Neural Network-based Question Answering over Knowledge Graphs on Word and Character Level

Denis Lukovnikov
University of Bonn
lukovnik@cs.uni-bonn.de

Asja Fischer
University of Bonn
fischera@iai.uni-bonn.de

Jens Lehmann
University of Bonn
lehmann@uni-bonn.de

Sören Auer
University of Bonn
auer@cs.uni-bonn.de

ABSTRACT

Question Answering (QA) systems over Knowledge Graphs (KG) automatically answer natural language questions using facts contained in a knowledge graph. Simple questions, which can be answered by the extraction of a single fact, constitute a large part of questions asked on the web but still pose challenges to QA systems, especially when asked against a large knowledge resource. Existing QA systems usually rely on various components each specialised in solving different sub-tasks of the problem (such as segmentation, entity recognition, disambiguation, and relation classification etc.). In this work, we follow a quite different approach: We train a neural network for answering simple questions in an end-to-end manner, leaving all decisions to the model. It learns to rank subject-predicate pairs to enable the retrieval of relevant facts given a question. The network contains a nested word/character-level question encoder which allows to handle out-of-vocabulary and rare word problems while still being able to exploit word-level semantics. Our approach achieves results competitive with state-of-the-art end-to-end approaches that rely on an attention mechanism.

Keywords

Question Answering, Knowledge Graphs

1. INTRODUCTION

The ever increasing availability of data and information requires significant advances in empowering users to master this new wealth of information. In particular, knowledge graphs (KG) comprising vast amounts of facts gain increasing importance. Examples are open and crowd-sourced knowledge graphs such as *DBpedia* [15][1] or *WikiData*[2], but also proprietary enterprise knowledge graphs such as Google's Knowledge Graph (comprising more than 70 billion facts), Microsoft Bing's Satori Knowledge Base[3] or Yandex' Object Answer[4]. Knowledge graphs are usually very large and not easily accessible for users as they need to know a query language as well as the the structure and relations in the knowledge graph. Question Answering (QA) over KGs, which automatically translates natural language questions (or fragments thereof) posed by humans into structured queries (e.g. SPARQL) is an easy way for users to access the vast and increasing amounts of information stored in such KGs.

Despite significant research efforts question answering is still a challenge. Even in well-researched domains, such as general domain factoid question answering in English, existing approaches struggle to match human level understanding of questions. Languages other than English and more specialized domains, such as legal or biomedical data, present additional challenges, making the task even more difficult.

Some of the challenges faced by QA systems are:

- *Lexical gap* – surface forms for relations expressed in a question can be quite different from those used in the KG,
- *Ambiguity* – the same word is used for different entities, such as president of a company or a country,
- *Unknown knowledge boundaries* – QA systems can often hardly decide whether a certain question is answerable at all give a certain knowledge base.

The traditional paradigm for QA approaches is (1) to construct complex natural language processing (NLP) pipelines (which can result in error propagation along the pipeline) and (2) require manual adaption to new domains. *AskNow* [10], for example, uses a pipeline comprising a POS tagger, template-fitting, relation extraction, token merging and entity mapping. The *Qanary* methodology [5] follows an extreme realization of this approach by enabling a fine-grained integration, cooperation and competition of pipeline components.

In this work, we follow a fundamentally different approach that mitigates the main disadvantages of the pipeline method. We develop an end-to-end neural network approach that generates a solution in a single process and thus avoids (1) complex NLP pipeline constructions, (2) error propagation,

[1] `http://dbpedia.org`
[2] `https://www.wikidata.org`

[3] `http://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/`
[4] `https://yandex.ru/company/technologies/entity_search`

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052675

and (3) can be retrained or reused for a different domain. All decisions can be handled together in an integrated fashion in order to leave the learning algorithm the freedom to decide how to use the given information.

In this work, we restrict ourselves to simple questions, which only require the retrieval of a single fact to be answered. This is the setup of the SIMPLEQUESTIONS task recently presented by Bordes et al. [4]. The task of QA over KG for simple questions (Simple QA) can be put more formally as follows. Let $\mathcal{G} = \{(s_i, p_i, o_i)\}$ be a background knowledge graph represented as a set of triples, where s_i represents a subject entity, p_i a predicate (also denoted as relation), and o_i an object entity. The task of Simple QA is then: Given a natural language question represented as a sequence of words $q = \{w_1, \ldots, w_T\}$, find a triple $(\hat{s}, \hat{p}, \hat{o}) \in \mathcal{G}$ such that $\hat{o}$ is the intended answer for question q. This task can be reformulated to finding the right subject $\hat{s}$ and predicate $\hat{p}$ that question q refers to and which characterize the set of triples in $\mathcal{G}$ that contain the answer to q.

For example, finding the answer to the question "*What cyclone affected Hainan?*" requires finding the FREEBASE entity `m.0166br` representing the Chinese province Hainan and the relation `fb:meteorology/affected_area/cyclone`. The entity representing Hainan has the label "*Hainan*" and a notable type with label "*Chinese province*".

We develop a holistic neural matching model that takes basic textual information about entities and predicates and uses some training and prediction enhancements to produce results competitive to the state-of-the-art. Even though the presented approach is restricted to simple questions, this work can serve as a foundation for the future development of more advanced neural QA approaches that can handle more complex questions.

Compared to recent existing approaches [26, 12, 8], our model is relatively simple in that it does not employ attention mechanisms or separate segmentation models and achieves state-of-the-art results when compared to end-to-end models. A particular innovation of our approach in this context is that we use both character- and word-level information from the question for both entity and predicate prediction. Character-level modelling of questions (or entity mentions) and entities has been shown advantageous for this task because of its out-of-vocabulary (OOV) word handling abilities [12, 26]. However, the downside of purely character-level models is their inability to exploit word-level semantics (as captured in word embeddings [17]), which are useful for modelling questions, predicates and types. Our model combines the OOV-related advantages of character-level models and the rich semantics of word-level models. In this respect, our work is close to that of Yin et al. [26], with the difference that our approach employs a single question representation network and is less specific for simple questions since it does not split the question into mention-pattern pairs.

In summary, our main contributions are as follows:

- a simple and portable neural match-based model for answering simple questions, that
  (a) merges word- and character-level representations of words modelling of the question to exploit the advantages of both.
  (b) learns to generate knowledge base-independent representations of entities and relations that are built only from textual information associated with the entities or relations.
- a custom negative sampling procedure.
- an extensive evaluation of our method and a detailed root cause analysis for 250 questions.

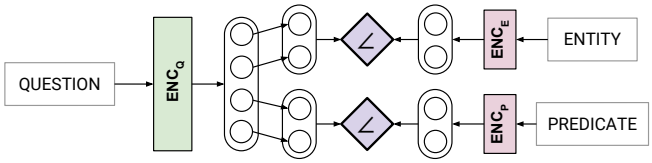

Figure 1: Visualization of the whole model. The question is encoded using the $\texttt{ENC}_\texttt{Q}$ network, producing a question representation vector. Entity and predicate representations are produced by $\texttt{ENC}_\texttt{E}$ resp. $\texttt{ENC}_\texttt{P}$. Question-entity and question-predicate pairs are scored using a cosine similarity between their representing vectors.

The rest paper is structured as follows: We present our approach in Section 2. We evaluate our implementation in Section 3 and provide a detailed analysis and discussion in Section 4. We survey related work in Section 5 and conclude in Section 6 with an outlook on future work.

2. APPROACH

The problem of single relation question answering could be divided into the following sub-tasks:

1. *segmentation* of q, i.e. finding the sub-sequences of words $\mathcal{M}_s$ and $\mathcal{M}_p \subset \{w_1, \ldots, w_T\}$, which refer to a subject and a predicate respectively,
2. *mapping* the chosen sub-sequences to an entity resp. a relation in the graph, i.e. finding s_g and p_g to which the sub-sequences $\mathcal{M}_s$ and $\mathcal{M}_p$ are referring to. Mapping to entities is known as *entity linking* whereas mapping to relations is referred to *relation prediction*.

Instead of explicitly finding solutions for the different subtasks separately, our approach considers them together without explicitly looking for a solution to either and instead directly generates a solution for the full problem. We do not employ any explicit relation lexicons (e.g. DBpedia Lemon [24]) or other external information (such as OpenIE extractions), instead letting the network learn the lexicon from examples.

Our approach relies on *recurrent neural networks* (RNN) since we are dealing with sequential data. RNNs take into account the order of words, which is important since the order of words significantly contributes to the meaning of a sentence or phrase.

The model is further elaborated in the next section (Section 2.1), followed by a description of the prediction process (Section 2.2) and the training process (Section 2.3).

2.1 Model description

From a broad perspective, our model (see Figure 1) is a matching function: given a question q and sets of candidate subject entities and relations, $\mathcal{C}_s = \{s_1, \ldots, s_n\}$ and $\mathcal{C}_p = \{p_1, \ldots, p_m\}$ respectively, it returns the subject and predicate that matches the question best. To do so, it

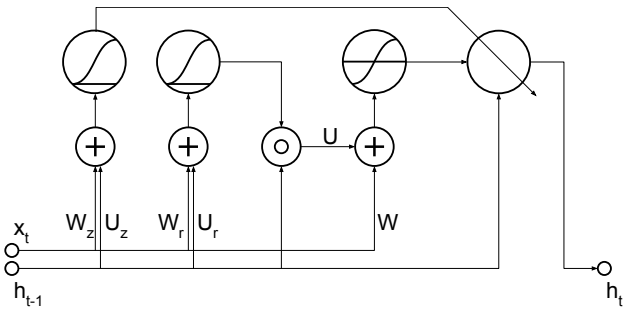

Figure 2: Gated Recurrent Unit (GRU)

(1) maps a question q to vector representation $r_q = (r_q^s, r_q^r)^T$, where r_q^s and r_q^r are subject and the relation sprecific encodings of the question respectively,
(2) maps each candidate subject $s_i \in \mathcal{C}_s$ to a vector representation r_{s_i},
(3) maps each candidate predicate $p_j \in \mathcal{C}_p$ to a vector representation r_{p_j},
(4) and computes scores $S_s(q, s_i)$ and $S_p(q, p_j)$ for each pair $r_q^s, r_{s_i}, i = 1, \ldots, n$, and $r_q^p, r_{p_j}, j = 1, \ldots, m$.

Based on these scores the final prediction is $(\hat{s}, \hat{p})$, with

$$\hat{s} = \underset{s_i \in \mathcal{C}_s}{\operatorname{argmax}} S_s(q, s_i) \ , \tag{1}$$

$$\hat{p} = \underset{p_j \in \mathcal{C}_p}{\operatorname{argmax}} S_p(q, p_j) \ . \tag{2}$$

Steps (1)-(3) heavily rely on RNNs with Gated Recurrent Units (GRUs) [6], which therefore are described first. In the subsequent subsections after that, the four parts of our model are described in detail.

Preliminary: Gated Recurrent Unit

The RNNs which are part of our model use GRUs [6, 7] for calculating the hidden states of the network which are described in the following.

The current hidden state $\boldsymbol{h}_t$ at time step t of the RNN (which is also its output at time t) is computed by interpolating between the state $\boldsymbol{h}_{t-1}$ at previous time step and the candidate state $\hat{\boldsymbol{h}}_t$

$$\boldsymbol{h}_t = (\mathbf{1} - \boldsymbol{z}_t) \cdot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \cdot \hat{\boldsymbol{h}}_t \ . \tag{3}$$

with $\boldsymbol{z}_t$ the update vector and $\cdot$ the element-wise vector product.

The *update gate* z_t for the interpolation is computed using the current input x_t and the previous state h_{t-1}

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z \boldsymbol{x}_t + \boldsymbol{U}_z \boldsymbol{h}_{t-1}) \ , \tag{4}$$

where $\boldsymbol{W}_z$ and $\boldsymbol{U}_z$ are parameter matrices to be learned during training and σ is the sigmoid activation function $\sigma(x) = 1/(1 + e^{-x})$ applied element-wise to the vector entries.

The current candidate state $\hat{\boldsymbol{h}}_t$ is computed based on the current input $\boldsymbol{x}_t$ and previous state $\boldsymbol{h}_{t-1}$

$$\hat{\boldsymbol{h}}_t = \mathsf{tanh}(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}(\boldsymbol{r}_t \cdot \boldsymbol{h}_{t-1})) \ , \tag{5}$$

where $\boldsymbol{W}$ and $\boldsymbol{U}$ are parameter matrices, tanh is the hyperbolic tangent activation function and $\boldsymbol{r}_t$ is the value of the *reset gate*, computed as follows

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r \boldsymbol{x}_t + \boldsymbol{U}_r \boldsymbol{h}_{t-1}) \ , \tag{6}$$

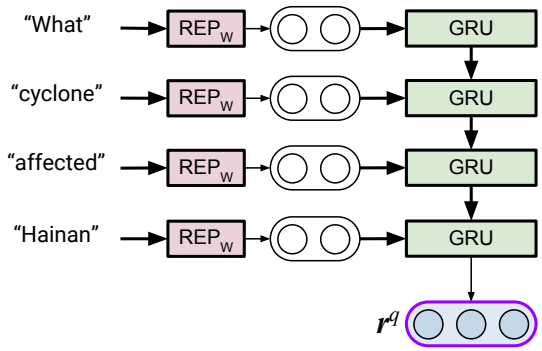

Figure 3: Question encoding network $\texttt{ENC}_\texttt{Q}$. Each word is represented by a vector using $\texttt{REP}_\texttt{W}$. The sequence of word vectors is encoded using a GRU.

with parameter matrices $\boldsymbol{W}_r$ and $\boldsymbol{U}_r$. A schematic representation of the GRU can be found in Figure 2.

The advantage of using gated units such as GRU or long short-term memory (LSTM) [14] is their ability to process longer sequences, which arises from their additive manipulation of the state vector and explicit filtering using gates. In the case of the GRU, the reset gate $\boldsymbol{r}_t$ determines which parts of the previous state $\boldsymbol{h}_{t-1}$ are "ignored" in the computation of the candidate state and the update gate $\boldsymbol{z}_t$ determines how much of the previous state is "leaked" into the current state $\boldsymbol{h}_t$. The update gate could decide to forget the previous state altogether or to simply copy the previous state and ignore the current input. Both gates are parameterized (and thus trainable) and their values depend on both the input $\boldsymbol{x}_t$ and the previous state.

2.1.1 Representing the question

The mapping of a question $q = \{w_1, \ldots, w_T\}$ to its subject and predicate related vector representations $\boldsymbol{r}_q^s$ and $\boldsymbol{r}_q^r$, respectively, is done using a single-layered unidirectional GRU based encoder network. We call this part of the model the *question encoder* $\texttt{ENC}_\texttt{Q}$

$$\boldsymbol{r}_q = \begin{bmatrix} \boldsymbol{r}_q^s \\ \boldsymbol{r}_q^p \\ \boldsymbol{r}_q^r \end{bmatrix} = \texttt{ENC}_\texttt{Q}(\{w_1, \ldots, w_T\}) \ . \tag{7}$$

The question encoder $\texttt{ENC}_\texttt{Q}$ first uses the word representation function $\texttt{REP}_\texttt{W}(w_t)$ to generate vector representations for all words $w_t, t = 1, \ldots, T$ (as described in the next paragraph), which are subsequently fed to the RNN until all words have been seen. Starting with the initial hidden state $\boldsymbol{h}_0$, the GRU of the question encoder RNN iteratively updates its hidden state $\boldsymbol{h}_t$ after processing each word according to Equations (3) to (6), where the word representation vector $\texttt{REP}_\texttt{W}(w_t)$ is fed as input to the GRU (i.e. $\boldsymbol{x}_t = \texttt{REP}_\texttt{W}(w_t)$). The final hidden state $\boldsymbol{h}_T$ (produced after processing the last word represented by $\texttt{REP}_\texttt{W}(w_T)$) is returned by $\texttt{ENC}_\texttt{Q}$ as the representation of question q.

The question encoder is visualized in Figure 3.

Word representation.

In the following we descibe how we generate the vector representations of the words $w_1, \ldots, w_T$. In order to exploit both word- and character-level information of the question,

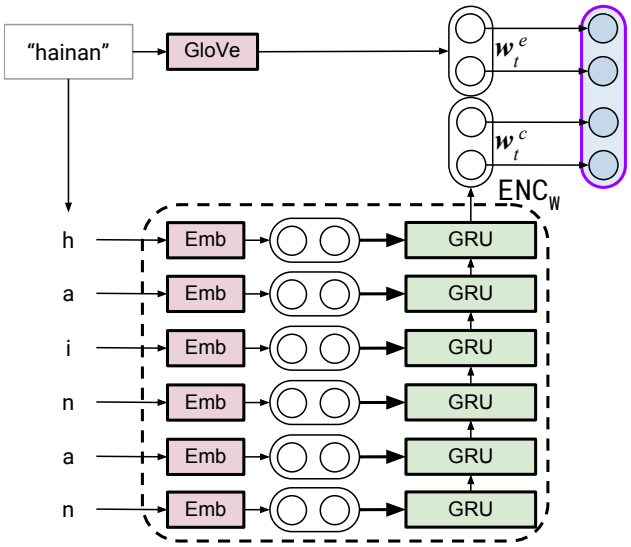

Figure 4: Word representation network $\texttt{REP}_W$ with example. The word is considered as a sequence of character and fed to $\texttt{ENC}_W$, where each character is embedded and the sequence of character vectors is encoded using a GRU to produce a character-level encoding of that word. This is concatenated to the word embedding (we use GloVe) to produce the complete word representation.

we use a "nested" word- and character-level aproach concatenating the pre-trained embedding of a word with an RNN-based encoding on character level.

As word embeddings, we use *GloVe* [17] vectors provided on the GloVe website[5]. Such pre-trained word embeddings implicitly incorporate word semantics inferred from a large text corpus based on the distributional semantics hypothesis [20]. This hypothesis can be phrased as "words with similar meanings occur in similar contexts", which in our case translates to similar vectors for words with similar meanings.

Using those pretrained word embeddings allows us to better handle synonyms and find better matches between words in the question and subject labels or predicate URI's. In addition, during testing, it allows to handle words that have not been seen during training.

The word embedding of w_t resulting in the d_{w^e}-dimensional vector representation $\boldsymbol{w}_t^e$ can be formally described as follows

$$\boldsymbol{w}_t^e = \boldsymbol{W}_g^\top \boldsymbol{v}_t \ , \tag{8}$$

where $\boldsymbol{W}_g \in \mathbb{R}^{|V_g| \times d_{w^e}}$ is the provided pretrained word embedding matrix for a vocabulary of size $|V_g|$ (GloVe covers 400k words), and $\boldsymbol{v}_t$ is the one-hot vector representation of w_t. Since the coverage of word embeddings is limited, many words appearing in the questions (especially those that are part of a reference to a particular subject entity, e.g. the last name "*golfis*" in the question "*what city was alex golfis born in*") are not contained in the vocabulary of the pre-trained embeddings. In such cases (20.8% and 14.5% of unique words in the train resp. test questions), we set the word embedding to the zero vector.

[5] `http://nlp.stanford.edu/projects/glove/`

The encoding of the word $w_t = \{w_t^1, \dots w_t^K\}$ on character-level is based on a RNN encoder:

$$w_t^c = \texttt{ENC}_W(\{w_t^1, \dots w_t^K\}) \ , \tag{9}$$

Inside $\texttt{ENC}_W$, the characters $w_t^k, k = 1, \dots, K$ are first embedded by

$$\boldsymbol{c}_t^k = \boldsymbol{W}_c^\top \boldsymbol{v}_t^k \ , \tag{10}$$

with character embedding matrix $\boldsymbol{W}_c \in \mathbb{R}^{|V_c| \times d_{c^e}}$ (for $|V_c|$ characters) learned during training, and $\boldsymbol{v}_t^k$ the one-hot vector representation of the character w_t^k. Then we feed the sequence of character vectors $\{\boldsymbol{c}_t^1, \dots, \boldsymbol{c}_t^K\}$ to a single-layered unidirectional GRU network and take its final state as the character-level word encoding $\boldsymbol{w}_c$.

The added character-level encoding provides information necessary for matching question words with entity labels, in addition to providing distinguishable representations for OOV words. This approach is similar to the *char2word* model proposed by Ling et al. [16] with the difference that we use a unidirectional GRU network.

Finally, to get the vector representation of a word w_t, the word embedding $\boldsymbol{w}_t^e$ and character-level encoding $\boldsymbol{w}_t^c$ are concatenated:

$$\texttt{REP}_W(w_t) = \begin{bmatrix} \boldsymbol{w}_t^e \\ \boldsymbol{w}_t^c \end{bmatrix} \ . \tag{11}$$

The whole word representation network is illustrated in Figure 4.

2.1.2 *Representing the subject*

We use both the entity label and the type label of the entities in the knowledge graph to build subject representations. Entity labels are encoded on the level of characters because of the high prevalence of OOV words and their importance for entity labels. On the other hand, OOV words are rather rare in type labels and thus type labels are encoded on word level.

For Freebase entities, we extract entity labels using the `type.object.name` properties of entities and type labels of entities by first getting the `common.topic.notable_types` of entities[6] and then taking the `type.object.name` value of the types.

The character-level entity label encoding $\boldsymbol{s}^l$ and word-level type label encoding $\boldsymbol{s}^t$ are concatenated to produce the subject representation vector

$$\boldsymbol{s}^l = \texttt{ENC}_{\texttt{SL}}(\{c_s^1, c_s^2, \dots\}) \ , \tag{12a}$$

$$\boldsymbol{s}^t = \texttt{ENC}_{\texttt{ST}}(\{w_t^1, w_t^2, \dots\}) \ , \tag{12b}$$

$$\boldsymbol{r}_s = \begin{bmatrix} \boldsymbol{s}^l \\ \boldsymbol{s}^t \end{bmatrix} \ , \tag{12c}$$

where $\texttt{ENC}_{\texttt{SL}}$ is the character-level encoder of the subject entity label and $\texttt{ENC}_{\texttt{ST}}$ is the word-level type label encoder. The label characters and type label words, respectively, are first embedded (following Equation 10 and Equation 8, respecrively) and the embedding vectors are fed to the respective encoding RNNs. Both $\texttt{ENC}_{\texttt{SL}}$ and $\texttt{ENC}_{\texttt{ST}}$ correspond to single-layer unidirectional GRU-based RNNs and take their final hidden state as the entity label encoding $\boldsymbol{s}^l$ and type

[6] The notable types property provide the single, most characteristic type for that entity. However, using all types of the entity (e.g. concatenating their labels) could be interesting as well, which we leave for future work.

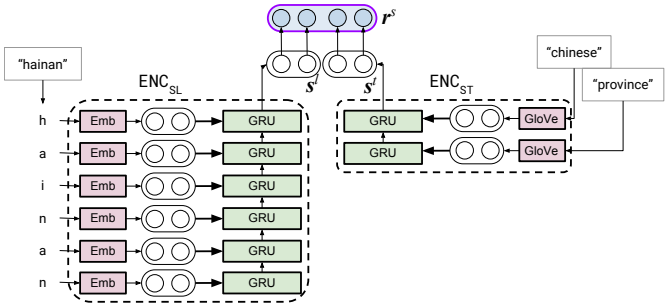

Figure 5: Entity encoder with example. The entity label is encoded on character level ($\texttt{ENC}_{\texttt{SL}}$) and the subject type label is encoded on word level ($\texttt{ENC}_{\texttt{ST}}$). The two are concatenated to produce the subject vector.

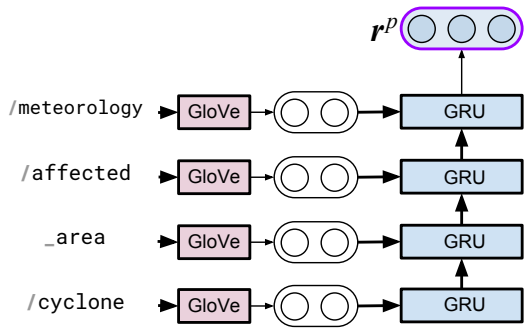

Figure 6: Predicate encoder network $\texttt{ENC}_{\texttt{R}}$ with example. The predicate URI is split into words and encoded on word level using GloVe embedding.

label encoding $\boldsymbol{s}^t$, respectively. The subject representation network is visualized in Figure 5.

This method of building subject representations is similar to the method proposed by Sun et al. [21] who focus on entity linking and CNNs for word-level entity name encoding (instead of using RNNs for character level based encodings like our model, which allows to handle OOV words) and word-level entity type name encoding, followed by an additional layer that merges the two (where we simply concatenate both representations).

2.1.3 Representing the predicate

We use the predicate URI's provided by the KG to build latent vector representations of the predicates. The predicate URI is first split into words $w_p^1, w_p^2, \dots$, each word is embedded (as described by Equation 8), and then the word embeddings are fed into a single-layer word-level GRU-based encoder $\texttt{ENC}_{\texttt{R}}$ that takes the final state of its RNN as the representation of the predicate URI, that is

$$\boldsymbol{r}_p = \texttt{ENC}_{\texttt{R}}(\{w_p^1, w_p^2, \dots\}) \ . \tag{13}$$

The relation encoding network is visualized in Figure 6.

2.1.4 Matching scores

Given the question encoding vector $\boldsymbol{r}_q = (\boldsymbol{r}_q^s, \boldsymbol{r}_q^p)$, the latent vector representation $\boldsymbol{r}_p$ of the relation, and the latent representation $\boldsymbol{r}_s$ of the subject entity, we compute two matching scores: one between the question and subject entity and one between the question and predicate, as follows:

$$S_s(q,s) = \mathsf{cos}(\boldsymbol{r}_q^s, \boldsymbol{r}_s) \tag{14a}$$
$$S_p(q,p) = \mathsf{cos}(\boldsymbol{r}_q^p, \boldsymbol{r}_p) \ , \tag{14b}$$

where cos is the cosine similarity given by $\mathsf{cos}(\boldsymbol{a}, \boldsymbol{b}) = \dfrac{(\boldsymbol{a} \cdot \boldsymbol{b})}{|\boldsymbol{a}||\boldsymbol{b}|}$.

2.2 Prediction

With the model described in the previous section we are now able to compute scores for question-subject and question-predicate pairs (Equation (14)). Using these scoring functions, we can solve the task of finding the right subject-predicate pair (s_g, p_g) (i.e. , retrieving triples $(s_g, p_g, o_i) \in \mathcal{G}$ such that the set of objects in these triples constitutes the answer to question q) by picking the best scoring subject entity and predicate given a question according to Equations (1) and (2), respectively. However, computing scores for all entities and predicates in the KG would be very costly and introduces a lot of noise. So we only consider small subsets of entities and predicates which are most likely to be the correct ones for a given question. We refer to this sets as *candidate subjects* $\mathcal{C}_s$ and *candidate predicates* $\mathcal{C}_p$, respectively.

We explore two options for generating predictions. The first proceeds in the following four steps:

1. generate subject candidates $\mathcal{C}_s$ (as described in Section 2.2.1).
2. score each entity in $\mathcal{C}_s$ and take top-scored one as prediction (Equations (1) and (14a))
3. generate predicate candidates $\mathcal{C}_p$ based on the *top subject entity* (see Section 2.2.2)
4. score each predicate in $\mathcal{C}_p$ and take top-scored one as prediction (Equations (2) and (14b))

The second option allows to correct wrong subject predictions by an added pruning step (which we refer to as *subject pruning* in the following):

1. generate subject candidates $\mathcal{C}_s$ (see Section 2.2.1).
2. score each entity in $\mathcal{C}_s$ (Equation (14a))
3. generate predicate candidates $\mathcal{C}_p$ based on the *top subject label* (see Section 2.2.2)
4. score each predicate in $\mathcal{C}_p$ and take top-scored one (Equations (2) and (14b))
5. prune $\mathcal{C}_s$ based on the predicted predicate as described in Section 2.2.1 and take the top-scoring entity from the filtered $\mathcal{C}_s$ as subject prediction

We now describe how the *candidate subjects* and *candidate predicates* are constructed and pruned in detail.

2.2.1 Generation of candidate subjects

First, we collect the lowercased English entity labels from the KG (for Freebase, we use the `type.object.name` property). To generate the set of candidate subjects $\mathcal{C}_s$, all word n-grams of size 1 to L contained in question q are retrieved, filtered and used for searching matching entities using the collected labels according to the following rules:

- An entity whose label exactly matches a n-gram is added to the set of candidates $\mathcal{C}_s$.
- An n-gram that is fully contained inside a bigger n-gram that matched at least one entity label is discarded, unless the bigger n-gram starts with one of

the following words: "*the*", "*a*", "*an*", "*of*", "*on*", "*at*", "*by*" .

- In case an exact match is not found for an n-gram, we search for entities whose label has an edit distance with the n-gram of at most 1 and add them to $\mathcal{C}_s$.
- If there are several entities matching a n-gram, the entities are ranked by the number of triples in the KG in which they play the role of the subject. Only the m entities with highest rank are added to the candidate set (in our experiments $m \in \{5, 10, 400\}$.)

Note, that the candidate generation can be efficiently implemented using information retrieval libraries such as Apache Lucene.

Pruning subjects based on predicted relation.

When generating the predicate candidates based on the *top subject label*, we perform additional pruning of $\mathcal{C}_s$: We remove all entities that *do not* appear as subject in *any* of the triples in $\mathcal{G}$ having the predicted relation as predicate.

2.2.2 Generation of candidate predicates

For generating the candidate set of predicates $\mathcal{C}_p$ we explore two approaches, which bose depend on $\mathcal{C}_s$ and the highest scored subject $\hat{s} \in \mathcal{C}_s$

- *Top subject entity based*: only predicates occurring in triples from $\mathcal{G}$ which have $\hat{s}$ as subject are added to $\mathcal{C}_p$. This way of generating $\mathcal{C}_p$ is highly dependent on subject prediction. If the wrong subject has been chosen, it is highly probable the right predicate is not in $\mathcal{C}_p$.
- *Top subject label based*: all subject candidates from $\mathcal{C}_s$ whose label is the same as that of the top-ranking candidate $\hat{s}$ are considered and all predicates that occur in triples having one of those entities as subject are added to $\mathcal{C}_p$.

2.3 Training

The model described above is trained with a ranking training approach, which drives the model to output a high score for question-entity and question-predicate pairs contained in the training set while producing a lower score for implausible pairs. The loss function minimized during training is given by

$$- \sum_{(q,s^+,p^+)\in\mathcal{D}} \Big(\max\left(0, S_s(q,s^-) - S_s(q,s^+) + \gamma\right) + \max\left(0, S_p(q,p^-) - S_p(q,p^+) + \gamma\right) \Big) \quad . \tag{15a}$$

That is, both the pair of question and true subject (q, s^+) and the pair of question and true predicate (q, p^+) are forced to have a score of at least margin γ higher than the score of corrupted pairs (q, s^-) and (q, p^-) consisting of the same question sentence and a false subject or relation, respectively. The false subjects s^- and relations p^- are randomly sampled at each epoch as described in the next paragraph. We also experimented with a binary softmax loss, which did not result in improvements and was slightly slower during training.

2.3.1 Negative sampling

At each iteration, a corrupted subject-predicate pair (s^-, p^-) is generated based on the true subject-predicate pair (s^+, p^+) from the training set.

In our experiments, we observed that the way we generate negative samples has a significant influence on the results. We developed the following negative sample generation methodology, which tries to expose the model to conditions close to those it will encounter during prediction.

Subject corruption.

To corrupt (s^+, p^+), a false subject s^- can be sampled from the space of all entities in the KG $\mathcal{G}$. However, we opt for a "close" corruption sampling scenario that forces the model to learn to distinguish the correct entity from more plausible wrong entities that are collected using the same candidate generation procedure as the one used for prediction.

In this "close" corruption scenario, the corrupted entity s^- for the correct entity s^+ is sampled from the set $\mathcal{X}^-_{s^+}$, which is constructed as

$$\mathcal{X}^-_{s^+} = \mathcal{C}^i_s \setminus \{s^+\} \ , \tag{16}$$

where C^i_s is the subject candidate set for i-th question q_i in the training set. This way, s^+ can be replaced by any entity that is a subject candidate for the same question.

At each training epoch, we randomly sample from this set $\mathcal{X}^-_{s^+}$ of false close candidates unless its size is smaller than 5, in which case we sample randomly from the whole set of entities in $\mathcal{G}$.

Predicate corruption.

For predicate corruption, we first take the set $\mathcal{X}^-_{p^+}$ of predicates that occur in triples in $\mathcal{G}$ having s^+ as subject. With probability $P_{\mathcal{X}^-_{p^+}}$ with

$$P_{\mathcal{X}} = \mathsf{tanh}(\mathsf{log}(|\mathcal{X}| + 1)/3) \ , \tag{17}$$

(which is higher as larger $\mathcal{X}$ is) we uniformly draw a random sample from this set. Otherwise, we uniformly draw a random sample from the set $\hat{\mathcal{X}}^-_{p^+}$ of predicates whose URI's have at least one word in common with the URI of the true predicate p^+ with probability $P_{\hat{\mathcal{X}}^-_{p^+}}$ or sample uniformly from the set of all predicates in $\mathcal{G}$ with probability $1 - P_{\hat{\mathcal{X}}^-_{p^+}}$. This way, preference is given to outgoing relations of the correct subject but if the set is too small (which would result in the same few predicates being used throughout the whole training process), we draw more samples from the set of similar predicates or the set of all predicates in $\mathcal{G}$.

2.3.2 Training settings

The loss is minimized using the AdaGrad optimizer [11] with learning rate 0.1 in a mini-batch setting with batch size 100. A margin of 0.5 gave the best results in our experiments[7]. We train our model for 50 epochs[8]. We

[7]We also tried margins of 0.25 and 1.0 but did not observe improvement.

[8]We observed that after about only 15 epochs our model already reaches decent performance on the validation set. Afterwards the accuracy continued to increase slowly, starting to stagnate around 50 epochs. We did not observe overfitting when training for at least 100 epochs.

use a 400-dimensional question encoding, 200-dimensional entity and predicate representations and 100-dimensional character-level word representations. (Training with a 600-dimensional question encoding and 300-dimensional entity and predicate representations did not increase performance.) We use 100-dimensional GloVe word embeddings and 50-dimensional character embeddings. A more thorough investigation of the hyperparameter space was not possible due to the long training time and resource constraints.

2.3.3 Training data

We train on SimpleQuestions [4], a dataset consisting of over 100.000 simple questions and the corresponding Freebase triples providing the answer. We use training, validation, and test splits as provided in the dataset, containing 75.910, 10.845, and 21.687 questions, respectively. We work against the FB2M subset of Freebase, as provided with the SimpleQuestions data, which contains approximately 2 million entities and 6701 relations. The number of questions referring to relations that have not been seen during training is negligible (161 questions with 148 unique relations). On the other hand, 78.5% of the 19406 distinct subject entities in the test set are unseen during training.

2.4 Technical details

We implemented our approach[9] using Theano[10], Lasagne[11], and NLTK[12]. Training takes approximately two days on a single Titan X GPU for 50 epochs with the reported hyperparameters.

3. EVALUATION

We evaluate our method on the provided test portion of the SimpleQuestions dataset which contains $N = 21.687$ questions and the corresponding triples. For each question we follow the procedure described in Section 2.2 to find the best-scoring subject-predicate pairs, resulting in $(\hat{s}_i, \hat{p}_i), i = 1, \dots N$. Comparing those to the right subject-predicate pairs $(s_i, p_i), i = 1, \dots N$ provided by the testset the accuracy is computed as

$$\frac{\sum_{i=1}^{N} \mathbf{1}_{[(\hat{s}_i,\hat{p}_i)=(s_i,p_i)]}}{N} \quad , \tag{18}$$

where $\mathbf{1}_{[.]}$ is the indicator function (that is, $\mathbf{1}_{[a=b]} = 0$ if $a \neq b$ and $\mathbf{1}_{[a=b]} = 1$ if $a = b$). In the following subsections, we discuss our obtained results in different settings and compare to existing approaches.

3.1 Influence of candidate generation

Table 1 shows the recall of the different candidate generation settings for subjects over the test set, where recall is computed as the fraction of questions for which the set of generated subject candidates includes the right subject entity. The numbers indicate that including partial matches (matches of n-grams with edit distance 1) results in a recall increase of up to 2%.

Table 2 shows test accuracy for different candidate generation and pruning settings. Test accuracy in the FB2M setting is reported for different numbers of subject entity candidates and with or without subject pruning after relation ranking. As can be seen in Table 2, pruning subjects after relation prediction provides an improvement of at most 1% for our model by enforcing consistency with the set of available triples. A larger number of candidates translates to a higher accuracy due to higher candidate generation recall (see Table 1).

To investigate how much the candidate generation contributes to the results we computed the accuracy resulting from randomly picking an entity from the list of subject candidate entities and subsequently randomly picking a relation predicate from the list of outgoing relations of the chosen subject entity. The results are shown in Table 3. The poor performance verifies that the task of finding the right subject predicate pair can not be solved by n-gram based search alone.

# Candidates	Partial matches	Recall %
5	no	84.8
10	no	87.3
400	no	91.8
5	yes	85.4
10	yes	88.4
400	yes	93.7

Table 1: Candidate generation recall for subjects over the test set. Recall is defined as the percentage of test questions for which the expected entity is retrieved during candidate generation.

# Candidates	Subject pruning	Accuracy %
5	no	69.3
10	no	70.1
400	no	70.2
5	yes	70.0
10	yes	70.9
400	yes	71.2

Table 2: Test accuracy for different candidate generation settings.

3.2 Comparison

We compare our results with the four recent state-of-the-art works that developed QA models for the SimpleQuestions dataset. These works include the *Memory Network* approach of Bordes et al. [4], the attentive CNN based approach by Yin et al. [26], the character-level attention-enhanced encoder-decoder approach by Golub and He [12], and the word-level RNN-based approach by Dai et al. [8] (cf. Section 5 for a more detailed discussion of these approaches).

We compare the results obtained by the state-of-the-art models when evaluated in an end-to-end setting to the results we obtain with our model (presented in Table 4). The second part of Table 4 shows the performance of two sys-

	5	10	400
Accuracy (%)	5.23	4.72	4.03

Table 3: Test accuracy of random choice model, for 5, 10 resp. 400 subject entity candidates.

[9] http://github.com/WDAqua/teafacto
[10] http://deeplearning.net/software/theano/
[11] https://github.com/Lasagne/Lasagne
[12] http://www.nltk.org/

Approach	Setting	Test Accuracy %
Bordes et al. [4]	end-to-end	62.7
Yin et al. [26]	end-to-end	68.3
Dai et al. [8]	end-to-end	62.6*
Golub and He [12]	end-to-end	70.9
Our approach	end-to-end	71.2
Dai et al. [8]	active linking	75.7*
Yin et al. [26]	focused pruning	76.4

Table 4: Comparison with state-of-the-art systems. When marked with an asterisk (*), accuracy in the FB5M test setting is given, otherwise FB2M test setting results are shown.

Cause of error	Counts
Missing	26
Indistinguishability	9
Hard ambiguity	11
Soft ambiguity	1
Wrong subject mention	4
Wrong predicate	28
Total	**69**

Table 5: Counts of sources of errors that our model makes on 250 test examples. The numbers have been obtained after prediction with 10 candidates per n-gram.

tems that train and use a separate question segmentation model that finds the entity mention, which improve their results significantly. However, since the segmentation model requires separate training with different data, we deem direct comparison unfair.

As shown in Table 4, our approach achieves competitive results in the end-to-end setting. Our approach beats the baseline results of Bordes et al. [4] by a high margin, outperforms the CNN-based approach of Yin et al. [26] and manages to match and surpass the attention-enhanced encoder-decoder approach of Golub and He [12]. However, it is outmatched by approaches that use a separately trained segmentation model [26, 8]. Including a similar segmentation model could improve our results as well by resolving the segmentation errors (see next section) and reducing noise in predicate relation patterns.

4. ERROR ANALYSIS

We perform error analysis on our single-layered unidirectional question encoder model with 10 subject entity candidates per n-gram to gain insight into the kind of errors our approach produces.

We distinguish between the following types of errors:

- *Missing candidates:* The correct entity is missing in the candidate set. For example, in the question "*which group recorded tokyo?*", "*tokyo*" has hundreds of possible candidates and by restricting the subject candidate set to a small number (e.g. 10), there is a high chance the right interpretation of "*tokyo*" is not among the considered candidates.
- *Indistinguishability:* The predicted entity is indistinguishable from the correct entity (i.e., has the same label and the same type). For example, the question "*what song was on the recording indiana?*" expects an entity with label "*indiana*" and type label "*musical recording*". However, there exist multiple entities in the KG conforming to these criteria and they all receive the same score.
- *Hard ambiguity:* The top-scored entity has the same label but different type from the lower-scored correct entity and the question does not provide clear indications which entity is expected. For example, for the question "*where was eddie smith born?*", the two top-ranked candidates are *Eddie Smith, baseball player* and *Eddie Smith, film actor*.
- *Soft ambiguity:* The top-scored and correct entities have the same label and different types, as with *Hard ambiguity*, however, the question provides information for correct disambiguation.
- *Wrong subject mention:* The label of the top-scored entity is different from the label of the lower-scored correct entity, indicating that the question encoder made a mistake by (implicitly) taking the wrong subject mention span. For example, for the question "*who directed the show dancin' homer?*", an entity with label "*The Show*" and type label "*film*" is the top-scored entity, with a score slightly higher than the correct subject entity with label "*Dancin' Homer*" and type label "*tv episode*".
- *Wrong predicate:* The subject is predicted correctly, but a wrong relation is chosen.

We manually inspect 250 questions from the test set and present the findings in Table 5. Among errors resulting in wrong subject prediction, missing candidates are the most frequent category when only the top-10 candidates per n-gram are used. This is due to the fact that in some cases, there are many more than 10 suitable candidates for some n-gram and the true subject entity was ranked lower than the cutoff (10) during candidate generation (we rank by triple count).

We believe the examples in error categories *Indistinguishability* and *Hard ambiguity* to be unresolvable with the information currently used in our approach (entity labels and entity type labels for matching and using provided knowledge for subject pruning) since there isn't enough information even for humans to pick the right answer. When using larger candidate sets (e.g. 400), we observed that many error cases from the *Missing candidates* category migrated to the *Hard ambiguity* or *Indistinguishability* categories . Other ambiguity errors and segmentation errors, however, offer room for the improvement of our approach.

We did not perform a thorough error analysis of relation prediction, but during inspection, we encountered cases where picking the right relation is very challenging. One such example is "*who produced the film woodstock villa?*", where our approach ranked `/film/film/produced_by` at the top and `/film/film/executive_produced_by`, the correct predicate, only second.

5. RELATED WORK

Our work most closely relates to recently proposed match-based neural network approaches for Simple QA over KG. The Memory Network based approach of Bordes et al. [4]

employs a bag-of-n-grams embedding approach. They also investigate the use of additional data (i.e. WebQuestions and ReVerb) for training.

Golub and He [12] propose a character-level approach based on the attention-enhanced encoder-decoder architecture recently developed for machine translation [1], where attention was introduced to better handle longer sequences. This architecture was shown to be suitable for different tasks, including for example constituency parsing [25] and semantic parsing [9]. The model of Golub and He consists of a character-level RNN-based question encoder and an attention-enhanced RNN decoder coupled with two separate character-level CNN-based entity label and predicate URI encoders. Given that their model works purely on character-level, which results in much longer input sequences, the influence of an attention mechanism should be more pronounced than with word-level models. From the output size perspective, it is reasonable to assume that compared to machine translation an attention mechanism is of lesser importance in Simple QA where only two predictions need to be made per input sequence. We developed a model without an attention mechanism to reduce complexity and improve efficiency. Instead, we focused on better question representation, where we operate on word level but also integrate character-level information for each word. However, when extending our model to more complex questions in future work, we believe the addition of an attention mechanism could be advantageous.

Yin et al. [26] employ CNNs and propose an attentive pooling approach to improve the model. First, the question is split into entity mention and relation pattern. Then, the entity mention is encoded using a character-level CNN and matched against a character-level encoded subject label; the relation pattern is encoded using a separate word-level CNN with attentive max-pooling and matched against a word-level encoded predicate URI. Instead of using an attention mechanism to implicitly perform segmentation, in this approach an attention mechanism is used to obtain better matches for predicates. Our work is close to this approach in that it also encodes subject labels on character level. However, our wokk is different in that we also add type label information in order to improve subject disambiguation and employ a hierarchical word- and character-level question encoder.

Dai et al [8] investigate a word-level RNN-based approach. In contrast to our work, they encode the question on word-level only and train/use Freebase-specific predicate and entity representation (such as pretrained TransE embeddings and type vectors). In contrast, our work (as well as that of Golub and He [12] and Yin et al. [26]) fully relies on textual information provided by the KG about the entities and predicates. Thus, the approach of Dai et al. is highly specific for Freebase and can less easily be transferred to other KGs such as DBpedia Live for example.

Both Dai et al. [8] and Yin et al. [26] improved the performance of their approaches using a BiLSTM-CRF tagging model that is separately trained to label parts of the question as entity mention or context (relation pattern).

A different line of research for QA over KG investigates semantic parsing approaches to translate questions into formal queries that can be executed against a knowledge base [2, 3, 18]. Most approaches evaluated on QALD assume a semantic parsing approach that translates questions into SPARQL queries [23, 22, 13, 10]. Our approach avoids a semantic parsing approach requiring complex NLP pipelines, in particular mitigating the error propagation problem in those.

6. CONCLUSION

In this article, we presented an end-to-end, neural network based approach for answering simple questions over large-scale knowledge graphs, leveraging a hierarchical word- and character-level question encoder. We also investigate the use of type information for better subject disambiguation. While keeping the model simple by refraining from using attention mechanisms or separate segmentation models, the proposed approach achieves competitive results when compared to other end-to-end approaches. In comparison to other character-level approaches for this task [12, 8], we explore hierarchical character-word level representation networks, which allows to exploit richer word-level semantics as captured in pretrained word embeddings.

We found that the choice of negative samples has a big impact on performance and experimented with a negative sampling generation approach that exposes the model to conditions equivalent to those under which it will be used during prediction.

As our model does not learn KG-specific symbol representations, we believe it to be easily portable to other knowledge graphs and suitable for KG's that change over time (such as DBpedia Live). However, we leave an evaluation of its portability for future work.

In future work, the investigation of implicit [12] or explicit [26, 8] segmentation modelling should improve subject prediction by better candidate pruning and reduce the noise for better learning of predicate patterns. Exploitation of external lexical resources could also prove advantageous as it should help to bridge the lexical gap.

Acknowledgments

This work is supported in part by the European Union under the Horizon 2020 Framework Program for the project WDAqua (GA 642795). We would like to thank Maria-Esther Vidal and Joanna Lytra for their useful comments and suggestions.

7. REFERENCES

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544. ACL, 2013.

[3] J. Berant and P. Liang. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[4] A. Bordes, N. Usunier, S. Chopra, and J. Weston. Large-scale simple question answering with memory networks. *CoRR*, abs/1506.02075, 2015.

[5] A. Both, D. Diefenbach, K. Singh, S. Shekarpour, D. Cherix, and C. Lange. Qanary - a methodology for vocabulary-driven open question answering systems. In Sack et al. [19], pages 625–641.

[6] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[7] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[8] Z. Dai, L. Li, and W. Xu. Cfo: Conditional focused neural question answering with large-scale knowledge bases. *arXiv preprint arXiv:1606.01994*, 2016.

[9] L. Dong and M. Lapata. Language to logical form with neural attention. In *ACL (1)*. The Association for Computer Linguistics, 2016.

[10] M. Dubey, S. Dasgupta, A. Sharma, K. Höffner, and J. Lehmann. Asknow: A framework for natural language query formalization in sparql. In Sack et al. [19], pages 300–316.

[11] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[12] D. Golub and X. He. Character-level question answering with attention. *EMNLP*, 2016.

[13] S. He, Y. Zhang, K. Liu, and J. Zhao. Casia@ v2: A mln-based question answering system over linked data. In *CLEF (Working Notes)*, pages 1249–1259, 2014.

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

[15] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.

[16] W. Ling, T. Luís, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and I. Trancoso. Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*, 2015.

[17] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[18] S. Reddy, M. Lapata, and M. Steedman. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392, 2014.

[19] H. Sack, E. Blomqvist, M. d'Aquin, C. Ghidini, S. P. Ponzetto, and C. Lange, editors. *The Semantic Web. Latest Advances and New Domains - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2, 2016, Proceedings*, volume 9678 of *Lecture Notes in Computer Science*. Springer, 2016.

[20] M. Sahlgren. The distributional hypothesis. *Italian Journal of Linguistics*, 20(1):33–54, 2008.

[21] Y. Sun, L. Lin, D. Tang, N. Yang, Z. Ji, and X. Wang. Modeling mention, context and entity with neural networks for entity disambiguation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1333–1339, 2015.

[22] C. Unger, L. BÃijhmann, J. Lehmann, A.-C. N. Ngomo, D. Gerber, and P. Cimiano. Template-based question answering over rdf data. In A. Mille, F. L. Gandon, J. Misselis, M. Rabinovich, and S. Staab, editors, *WWW*, pages 639–648. ACM, 2012.

[23] C. Unger and P. Cimiano. Pythia: Compositional meaning construction for ontology-based question answering on the semantic web. In *International Conference on Application of Natural Language to Information Systems*, pages 153–160. Springer, 2011.

[24] C. Unger, J. P. McCrae, S. Walter, S. Winter, and P. Cimiano. A lemon lexicon for dbpedia. In S. Hellmann, A. Filipowska, C. BarriÃÍre, P. N. Mendes, and D. Kontokostas, editors, *NLP-DBPEDIA@ISWC*, volume 1064 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[25] O. Vinyals, Ł. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781, 2015.

[26] W. Yin, M. Yu, B. Xiang, B. Zhou, and H. Schütze. Simple question answering by attentive convolutional neural network. In *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, 11-16 December 2016, Osaka, Japan, ACL 2016*, 2016.