LSTM) encoder-decoder model with a probabilistic language model and obtain good results on the comparatively small Geo880 dataset. Soru et al. [9, 10] propose a dataset and two-layer LSTM model for this task, both of which are used as baselines in this paper.

To advance the use of NMT models in this area, this paper presents a large-scale comparison of three distinct neural network architectures (Recurrent Neural Networks (RNNs), CNNs, and Transformer) with a total of eight models across three datasets (Monument [9], Largescale Complex Question Answering Dataset (LC-QUAD) [11], and, as the largest of the three, DBpedia Neural Question Answering (DBNQA) [12]). The objective of such comparison is to identify the best contender for this specific task

[1]Full details of the experiments and its code are available at `https://github.com/xiaoyuin/tntspa`

among state-of-the-art NMT architectures. Results are evaluated with a common NMT metric called BLEU, string-matching accuracy, and model perplexity. All datasets, model checkpoints, and evaluation graphs (perplexity and BLEU score) are publicly available.[1]

In terms of structure, we will first provide some preliminary details on SPARQL in Section 2.1, NMT models in Section 2.2 and evaluation metrics in Section 2.3. Section 3 discusses related work and Section 4 details the encoding of SPARQL queries for usage in Sequence to Sequence (S2S) models and the models that are used for this comparison as well as the utilized evaluation metrics. Section 5 describes the datasets and experimental setup, before details on the results are provided in Section 6. Prior to some concluding remarks in Section 8, we discuss comparisons of datasets, models, and the limitations of this approach in Section 7.

2. Preliminaries

This section provides background information central to our approach. First, the query language SPARQL is briefly introduced. Second, a general overview of NMT and its recent developments will be presented. Finally, common evaluation metrics to automatically assess the performance of machine translation models are described.

2.1. SPARQL

SPARQL [1] is a structured, syntactically and semantically defined language to query graphs in the RDF [2] data format. A SPARQL query generally consists of a set of triple patterns that might contain variables. A query consists of two parts: a `SELECT` operator that identifies the variables to appear in the query and a `WHERE` clause that provides the graph pattern to which a subgraph of the RDF data store is to be matched. Extensions with additional operators help to filter, limit, or group the results. Additionally, the query language supports negation, aggregation, counting and several other features. Each resource iNLs represented by an Internationalized Resource Identifier (IRI), a generalization of a URI, which may be abbreviated by prefixed names. An example of a SPARQL query is provided in Listing 1, where the first two lines exemplify a prefix declaration, which is required in order to resolve the IRIs in the query. All strings preceded by a question mark denote variables. The triple pattern in the `WHERE` clause of Listing 1 returns a solution set, which in our example only contains `http://dbpedia.org/resource/Chess` as this is what both people in the query are known for and the only value matching the corresponding RDF graph.

2.2. Neural Machine Translation

NMT approaches are designed to automatically translate from one NL to another. This section specifies some of the important developments in the field to justify our choice of architectures. One popular choice in NMT and S2S problems in general is the encoder-decoder architecture. The popularity of this setting is based on the fact that the input and output sequence can be of variable length. It has also been shown to be superior to traditional phrase-based models in ease of extracting features, flexibility with regard to the configuration of models, and better accuracy [13].

Encoder and decoder can be implemented by means of any (and even different) neural network architectures, since they are solely connected by sharing a compressed representation of the input sequence. Sutskever et al. [14] implement encoder and decoder as a multilayer LSTM model and also utilize Bidirectional LSTMs (BiLSTMs), which connect two hidden layers of opposite directions from past (backwards) and future (forward) states simultaneously, that is, reading the input sequence from left to right and right to left. However, their architecture experiences a drop in performance with an increase in input sequence length. To address this issue, attention mechanisms have been proposed. Instead of condensing the entire input sequence into one context vector, attention takes the last hidden layer of the encoder and is trained to assign higher weights to those elements of the input sequence that are more important for a given time step than others, which are then combined to one context vector. This approach is usually referred to as global attention [15] and comes with an increased training cost. For this reason, local attention has been proposed [16], which only considers a subset of the whole input sequence in each time step. Since RNN-based architectures are time- and resource-intensive, alternative models based on CNNs have been proposed [17], which can parallelize the process. One ground-breaking success in NMT was achieved with a Transformer model [18] that as further simplification of the neural architecture omits recurrence and convolution and alternates fully connected layers and attention mechanisms.

2.3. Evaluation metrics

While there are several metrics to assess the performance of machine translation models, in this approach the BLEU score is utilized and thus introduced in the following. Additionally, it is common to assess the prediction ability of a neural network model with perplexity, which will be introduced and its specific use for machine translation models will be explained.

2.3.1. BLEU score

Human evaluations are usually time- and cost-intensive and to some degree subjective. As a more efficient alternative to evaluate machine translation outputs, Papineni et al. [19] suggested the BLEU score that compares tokens in the reference translation(s) with tokens in the candidate translation generated by the NMT model. It counts the number of times an n-gram occurs in the reference translation(s), takes the maximum count of each n-gram, and then clips the count of the n-grams in the candidate translation to the maximum count in the reference. This helps avoiding distorted precision counts based on duplicates. However, there is a problem when the length c of the candidate translation is too short, in which case the final precision is likely to be too high. To counteract this, a brevity penalty (BP) is introduced which is set to 1 if c is larger than the maximal reference length r and set to $\exp(1 - r/c)$ otherwise. Next, a set $\{w_1, \ldots, w_N\}$ of positive weights with $\sum_{n=1}^{N} w_n = 1$ is chosen to take the weighted geometric mean of the scores of the modified n-gram precision p_n with different n-gram sizes up to some maximum length N. The final BLEU score is computed as shown in Equation 1.

$$\mathit{BLEU} = \mathit{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{1}$$

Experimentally, N=4 and uniform weights w_n=$\frac{1}{N}$ have shown good results [19].

2.3.2. *Perplexity*

Recall that for a target probability distribution p and an estimated probability distribution q, the cross entropy $H(p,q)$ measuring their similarity is defined by Equation 2, where x stands for the possible values in the distribution.

$$H(p,q) = -\sum_{x} p(x)\log q(x) \tag{2}$$

The perplexity is then defined as the exponentiation of the cross entropy as shown in Equation 3.

$$\textit{Perplexity}(p,q) = 2^{H(p,q)} \tag{3}$$

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 has been shown in related research [13, 20] that perplexity is a good measure for MT and our results also confirm that source-conditioned perplexity strongly correlates with MT performance.

3. Related work

The primary focus of our investigation has been on neural network models that can be used to map NL statements to SPARQL expressions. Thus, we focus our related work on neural network models but broaden the range to models that translate to other structured query languages. Approaches that do not use neural network architectures are not considered here.

Dong et al. [6] presented a method based on an encoder-decoder model with attention mechanism aimed at translating an NL sequence to a logical form representation of its meaning with minimum domain knowledge. To address the problem that general S2S models ignore hierarchical structures, they proposed a Sequence to Tree (S2T) model that has a special decoder better able to capture the hierarchical structure of logical forms. The two models are tested on four different datasets and evaluated with accuracy, where the S2T model achieves a score of up to 90.0 on a job listing dataset.

Given the difference in training objective and nature of the task as well as recent advances in S2S models, we did not consider this model as a baseline.

Cai et al. [5] proposed an enhanced encoder-decoder framework building on LSTMs for the task of translating NL to the Structured Query Language (SQL), a query language similar to SPARQL but targeting relational databases instead of graph databases. On the encoder side, input is augmented with language-aware semantic labels, such as table names or column names. On the decoder side, grammatical states, that is, states of the translation output in Backus-Naur Form, are added as hidden states in the memory layer. They used not only BLEU [19], but also query accuracy, tuple recall, and tuple precision for measuring the quality of output queries, and achieved a BLEU score of 85.2 on the Geo880 dataset, outperforming previous approaches. Providing NMT models with additional information as done in this approach and the approach described next might be interesting future work, while the focus in this paper is a comparison of state-of-the-art neural architectures to single out the best contender for such augmentation and model fine-tuning.

Zhong et al. [7] proposed a framework called *Seq2SQL* that utilized an LSTM-based encoder-decoder architecture to translate NL questions to SQL. Similar to Cai et al. [5], input NL questions were augmented by adding the column names of the queried table. Correspondingly, the decoder was split into three components, predicting aggregation classifier, column names, and `WHERE` clause part of a SQL query, respectively. Instead of conventional teacher forcing, the model was trained with reinforcement learning to avoid that queries delivering correct results upon execution but not having exact string matches would be wrongly penalized. To address this issue in the evaluation, execution accuracy and logical form accuracy of the generated queries were measured. An execution accuracy of up to 59.4% on the test set could be achieved for the WikiSQL dataset. One downside of this training and evaluation method is that the query might not correspond to the NL question, even though the correct query result is obtained.

Luz et al. [8] also used an LSTM encoder-decoder model with the purpose to encode NL and decode to 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 NL 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 syntactic errors. While they obtained reasonable results to comparable approaches, they did not handle the out of vocabulary issue and lexical ambiguities.

Soru et al. [9, 10] proposed a generator-learner-interpreter architecture, called *Neural SPARQL Machines* (NSpM), to translate any NL 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 (KB) and generated pairs of NL expression and SPARQL query accordingly. After encoding operators, brackets, and URIs contained in original SPARQL queries, the pairs were fed as training data to an S2S learner model. The model was able to generalize to unseen NL sentences and generate encoding sequences of SPARQL for the interpreter to decode, with a BLEU score of up to 80.3 on the Monument dataset. Given its recent publication and state-of-the-art architecture, this model serves as a baseline in our comparison.

4. Methodology

To test the use of NMT models on the translation from NL to SPARQL, the SPARQL queries need to be encoded in a way that they can be fed to an S2S model. We first describe this encoding before detailing the three different types of state-of-the-art neural network architectures considered in this comparison and their respective specific implementations. Finally, we detail the evaluation method to compare the performance of the chosen NMT models on this specific task of translating from NL to SPARQL.

4.1. SPARQL encoding

Unlike NL that can easily be tokenized, SPARQL queries are internally structured, combining elements of the query language with elements from the RDF data store and variables. Thus, our first step is to encode each query as a sequence. Following the encoding approach suggested by Soru et al. [9], URIs are abbreviated using their prefixes (if necessary) and concatenated with the entities using underscores; brackets, wildcards, and dots are replaced by their verbal description, and SPARQL operators are lower-cased and represented by a specified number of tokens. These operations can be implemented as a set of replacements and applying them turns an original SPARQL query to a final token sequence containing only characters and underscores. An example SPARQL query for the NL question "*For which games are Sam Loyd and Eric Schiller both famous?*" is provided in Listing 1 and its encoding to be utilized in the NMT architectures is shown in Listing 2. The training thus consists in translating from the NL question to the type of encoded SPARQL sequence exemplified in Listing 2. After training, when an encoded form of a SPARQL query has been generated, it can be easily decoded back by reversing the encoding replacements.

Listing 1: Example SPARQL query

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT DISTINCT ?uri
WHERE {
dbr:Sam_Loyd dbo:knownFor ?uri .
dbr:Eric_Schiller dbo:knownFor ?uri . }
```

Listing 2: Encoding of SPARQL query from Listing 1

```
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
```

4.2. Description of tested NMT models

We compare three types of network architectures with a total of eight individual NMT models. The three types are RNN-based, CNN-based, and Transformer-based models since those represented the best performing NMT architectures in the field at the time of the experiment without considering hybrid and ensemble methods. Encoded SPARQL queries and NL questions are fed to the network on a word-level basis.

4.2.1. RNN-based models

Many variants of RNN-based models exist, since they were considered the most natural choice for S2S learning for a long time. The main differences can be found in the number of hidden layers, types of units, and other main architectural decisions. In a closely related approach, a two-layer LSTM network has been used to translate from NL to SPARQL [9], the NSpM model presented in Section 3, which also serves as our baseline model.

In a first experiment, the effects of distinct types of attention on the baseline model are investigated. Global attention [15] (NSpM+Att1) is compared to local attention [16] (NSpM+Att2) to evaluate their impact. Moreover, we adopt the model proposed by Luong et al. [16] (LSTM_Luong) and the GNMT system proposed by Wu et al. [13], both of which achieved state-of-the-art results on NL translation benchmarks. The former is essentially a deeper LSTM with 4 layers and a local attention mechanism. For the GNMT we differentiate between a model with four layers (GNMT-4) and one with eight layers (GNMT-8). Starting from the third layer, the GNMT architecture uses residual connections to remedy the loss of information caused by the transgression through many layers. Therefore, input and output of the LSTM cell are added and serve as input to the next layer. In addition, GNMT utilizes a bi-directional RNN on the first layer of the encoder, that is, the input sequence is read from left to right and right to left and combined before giving it to the next layer.

4.2.2. CNN-based models

RNNs suffer from a decrease in performance for longer sequences, high computational costs, and have issues with rare words. CNN-based models are capable of overcoming some of those issues. Long-range dependencies have shorter paths when the inputs are processed in a hierarchical multi-layer CNN as opposed to the chain structure of RNNs. A CNN is able to create representations for n continuous words in $\mathcal{O}(\frac{n}{k})$ convolutions with k-width kernels, while an RNN needs $\mathcal{O}(n)$. CNNs also allow for faster training since they permit parallelization over all elements in a sequence, whereas the computations in RNNs are sequentially dependent on each other. On the other hand, the input needs to be padded to serve as input to the model, since CNNs can only process sequences of fixed length. An additional position encoding is required to provide the model with a sense of ordering of the input elements.

One of the first CNN-based approaches to address an S2S problem is the Convolutional S2S (ConvS2S) [17] model. It is still an encoder-decoder architecture with attention, where both consist of stacked convolutional blocks. Each block is composed of a single dimensional convolutional layer followed by a Gated Linear Unit (GLU). The input to the convolution block can either be the output of the previous layer or, for the bottom layer of the encoder, the combined word and position embeddings. Note that because each convolutional block can only receive a fixed number of inputs, the input of each block needs to be padded with zero vectors. There is a residual connection between the input to the convolutional layer and the output of the GLU. Compared to the encoder, each convolutional block in the decoder has one more attention module located after the output of GLU and before the residual connection. This type of attention is considered multi-step attention, since the higher layers each compute attention individually and have access to the information which elements the lower layers attended to.

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 suffices to make this model faster than most of its RNN rivals [17].

4.2.3. Transformer models

Transformer models are highly popular due to their simple and effective architecture without a need for convolution or recurrence. Each layer of the encoder and decoder in this architecture has two sub-layers: a multi-head self-attention mechanism and a point-wise fully connected feed-forward network, with an additional, third multi-head attention layer in the decoder over the encoder stack [18]. In multi-head attention, the input is composed of three parts: queries, keys, and values which are all vectors. Each head performs a scaled dot-product attention that maps from queries and a set of key-value pairs to an output attention matrix, where the queries and all keys are computed first, then scaled, and finally put through a softmax function to obtain weights on the different positions of values. At the connection between encoder and decoder, the queries come from the previous decoder layer while the keys and values represent the outputs from the encoder. This model shows both quality and speed advantages and has achieved state-of-the-art results on multiple translation tasks.

4.3. Evaluation

To compare the outputs of the eight models in this paper, we employ a combination of evaluation metrics. First, we use the common NMT metric called BLEU score, which is detailed in Section 2.3. While BLEU shows high correlations with human evaluations and low marginal computational costs [19], it does not account for word order. To this end, the exact string matching accuracy is computed as well as the F1 score based on individual syntactical elements of the candidate and target translation. In addition, we utilize perplexity to show the model's intrinsic behavior which represents the exponentiation of the cross entropy to the base of two. A better language model is one that assigns higher probabilities to the words that actually occur.

5. Experimental setup

Experimental details of the conducted experiments consist of selected datasets, NMT frameworks utilized for the implementation, dataset splits and hyperparamter settings. This section additionally describes the runtime environments on which training and test runs were performed.

5.1. Datasets

To successfully train an NMT model, a large-quantity bilingual parallel corpus is needed. Our setting equally requires a large parallel corpus, however, with NL questions aligned with their corresponding SPARQL queries. In terms of NL, we only consider English in this evaluation. Some difficulties in creating such a dataset are that expertise in SPARQL is required, knowledge about the underlying database to be queried is necessary (e.g. "located at" is represented as `dbo:location` in DBpedia), and changes to the whole KB can affect the validity of the dataset. The three datasets

Table 1: A template pair in the Monument dataset

Question template	Query template
Where is <A> ?	`SELECT ?x` `WHERE` `{ <A> dbo:location ?x . }`

used were constructed by creating a list of template pairs (see Table 1) with placeholders and then replacing the placeholders with extracted entities or predicates from the latest endpoint of an online KB. Due to this limitation and the complexity of SPARQL itself, only a subset of SPARQL operators are included in the target SPARQL queries of the involved datasets, which are: `SELECT`, `ASK`, `DISTINCT`, `WHERE`, `FILTER`, `ORDER BY`, `LIMIT`, `GROUP BY`, and `UNION`.

The Monument dataset is generated and used by the NSpM [9] system. It has 14,788 question-query pairs. 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 the specific class `dbo:Monument`, which is why we call it the Monument dataset. The data is generated from a list of manually crafted template pairs and related assistant SPARQL queries that can be executed directly on a DBpedia endpoint. We briefly exemplify the data generation method performed by Soru et al. [9] to create the Monument dataset. For example, given a template pair in Table 1, where <A> belongs to the class `dbo:Monument` in DBpedia, one can then retrieve a list of entities and their corresponding English labels to replace <A> by executing an assistant SPARQL query on a DBpedia endpoint. An example of a result of an assistant query to perform such a replacement in the query template is shown in Table 2. Soru et al. [9] report that 38 manually annotated templates were used in generating the Monument dataset. For each query template, they generated 600 examples with the method described and exemplified above. Some English templates are partial phrases instead of full sentence (e.g. latitude of <something>).

The LC-QUAD [11] contains 5,000 pairs, in which about 7,000 English words and 5,000 SPARQL tokens are used. The method for generating SPARQL tokens is pre-

Table 2: An example result from running an assistant query to instantiate the template in Table 1

?uri	`http://dbpedia.org/resource/Carew_Cross`
?label	“Carew Cross"@en
Generated question	Where is Carew Cross ?

Generated query	`SELECT ?x`
	`WHERE`
	`{ http://dbpedia.org/resource/Carew_Cross`
	`  dbo:location ?x . }`

sented in Section 4.1 in detail. The SPARQL queries can be applied to query 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 produce this dataset, 38 unique templates as well as 5,042 entities and 615 predicates from DBpedia were involved in the generation workflow.

In contrast to the Monument dataset generation method, LC-QUAD allocates executable SPARQL queries to each English-SPARQL template pair to retrieve a list of entity instances. Using a previously prepared entity seed list and a predicate whitelist, DBpedia subgraphs are extracted using a generic SPARQL query. The triples in the subgraphs are utilized to instantiate the SPARQL and English templates. The final result is peer reviewed to ensure grammatical correctness.

To the best of our knowledge and up to the completion of our experiments, DBNQA [12] is the largest DBpedia-targeting dataset and a superset of the Monument dataset. It is also based on English and SPARQL pairs and contains 894,499 instances in total. In terms of vocabulary, it contains approximately 131,000 words for English and 244,900 tokens for SPARQL without any reduction. DBNQA provides a remedy for some drawbacks of the previous two datasets. A large number of generic templates are extracted from the concrete examples of two existing datasets LC-QUAD and QALD-7-Train [21] by replacing the entities with placeholders. These templates can subsequently be utilized in the same dataset generation method as applied to the Monument dataset to generate a large dataset. One drawback of such a large dataset is the problem of memory shortages due to the large vocabulary, which nevertheless is considerably smaller than common NL datasets, making a comparison to NL translation tasks more difficult.

5.2. Frameworks

Two frameworks are used in this comparison due to their popularity in NMT, one based on TensorFlow and one on PyTorch. TensorFlow NMT[2] provides a flexible implementation of the RNN-based NMT models. One can easily build and train a variety of 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 training and testing five different models including three baseline 2-layer LSTMs, a 4-layer GNMT, and an 8-layer GNMT. The Facebook AI Research S2S Toolkit[3] (Fairseq) [17] is another framework that implements various S2S models based on PyTorch. Fairseq provides off-the-shelf models as well as packed hyperparameters to configure user experiments. We used it to train and test three models including the 4-layer LSTM with attention proposed by Luong et al. [16], the ConvS2S, and the Transformer.

5.3. Dataset splits and hyperparameters

We split each dataset in a ratio of 80%-10%-10% for training, validation, and test. Additionally, we perform two splits on the Monument dataset. First, we split at a ratio of 50%-10%-40% for training, validation, and test to evaluate the complexity of the dataset (Monument50), and second, we use the splitting approach in Soru et al. [9] to directly compare our results with NSpM (Monument). The latter split essentially fixes 100 examples for both validation and test set and keeps the rest for the training set. Each time we test a new dataset split, the models are trained from scratch, which means test data are always previously unseen in training. In summary, we have five experimental datasets, namely: Monument, Monument50, Monument80, LC-QUAD, and DBNQA.

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

[3]Available at `https://github.com/pytorch/fairseq`

Table 3: Hyperparameter settings of this experiment

Name	Layers	H. Units	Attention	Optimizer	LR	Dropout	Size
NSpM[4]	2	128	-	SGD	1	0.2	50,000
NSpM + Att1	2	128	yes	SGD	1	0.2	50,000
NSpM + Att2	2	128	yes	SGD	1	0.2	50,000
GNMT-4	4	1,024	yes	SGD	1	0.2	30,000
GNMT-8	8	1,024	yes	SGD	1	0.2	30,000
LSTM_Luong	4	1,000	yes	Adam	0.001	0.3	500
ConvS2S	15	512*	yes	SGD	0.5	0.2	500
Transformer	6	1,024	yes	Adam	0.0005	0.3	500

* First 9 layers: 512 kernel width 3; next 4 layers: 1,024 kernel width 3, final two: 2,048 kernel width 1

The hyperparameter settings for all models correspond to the original, best-performing NL NMT settings and are as shown in Table 3, which details the number of layers for encoder and decoder, the number of hidden units, whether the model uses an attention mechanism, which optimizer it uses with which learning rate (LR), and the applied dropout rate. The final column *Size* refers to maximum training steps in TensorFlow NMT (the first 5 lines) and to the maximum epochs in Fairseq (the last three lines). The training is based on cross entropy loss minimization. For the decoding, beam search of beam width 5 is used for all experiments.

5.4. Runtime environment

Given that we have 40 different experiments, where each experiment consumes different amounts of memory depending on the size of its model and dataset, we assigned them to three GPUs with memory capacity from small to large running on a High Performance Computing (HPC) server. Their configurations are listed in Table 4. In terms of assignment of the resources to the individual models, we used only the small configuration for all models and datasets with the exception of the GNMT-8 dataset, which always utilized the medium setting, and the exception of DBNQA. For DBNQA the first three models in Table 3 were run in the medium configuration, while all other models utilized the large configuration. Training was completed using Linux

Table 4: Three hardware configurations on the HPC server used in this paper

	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

with Python 3.6.4, TensorFlow 1.8.0, and PyTorch 0.4.1 installed.

6. Results

In order to reflect on how well our models have been trained on different datasets, we report the perplexity for each experiment along with the training steps (in TensorFlow NMT) or epochs (in Fairseq). During training, we stored the model checkpoints of the step or epoch, which led to the best[5] performance on the validation set in a model checkpoint file. This best performing checkpoint model was then used to perform decoding on the test set and report the BLEU scores. This section describes the statistical results of the 40 experiments, while their analysis and implications are discussed in Section 7. Details on the results and corresponding visualizations are available at `https://github.com/xiaoyuin/tntspa`.

6.1. Perplexities

All final perplexity scores per model and dataset are reported in Table 5, where *T* represents training scores and *V* refers to validation scores. In *Mon*, which represents the Monument dataset, a full epoch for Fairseq is approximately equivalent to 120

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

[5]Depending on the support of the frameworks, we stored the one with best validation BLEU in TensorFlow NMT and the one with best validation loss in Fairseq.

Table 5: Perplexity scores for all models and all training and validation sets

	Mon		Mon80		Mon50		LC-QUAD		DBNQA	
Models	T	V	T	V	T	V	T	V	T	V
NSpM	1.00	1.09	1.00	1.21	1.00	1.29	1.00	16.46	2.05	2.32
NSpM+Att1	1.01	1.16	1.00	1.44	1.00	1.62	1.00	56.23	1.07	1.42
NSpM+Att2	1.01	1.14	1.00	1.44	1.00	1.62	1.00	43.20	1.05	1.37
GNMT-4	1.03	1.11	1.00	1.31	1.00	1.41	1.00	33.76	1.74	2.24
GNMT-8	1.04	1.15	1.01	1.32	1.00	1.59	1.01	229.96	2.13	2.43
LSTM_Luong	1.11	1.19	1.11	1.24	1.11	1.26	1.12	4.92	1.90	2.15
ConvS2S	1.11	1.14	1.11	1.19	1.11	1.20	1.14	3.25	1.12	1.25
Transformer	1.13	1.09	1.12	1.14	1.14	1.17	1.16	3.15	2.21	3.34

steps for TensorFlow NMT. One epoch in Fairseq (also in TensorFlow NMT) is a full iteration over all the examples in the training set. One step in TensorFlow NMT means a batch of examples from the training set. For instance for the Monument dataset, there are in total 14,588 examples, which with a batch size of 128 corresponds approximately to 120 steps. For this dataset, all perplexities approach one. An unusual phenomenon can be observed for the Transformer model, where the validation perplexity becomes lower than the training perplexity. For the Monument80 dataset, around 100 steps in the TensorFlow NMT are equivalent to an epoch in the Fairseq models. With this dataset with less training examples, the validation perplexities are mostly higher than with the Monument dataset. Slight overfitting can be observed in the validation loss curve of the two NSpM with attention and the two GNMT models. Although the training datasetset size in the Monument50 dataset is reduced by half, it appears that the performance on the validation set is not much affected, especially in case of LSTM_Luong, ConvS2S, and Transformer all implemented in Fairseq, which is also reflected by their BLEU scores reported below.

In the LC-QUAD dataset, we observed serious overfitting of all eight models. Most of the models have difficulty in providing low perplexity on the validation set, among which GNMT-8 performed the worst and ConvS2S the best. LC-QUAD has comparatively more query templates with fewer examples for each template, which makes it

Table 6: BLEU scores for all models and validation and test sets

	Mon		Mon80		Mon50		LC-QUAD		DBNQA	
Models	V	T	V	T	V	T	V	T	V	T
NSpM	80.43	80.28	87.55	87.03	85.19	85.54	43.91	43.50	65.89	65.92
NSpM+Att1	80.36	80.58	87.82	87.34	85.98	86.17	52.68	50.13	89.87	89.87
NSpM+Att2	80.88	80.03	87.99	87.37	86.60	86.52	53.03	50.86	91.51	91.50
GNMT-4	80.12	79.53	85.94	85.39	82.92	83.01	43.69	42.71	69.65	69.61
GNMT-8	79.30	79.07	84.94	84.14	80.35	80.76	44.32	43.91	68.43	68.41
LSTM_Luong	92.39	91.67	96.35	96.12	94.05	94.75	52.43	51.06	77.64	77.67
ConvS2S	**98.35**	**97.12**	**96.74**	**96.47**	**96.44**	**96.62**	**61.89**	**59.54**	**96.05**	**96.07**
Transformer	95.25	95.31	95.16	94.87	93.80	93.92	58.99	57.43	68.68	68.82

a harder dataset for training neural network models. It only takes 34 steps to finish training an epoch inside the Tensorflow NMT framework. Across all of the models, no evident overfitting is spotted for the DBNQA dataset. In a batch size of 128, an epoch of DBNQA takes nearly 5,600 training 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. Nevertheless, all of the models have reached a perplexity of at least 2 on the validation set, where NSpM+Att1, NSpM+Att2, and ConvS2S have achieved a perplexity lower than 2, which is also reflected in their accuracy below.

6.2. BLEU scores

BLEU scores on the validation *V* and test *T* set for each dataset for the best performing version of each model are reported in Table 6. ConvS2S outperforms all other models on all datasets. On all Monument datasets, the Fairseq models outperform the Tensorflow NMT models by a large margin. On the other two datasets, attention-equipped NSpM models perform equivalent to or even better than the Tensorflow NMT models. For the DBNQA dataset, they even come second place after ConvS2S. The low scores on the LC-QUAD dataset are also consistent with the high perplexity scores on this dataset.

Table 7: Accuracy (in %) of syntactically correct generated SPARQL queries and F1 score

	Mon				Mon80				Mon50				LC-QUAD				DBNQA			
Models	V		T		V		T		V		T		V		T		V		T	
	%	F1	%	F1	%	F1	%	F1	%	F1	%	F1	%	F1	%	F1	%	F1	%	F1
NSpM	71	95	75	93	75	95	76	95	82	97	79	96	0	61	0	61	0	77	0	77
NSpM+Att1	71	95	75	93	77	96	78	96	83	97	82	97	1	68	1	66	63	93	63	93
NSpM+Att2	73	96	74	92	79	97	78	96	84	97	81	97	1	68	1	67	69	94	69	94
GNMT-4	70	95	71	92	67	95	68	95	77	96	75	96	0	62	0	61	1	84	1	84
GNMT-8	68	95	73	91	58	94	60	94	74	96	71	95	0	65	0	64	0	84	0	84
LSTM_Luong	75	94	76	94	82	95	84	96	**90**	**98**	89	97	0	68	0	67	34	82	34	82
ConvS2S	**94**	**99**	**95**	**96**	**91**	**98**	**90**	**98**	**89**	**98**	**90**	**98**	**8**	**74**	**8**	**73**	**85**	**98**	**85**	**97**
Transformer	88	98	91	95	83	96	84	96	86	92	84	92	7	71	4	70	3	79	3	80

6.3. Accuracy

To take word order of the produced query into account, we count all produced queries that exactly match the target query and divide that count by the number of all queries in each subset of the datasets. The accuracy results and F1 scores presented in Table 7 show that also for these measures, ConvS2S consistently outperforms all other models. Only once the LSTM_Luong model produces a validation set accuracy equivalent to that of the best performer. These results also show the drastic problems with the LC-QUAD dataset, where most models fail to produce a single fully equivalent query and highlight the problems of some models on the DBNQA dataset. In the latter dataset, a positive effect of attention becomes evident with respect to the baseline model NSpM.

7. Discussion

A comparison across the utilized datasets shows some explicit trends. One of the reasons for utilizing the Monument dataset is to allow for a direct comparison of the baseline NSpM model [9] with other NMT models. After training the NSpM, the results of a BLEU test score of 80 could be reproduced. Adding attention would be expected to outperform the baseline, which could not be observed in the BLEU scores. In the accuracy measure, attention mechanisms show their potential, since NSpM+Att2 (local attention) consistently outperforms the baseline. The GNMT models could not outperform this baseline. Changing the dataset split led to an increase in BLEU scores. We believe this is due to the increase of the number of examples in the validation and test set. Additionally, the difference between Monument80 and Monument50 in BLEU score is only 1-2 on average, which means that the models are still able to achieve a good performance after training from dramatically smaller proportions of the whole dataset. All of the above suggests that the Monument dataset has little variation in sentence structures, which is expected from its generation methods (see Section 5.1).

A dataset with higher variance is provided by LC-QUAD. Due to its relatively small size and high diversity of NL and SPARQL templates it seems considerably harder to synthesize for training neural models. All of the models experienced difficulties in training and showed low performances on the validation set. Especially in terms of accuracy most models fail to produce one fully equivalent and correctly ordered query. It could be the case that the data contained in the training set and validation set are somewhat distinct in types (i.e. using different templates) and the number of training samples is not enough to generalize the model to unseen data. In addition, even though the dataset was peer reviewed, some grammatical errors still exist and some questions contain unusual punctuation that leads to undetectable incorrect tokenizations during vocabulary building (e.g. “the u.n.i.t.y group" will be split into seven tokens “the, u, n, i, t, y, group").

Given the above, we found DBNQA better suited to the task. It has a sufficient number of query types (i.e. templates) as well as volume for each template. From Table 5 and 6, we found that the training of the models was overall normal and the results have shown the differences between the models. One unexpectedly low performance could be observed by the Transformer model. This might be attributed to a worse performance of self-attention with a very large vocabulary, which could not be observed with the other two attention-based models. However, when looking at accuracy, all but the attention-based and the ConvS2S models experience serious problems in producing a sequentially correctly ordered query. While we still believe that the DBNQA dataset is the best choice for training NMT models to translate from NL to SPARQL, the dataset also has obvious limitations. The vocabulary size is too large, which may be attributed to its generation method (see Section 5.1) that might have instantiated the templates with too many distinct entities. It was necessary to carefully consider this problem in our approach since a simple reduction in vocabulary size would certainly result in a drastic increase of unknown tokens in source and target sequences. In order to handle this large vocabulary size we had to move to a more powerful HPC cluster partition (see GPU Large in Table 4) and reduce the training epochs considerably (see Section 6 for details), since training was extremely time- and resource-consuming. One possible other solution could be to adopt the dataset generation approach proposed in LC-QUAD (see Section 5.1) which somewhat restricts the number of involved entities and predicates from the beginning.

In terms of model performance, there is a clear winner in our comparison. From the results, it becomes evident that the ConvS2S model performs best on this specific task of translating NL to SPARQL. The ConvS2S model outperformed other models in BLEU scores on all of the datasets and achieved the fastest convergence and highest accuracy because it is able to achieve parallelized computations. One thing we did not expect is that the performance of GNMT models was consistently below average on all of the datasets. Looking into the perplexity graphs, we found that GNMT-8 was the slowest to converge and suffered relatively large fluctuations. Although reducing the number of layers to four in GNMT-4 considerably improved the converging speed, it still struggled to achieve good results in BLEU scores and accuracy. We speculate that this might be due to the rather complicated architecture of GNMT, which to some extent may magnify the degree of overfitting.

Given the popularity of the attention mechanism, a direct comparison of its effect could be obtained by equipping the baseline model with two different types of attention. Figure 1 shows the comparison between three baseline RNN-based models on their test BLEU scores. The attention enhanced NSpM models generally performed equally to or better than the original NSpM. Furthermore, we found that NSpM+Att2 (local multiplicative attention) slightly outperformed the NSpM+Att1 (global additive attention) on the LC-QUAD and DBNQA datasets. Therefore, we believe that the

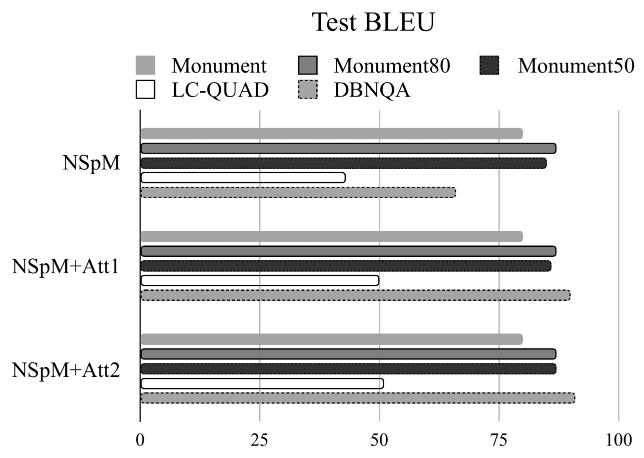

Figure 1: The comparison between three NSpM models on test BLEU scores

attention mechanism can indeed boost the translation performance on the task of translating NL to SPARQL, similar to traditional NMT tasks.

Compared to their original task of translating between NLs, the models performed better on this task. The ConvS2S merely achieved a BLEU score of 26.43 at best on the WMT'14 English-German dataset [17], while it is giving a highest of 97.12 and lowest of 59.54 in our experiments. The overall better performance of NMT models on this task of NL to SPARQL translation can be explained by the significant difference in the complexity of the utilized datasets. The composition of SPARQL queries contained in our datasets does not vary much, e.g. they all start with the same headings such as `SELECT DISTINCT`.

In terms of evaluation metric, there is room for improvement. A clear correlation between perplexity and BLEU score could only be verified for the DBNQA dataset, but not for the other datasets. Experiments with accuracy revealed that the BLEU score is not entirely reliable when the word order of the output sequence is important. While models still obtained reasonable BLEU scores on the LC-QUAD and DBNQA datasets, at times not a single output query was entirely correct. SPARQL syntax elements were generally produced correctly, however, problems with DBpedia entities could be observed, such as the models replacing `dbr_Radiant_Silvergun` with `dbr_Ella_Fitzgerald`. Interestingly, it could be observed that this erroneous replacement of entities was done consistently across all output queries, always replacing the correct entity with the same erroneous one. While this combination of BLEU score and accuracy provides a good estimation of model performance, it would be beneficial to devise an evaluation metric specific to this task or for translating to structured languages in general. To compare the performances of the models in more detail, it would be necessary to fine-tune the hyperparameter settings to the task at hand specifically. This experiment, instead, utilized the parameters provided for translating between NLs.

8. Conclusion and future work

In this comparative study on using NMT models for automatically translating from NL to SPARQL queries, three representative neural network architectures were selected and eight variations thereof were tested. All of these models were trained and tested on three different datasets and evaluated using perplexity, BLEU scores, and a simple accuracy measure to ensure the correct word order of the resulting SPARQL queries. As a result, we found that the ConvS2S model consistently, significantly outperformed all other models at a margin. As a side note it has to be considered that we did not fine-tune the hyperparameters for each model, which, when done, might affect the results. However, ConvS2S was also not fine-tuned and provided the best results, which is why we believe it to be a solid choice for this task. In terms of dataset, the large and recent DBNQA dataset proved to be most adequate for the task at hand.

Even though the DBNQA dataset allowed the models to improve their performance largely, it still suffers from limitations regarding the complexity of questions and queries and vocabulary size. One potential direction for future work could be to advance template discovery techniques to increase the number of templates and combine these techniques with the query generation approach of DBNQA in order to derive a large but also more diverse dataset. Furthermore, an evaluation metric specifically targeted towards the evaluation of structured queries could strongly improve the comparison of neural models for this task. It would also be desirable to repeat similar experiments within the same framework and training environment for a more controlled comparison. Testing transfer learning approaches on the trained models could equally be interesting, especially on datasets with differing data distributions. Finally, these experiments were restricted to the English language. It would be interesting to see changes in the performance when utilizing a different NL to generate SPARQL queries.

Acknowledgements

We would like to thank the reviewers and guest editors for their valuable comments. Part of this work has been supported by the NexusLinguarum COST-Action (CA18209) for Web-centered linguistic data science and the pilot project Text2TCS funded by the European Language Grid (Grant agreement Nr. 825627).

References

[1] A. Seaborne, S. Harris, SPARQL 1.1 Overview, W3C Recommendation, W3C, http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/ (Mar. 2013).

[2] G. Schreiber, Y. Raimond, RDF 1.1 Primer, W3C Recommendation, W3C, http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/ (Jun. 2014).

[3] S. Ferré, squall2sparql: A translator from controlled english to full SPARQL 1.1, in: P. Forner, R. Navigli, D. Tufis, N. Ferro (Eds.), Working Notes for CLEF 2013 Conference, Vol. 1179 of CEUR Workshop Proceedings, CEUR-WS.org, 2013.

[4] M. Dubey, S. Dasgupta, A. Sharma, K. Höffner, J. Lehmann, AskNow: A framework for natural language query formalization in SPARQL, in: H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S. P. Ponzetto, C. Lange (Eds.), Proceedings of the 13th European Semantic Web Conference (ESWC), Vol. 9678 of LNCS, Springer, 2016, pp. 300–316. doi:10.1007/978-3-319-34129-3_19.

[5] R. Cai, B. Xu, Z. Zhang, X. Yang, Z. Li, Z. Liang, An encoder-decoder framework translating natural language to database queries, in: J. Lang (Ed.), Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI), IJCAI, 2018, pp. 3977–3983. doi:10.24963/ijcai.2018/553.

[6] L. Dong, M. Lapata, Language to logical form with neural attention, in: K. Erk, N. A. Smith (Eds.), Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL, 2016, pp. 33–43. doi:10.18653/v1/P16-1004.

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

[8] F. F. Luz, M. Finger, Semantic parsing natural language into SPARQL: Improving target language representation with neural attention, CoRR abs/1803.04329. arXiv:1803.04329, 2018.

[9] T. Soru, E. Marx, D. Moussallem, G. Publio, A. Valdestilhas, D. Esteves, C. B. Neto, SPARQL as a foreign language, in: J. D. Fernández, S. Hellmann (Eds.), Proceedings of the Posters and Demos Track of the 13th International Conference on Semantic Systems, Vol. 2044 of CEUR Workshop Proceedings, CEUR-WS.org, 2017.

[10] T. Soru, E. Marx, A. Valdestilhas, D. Esteves, D. Moussallem, G. Publio, Neural machine translation for query construction and composition, CoRR abs/1806.10478. arXiv:1806.10478, 2018.

[11] P. Trivedi, G. Maheshwari, M. Dubey, J. Lehmann, LC-QuAD: A corpus for complex question answering over knowledge graphs, in: C. d'Amato, M. Fernández, V. A. M. Tamma, F. Lécué, P. Cudré-Mauroux, J. F. Sequeda, C. Lange, J. Heflin (Eds.), Proceedings of the 16th International Semantic Web Conference (ISWC), Vol. 10588 of LNCS, Springer, 2017, pp. 210–218. doi:10.1007/978-3-319-68204-4_22.

[12] A.-K. Hartmann, E. Marx, T. Soru, Generating a large dataset for neural question answering over the DBpedia knowledge base, in: Workshop on Linked Data Management, co-located with the W3C WEBBR 2018, 2018.

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

[14] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in: Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 27, Curran Associates, Inc., 2014, pp. 3104–3112.

[15] D. Bahdanau, K. Cho, Y. Bengio, Neural machine translation by jointly learning to align and translate, in: Y. Bengio, Y. LeCun (Eds.), Proceedings of the 3rd International Conference on Learning Representations (ICLR), iclr.cc, 2015.

[16] T. Luong, H. Pham, C. D. Manning, Effective approaches to attention-based neural machine translation, in: L. Màrquez, C. Callison-Burch, J. Su, D. Pighin, Y. Marton (Eds.), Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP), ACL, 2015, pp. 1412–1421. doi:10.18653/v1/D15-1166.

[17] J. Gehring, M. Auli, D. Grangier, D. Yarats, Y. N. Dauphin, Convolutional sequence to sequence learning, in: D. Precup, Y. W. Teh (Eds.), Proceedings of the 34th International Conference on Machine Learning (ICML), Vol. 70 of Proceedings of Machine Learning Research, PMLR, 2017, pp. 1243–1252.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, pp. 5998–6008.

[19] K. Papineni, S. Roukos, T. Ward, W. Zhu, BLEU: A method for automatic evaluation of machine translation, in: P. Isabelle, E. Charniak, D. Lin (Eds.), Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, ACL, 2002, pp. 311–318. doi:10.3115/1073083.1073135.

[20] T. Luong, M. Kayser, C. D. Manning, Deep neural language models for machine translation, in: A. Alishahi, A. Moschitti (Eds.), Proceedings of the 19th Conference on Computational Natural Language Learning (CoNLL), ACL, 2015, pp. 305–309. doi:10.18653/v1/k15-1031.

[21] R. Usbeck, A.-C. N. Ngomo, B. Haarmann, A. Krithara, M. Röder, G. Napolitano, 7th open challenge on question answering over linked data QUALD-7, in: M. Dragoni, M. Solanki, E. Blomqvist (Eds.), Semantic Web Evaluation Challenges, Vol. 769 of Communications in Computer and Information Science, Springer, 2017, pp. 59–69. doi:10.1007/978-3-319-69146-6_6.