Computational Linguistics and Intellectual Technologies:
Proceedings of the International Conference "Dialogue 2020"

Moscow, June 17–20, 2020

SPARQL QUERY GENERATION FOR COMPLEX QUESTION ANSWERING WITH BERT AND BILSTM-BASED MODEL

Evseev D. A. (dmitrij.euseew@yandex.ru),
Arkhipov M. Yu. (arkhipov@yahoo.com)

Neural Networks and Deep Learning Lab, Moscow Institute of Physics and Technology, Moscow, Russia

In this paper we describe question answering system for answering of complex questions over Wikidata knowledge base. Unlike simple questions, which require extraction of single fact from the knowledge base, complex questions are based on more than one triplet and need logical or comparative reasoning. The proposed question answering system translates a natural language question into a query in SPARQL language, execution of which gives an answer. The system includes the models which define the SPARQL query template corresponding to the question and then fill the slots in the template with entities, relations and numerical values. For entity detection we use BERT-based sequence labelling model. Ranking of candidate relations is performed in two steps with BiLSTM and BERT-based models. The proposed models are the first solution for LC-QUAD2.0 dataset. The system is capable of answering complex questions which involve comparative or boolean reasoning.

Key words: knowledge base, complex question answering, query generation, entity detection, relation prediction

DOI: 10.28995/2075-7182-2020-19-270-282

ГЕНЕРАЦИЯ SPARQL-ЗАПРОСОВ ДЛЯ ОТВЕТА НА СЛОЖНЫЕ ВОПРОСЫ С ПОМОЩЬЮ BERT И BILSTM

Евсеев Д. А. (dmitrij.euseew@yandex.ru),
Архипов М. Ю. (arkhipov@yahoo.com)

Лаборатория нейронных систем и глубокого обучения, Московский физико-технический институт (национальный исследовательский университет), Москва, Россия

В данной работе описывается вопросно-ответная система для ответа на сложные вопросы по базе знаний Wikidata. В отличие от простых вопросов, для ответа на которые требуется найти один факт в базе знаний, сложные вопросы требуют извлечения более 1 триплета, а также логические или сравнительные рассуждения. Предложенная система переводит вопрос на естественном языке в запрос на языке SPARQL, выполнение которого дает ответ. В состав системы входят модели, которые определяют шаблон SPARQL-запроса, соответствующего вопросу, и затем заполняют пустые места в шаблоне сущностями, отношениями и численными значениями. Для извлечения сущностей мы использовали модель маркировки последовательностей на основе BERT. Ранжирование возможных отношений для вопроса происходит в два этапа с помощью моделей на основе BiLSTM и BERT. Предложенные модели — первое решение для датасета LC-QUAD2.0. Система способна отвечать на вопросы, требующие сравнительное или логическое рассуждение.

Ключевые слова: база знаний, ответ на сложные вопросы, генерация запросов, извлечение сущностей, извлечение отношений

1. Introduction

Question answering has been an active area of research over past decades. Question answering systems can use two kinds of sources to find an answer: unstructured text corpora [11], [10], and knowledge bases (KB). KBs are an important source of information which integrates information from different sources [15]. Question answering models using KBs are compact and interpretable [16].

Knowledge base question answering (KBQA) requires matching of a subgraph with a question. If the question corresponds to a single triplet in a KB, the task is called simple question answering [1]. Complex question answering requires matching several triplets and logical, quantitative and comparative reasoning over knowledge graphs [13], [4].

One of the key approaches to complex question answering is SPARQL query generation. LC-QUAD [13] is a dataset with 5,000 questions and corresponding SPARQL queries over DBpedia, which involve logical and quantitative reasoning. LC-QUAD2.0[1] [4] is a dataset of 30,000 questions compatible with both DBpedia and Wikidata[2], which contains more types of SPARQL queries compared with previous version. The queries involve ranking of graph edges, boolean reasoning over more than one triplet and comparative constraints.

In this paper we describe models for SPARQL query generation trained on LC-QUAD2.0 dataset. For translation of a question to a SPARQL query, we first define the type of the query template. Then we fill the empty slots in the template with entities, relations from Wikidata and constraints. For entity detection we use BERT sequence labeling model. Relation ranking is performed by BiLSTM, path ranking—by BERT-based ranking model. We use pretrained cased 12-layer BERT-Base[3]. For extraction of comparative constraints we use regular expressions. Our KBQA system is capable of answering complex questions with logical or comparative reasoning. The proposed KBQA system was released as a component of open-source DeepPavlov library[4].

2. Related work

The first approaches to KBQA considered single-fact questions. Simple Questions [1] is the most widely used dataset for training models to answer single triplet questions. The model of [1] uses memory networks to store candidate facts and then score them by cosine similarity between question and fact vectors (each vector is a product of trainable embedding matrices and bag-of-ngrams representations of the question and fact). In [2] relations and entities in candidate triplets are separately ranked. Dot product of trainable relation embedding and vector representation of the question (final hidden states of BiGRU + linear layer) is used for scoring. Dot product of TransE entity embedding and vector representation of the question is used for entity scoring. Another approach is generation of the query with character-level encoder-decoder architecture [5]. Encoding of questions, entities and relation labels with BiGRU at word and character level is described in [7].

Decomposition of knowledge-base question answering into entity detection, linking, relation prediction and answer parsing components is a simple approach but it is competitive with more complicated architectures [14]. KBQA system proposed in our work consists of the similar steps and several other steps specific for complex questions. The approach of [14] utilises vanilla RNNs for entity detection and relation prediction. These subtasks of KBQA can be solved with BiLSTM and BiGRU [9] and improve accuracy of [14] on Simple Questions dataset.

Query building for complex question answering includes query generation and ranking steps. Model proposed by [8] generates candidate paths in the knowledge graph starting from extracted entities (entity detection and linking is omitted with

1 https://github.com/AskNowQA/LC-QuAD2.0

2 https://www.wikidata.org

3 https://github.com/google-research/bert

4 https://github.com/deepmipt/DeepPavlov

the assumption that correct entities are given). The question and candidate paths are encoded with BiLSTM. Dot products of vectors representing the question and candidate paths are used to rank candidate paths. The approach of [18] uses Tree-LSTM which considers tree representations of candidate walks and the question with respect to the syntactical structure. Assuming that the lists of candidate entities and relations are given, Tree-LSTM produces latent representations of the question and candidate queries, which are ranked by the similarity function.

The model of [15] uses message passing for query ranking, which means propagation of confidence scores from candidate entities and relations to the adjacent nodes in the extracted subgraph. The model also includes entity and relation extraction steps. The substrings in the questions in LC-QUAD dataset, corresponding to entities and relations, are tagged "E1", "E2", "P1", "P2", "C1", "C2", which means "first entity" in the question, "second entity" (if exists), "first relation", "second relation" (if exists), "class of first entity", etc. BiLSTM + CRF network was trained for labeling of question tokens sequence with the corresponding tags. After entity and relation linking, for all entities in the subgraph the confidence scores are aggregated from adjacent entities and the entity with the highest score is considered as the answer.

The work of [12] presents Complex Imperative Program Induction from Terminal Rewards, a model which can perform set, logical and arithmetic operations on the extracted subgraph assuming that the list of gold entities and relations is given. The query is generated with an imperative sequential program. Each step of the program selects the atomic operator and a set of previously defined variables (for example, entities and relations), and writes the result to memory, which is used in subsequent steps. The model achieves state-of-the-art performance for the Complex Sequental Question Answering dataset.

KBQA system, proposed in this work, can perform all the steps of complex question answering from entity extraction to query generation and is capable of answering to both simple questions and complex questions with boolean, quantitative and comparative reasoning from LC-QUAD2.0 dataset.

3. Overview of LC-QUAD2.0 dataset

Numbers of questions and percentage of the total number of questions for different query template types in train and test sets are shown in **Table 1**.

Table 1: Percentage of different query template types in the dataset

Query template type	Percentage of the total number of questions	Number of questions, train set	Number of questions, test set
statement_property	25.5	5,852	1,484
right-subgraph	15.6	3,574	854
center	14.0	3,220	824
Simple question left	7.0	1,604	438
Simple question right	6.5	1,494	378

Query template type	Percentage of the total number of questions	Number of questions, train set	Number of questions, test set
string matching simple contains word	6.4	1,466	338
left-subgraph	6.2	1,418	373
boolean with filter	5.8	1,331	341
rank	4.0	921	210
string matching type + relation contains word	2.9	662	148
two intentions right subgraph	2.6	599	141
boolean double one_hop right subgraph	1.8	411	89
boolean one_hop right subgraph	1.7	399	101

“statement_property” are complex questions which deal with a numerical value or date as an answer or one of the entities (1).

(1) When did Jean-Paul Sartre move to Le Havre?

“center” are single-fact questions (one entity and one relation).

“simple question right” and “simple question left” are single-fact questions and the answer entity is connected with one of the entities in the question with the relation “P31” (“instance of”).

“left-subgraph” questions require finding paths in subgraph of the length of 2.

“right-subgraph” are questions with two entities and two relations.

“two intentions right subgraph” questions contain one entity and two relations and these questions have two answers corresponding to two facts about the grounding entity.

“boolean one_hop right subgraph” and “boolean double one_hop right subgraph” require determining whether one or two facts are true or false. To solve these questions we need to look for these facts in Wikidata and if the facts exist in the knowledge base, we consider the statement true, otherwise the statement is false.

“boolean with filter” questions require comparison of the object entity with the numerical value from the question.

“string matching simple contains word” and “string matching type + relation contains word” are questions where the answer entity should contain a particular letter or word.

“rank” questions require ordering of answer entities by ascending or descending.

4. Components of proposed KBQA system

4.1. KBQA pipeline

We decompose the task of KBQA into query template prediction, entity detection, entity linking, relation ranking, path ranking, constraint extraction (if the question has constraints) and generation of query from extracted entities, relations and constraints.

Let us consider as an example the steps of KBQA (**Figure 1**) for "statement property" question with SPARQL query template (2).

(2) SELECT ?obj WHERE { wd:Q_1 p:P_1 ?s . ?s ps:P_1 ?obj .
?s pq:P_2 ?x filter(contains(?x, N)) }

On entity detection step we extract the entity substring S from the question. After entity linking step we obtain candidate entities $E_1, \ldots, E_N$ with corresponding confidences $P_{E1}, \ldots, P_{EN}$. Then we extract relations $R_1^1, \ldots, R_M^1$, connected to entities $E_1, \ldots, E_N$, rank them with BiLSTM ranking model, and leave 15 relations $R_1^1, \ldots, R_{15}^1$ with maximal confidences $P_{R1}, \ldots, P_{R15}$. Number N for the expression "filter(contains(?x, N))" is extracted from the question with regular expressions. Then we execute SPARQL queries (3)

(3) SELECT ?obj ?p2 WHERE { wd:E_i p:R_j ?s . ?s ps:R_j ?obj .
?s ?p2 ?x filter(contains(?x, N)) }

for combinations $< E_i, R_j >$ of entities $E_1, \ldots, E_N$ and relations ${R_1}^1, \ldots, {R_{15}}^1$ and obtain the list of candidate second relations ${R_1}^2, \ldots, {R_K}^2$. Combinations of relations $< {R_j}^1, {R_k}^2 >$ are ranked with BERT-based ranking model. The model outputs confidences P_{Rjk}. Entity E_i and relations R_j and R_k with maximal confidences product $P_{Ei} \cdot P_{Rjk}$ are filled in the slots of the SPARQL query template (4):

(4) SELECT ?obj WHERE { wd:E_i p:${R_j}^1$?s . ?s ps:${R_j}^1$?obj . ?s pq:${R_k}^2$
?x filter(contains(?x, N)) }

Other types of questions are processed similarly: first we find candidate entities, then extract and rank candidate relations with BiLSTM, if necessary, extract numerical values with regular expressions, find valid combinations of entities and relations according to the query template, rank combinations of relations with BERT and consider combination of entities and relations with maximal product of confidences as the required query.

4.2. Classification of questions by query template type

Query template types "right-subgraph", "simple question right", "simple question left", "left-subgraph", "center" we united into one class.

"statement_property" questions can be translated into 5 types of SPARQL queries, "rank" questions—into 2 types. Each type is considered as a separate class. All other types of questions are put into a separate class. Total number of classes is 14.

Classification of questions is performed with BERT-based model from DeepPavlov library. Output representation of BERT [CLS] token is fed into a dense layer for classification into 14 classes. For comparison we used tf-idf+SVC model (**Table 2**).

Table 2: Accuracy of question classification by query template types

BERT	TF-IDF+SVC
90.8	85.5

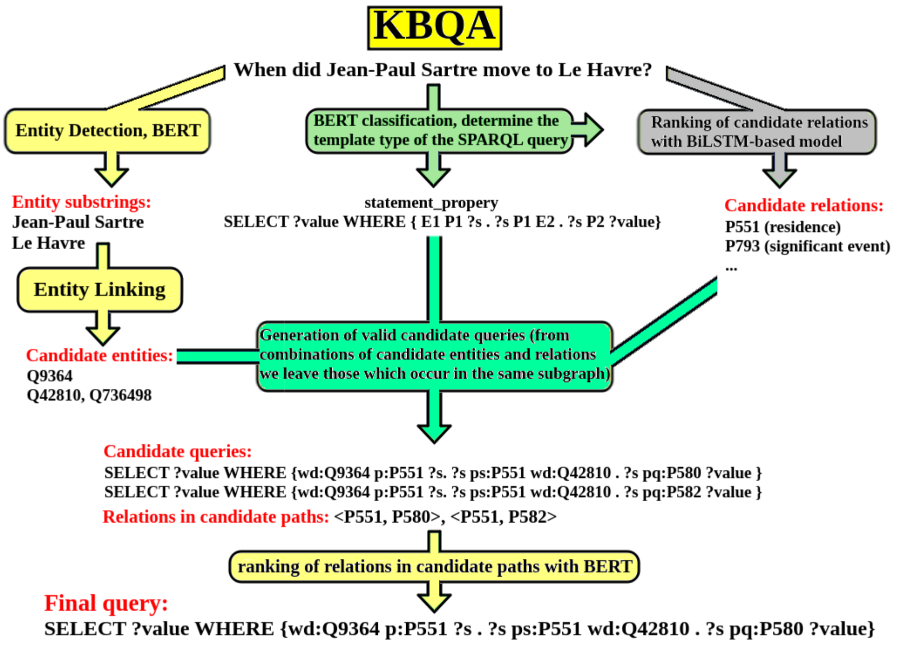
KBQA
When did Jean-Paul Sartre move to Le Havre?
Entity Detection, BERT
BERT classification, determine the template type of the SPARQL query
Ranking of candidate relations with BiLSTM-based model
Entity substrings:
Jean-Paul Sartre
Le Havre
statement_propery
SELECT ?value WHERE { E1 P1 ?s . ?s P1 E2 . ?s P2 ?value}
Candidate relations:
P551 (residence)
P793 (significant event)
...
Entity Linking
Candidate entities:
Q9364
Q42810, Q736498
Generation of valid candidate queries (from combinations of candidate entities and relations we leave those which occur in the same subgraph)
Candidate queries:
SELECT ?value WHERE {wd:Q9364 p:P551 ?s. ?s ps:P551 wd:Q42810 . ?s pq:P580 ?value }
SELECT ?value WHERE {wd:Q9364 p:P551 ?s. ?s ps:P551 wd:Q42810 . ?s pq:P582 ?value }
Relations in candidate paths: <P551, P580>, <P551, P582>
ranking of relations in candidate paths with BERT
Final query:
SELECT ?value WHERE {wd:Q9364 p:P551 ?s . ?s ps:P551 wd:Q42810 . ?s pq:P580 ?value}

Figure 1: KBQA pipeline

4.3. Entity Detection and Entity Linking

Entity Detection is implemented as labeling of sequence of question tokens

$$q_{seq} = \{w_1, w_2, ..., w_n\} \tag{1}$$

with one of two labels: "I-TAG" if the token w_i is in the entity substring and "O-TAG" otherwise. For example, in question

(5) When did Jean-Paul Sartre move to Le Havre?

tokens "Jean-Paul", "Sartre", "Le", "Havre" are labelled with "I-TAG", the other tokens with "O-TAG". We prepared the dataset from LC-QUAD2.0 for Entity Detection using labels of gold entities to find substrings in questions, corresponding to entities and annotated matched tokens as "I-TAG". This dataset is used for training of BERT-based sequence labeling model from DeepPavlov library. Output representations of question sub-tokens are fed into a dense layer for classification of sub-tokens into classes, corresponding to two tags (Figure 2). We obtained F1-score of 87 on test-set.

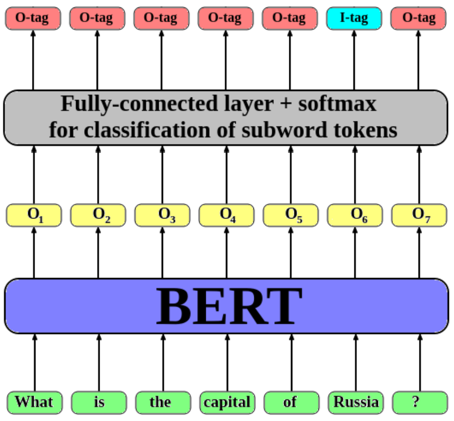

Figure 2: BERT for sequence tagging

For all entities in Wikidata we built an inverted index over unigrams in entity's label (a dictionary where keys are tokens and values are lists of entities containing these tokens). Entity Linking is implemented using fuzzy matching of the string extracted at Entity Detection step with inverted index. For example, tokens "Jean-Paul" and "Sartre" from the substring "Jean-Paul Sartre" are used as keys to obtain the list of candidate entities, and candidate entity "Q9364" with the label "Jean-Paul Sartre" has the maximum fuzz ratio of 100. Candidate entities, extracted from inverted index dictionary, are ranked by fuzz ratio of their titles with the entity substring and number of relations (the more relations an entity has, the more popular it is).

4.4. Model of relation ranking

The model of relation ranking is inspired by [17] (Figure 3). The sequence of question tokens $q_1, ..., q_n$ is passed through an embedding lookup layer. The sequence of Word2Vec embeddings $e_1, ..., e_n$ is the input of 2-layer BiLSTM to encode the token sequence with hidden representations $h_1, ..., h_n$. For linked entities we extract all relations from Wikidata which these entities have and consider them as candidate relations. Candidate relations are encoded with PyTorch-BigGraph embeddings [6] $rel\_emb_1, ..., rel\_emb_k$. Dot products of each candidate relation embedding and hidden states $rel\_emb_i \cdot h_1, ..., rel\_emb_i \cdot h_n$ are passed throught softmax layer to obtain coefficients $\alpha_1, ..., \alpha_n$. Then we sum hidden states weighted with coefficients:

$$q = \sum_{j=1}^{n} \alpha_i \cdot h_j \tag{2}$$

The model is trained to maximize dot product $q \cdot rel\_emb_i$ if rel_emb_i is the embedding of the right relation and minimize if rel_emb_i is the embedding of the wrong relation.

$q \cdot rel\_emb_i$ is the confidence that rel_emb_i is the right relation. For example, for the question

(6) What periodical literature does Delta Air Lines use as a mouthpiece?

with the corresponding SPARQL query

(7) SELECT DISTINCT ?obj WHERE wd:Q188920 wdt:P2813 ?obj . ?obj wdt:P31 wd:Q1002697

the model is trained to output maximal dot product for embeddings of relations P2813 and P31.

For every question Q_k in test set we extracted candidate relations $R_1^k, \dots, R_n^k$ for gold entities $E_1^k, \dots, E_m^k$. Candidate relations are ranked with relation ranking model and we check if candidate relation with the maximum score is one of the gold relations $R_{1g}^k, \dots, R_{mg}^k$. We measure the percent of questions in test set which have one of the gold relations ranked with the highest score (84% of questions) (Table 3). The model is more accurate if PyTorch-BigGraph embeddings of relations are replaced with average embeddings of relation title tokens (89%).

Table 3: Percent of questions in test set with one of the gold relations ranked with highest score

Relation embeddings used in the model	% of questions
PyTorch-BigGraph	84
Word2Vec	89

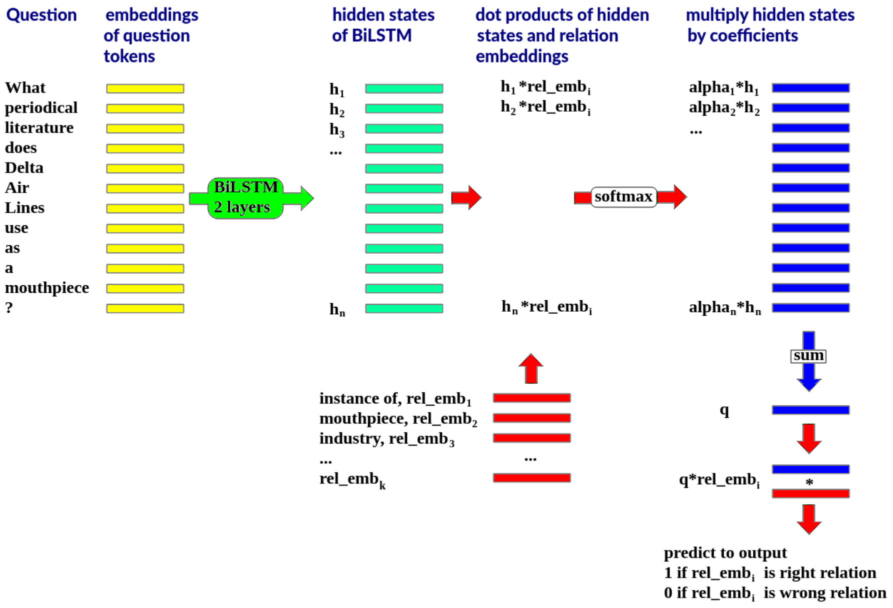

Figure 3: Relation ranking network

4.5. BERT for path ranking

Path ranking model is inspired by [8]. The input to the model is the following: the question q followed by [SEP] token and candidate path

$$C_i = \{R_1, ..., R_L\}, L \in \{1, 2\} \tag{3}$$

from the set of candidate paths $C_1, ..., C_n$. For example, one of the candidate paths for the question

(8) What is stable version of user interface of Amazon Kindle?

is {P1414, P348}, where the label of relation with identificator in Wikidata "P1414" is "GUI toolkit or framework" and the label of "P348" is "software version". So, the input to BERT is the question q and relation titles "GUI toolkit or framework" and "software version" (Figure 4).

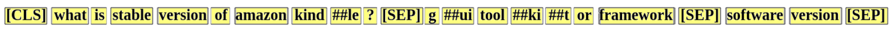

Figure 4: BERT input representation

Output representation of BERT [CLS] token is fed into a dense layer for binary classification into 2 classes: 1 if the candidate path is the gold path for the question (positive sample) and 0 otherwise (negative sample). For training of the model we generated negative samples in the ratio of 20:1 to positive samples. The model achieves F1 of 87.2 on the test set.

4.6. Using regular expressions

Regular expressions are used in "statement_property" questions for extraction of dates and numerical values. "rank" questions require determination of the order of answer ranking (ascending or descending). For example, words "What is the highest", "the biggest", "the longest", etc. point at ascending order (corresponding to "ORDER BY ASC(?obj)" in the SPARQL query) and "the smallest", "the lowest", etc. point at descending order ("ORDER BY DESC(?obj)"). Such keywords are extracted with regular expressions. In "boolean with filter" questions regular expressions are used for extraction of numerical values and comparison operators. For example, in the question

(9) Is the maximum wavelength of sensitivity of the human eye equal to 700?

the numerical value is "700" and "equal to" corresponds to "=". So the SPARQL query for the question is

(10) ASK WHERE wd:Q430024 wdt:P3737 ?obj filter(?obj = 700)

Table 4: Question answering accuracy

Query template type	Answering accuracy
statement_property	51.5
right_subgraph	33.3
center	78.1
Simple question left	67.3
Simple question right	68.7
string matching simple contains word	80.4
left-subgraph	27.9
boolean with filter	75.9
rank	48.5
string matching type + relation contains word	46.9
two intentions right subgraph	43.4
boolean double one_hop right subgraph	63.8
boolean one-hop right subgraph	59.1
Total	56.3

4.7. Results of the KBQA system on LC-QUAD2.0 dataset

Question answering accuracy for different types of questions is shown in Table 4. The answer is considered correct if the answer entities and numerical values or dates match with gold answers. The proposed KBQA system gives correct answers to almost one-half of double-fact questions with numerical values or dates ("statement_property") and questions with ranking of answers ("rank"). The system achieves high scores on single-fact questions ("center", "boolean with filter", "string matching simple contains word"). Two-hop questions ("left-subgraph" and "right-subgraph") present difficulties to the system and are the subject of further research and improvement of the model.

4.8. Results of the KBQA system on LC-QUAD1.0 dataset

We divide questions in LC-QUAD1.0 into the following types: simple (one entity and one relation in the SPARQL query), simple with type (one entity, one relation and entity, which defines the type of answer entities), double (two entities and two relations), 2-hop (one entity and two relations) and boolean (the SPARQL query contains two entities, one relation and "ASK WHERE" keywords). BERT-based model is used for classification of the question by 5 query template types. Extraction of keywords, such as "how many", "count", etc. is used to define whether the question requires counting of number of answer entities.

The other details of the solution are the same as in KBQA system for LC-QUAD2.0, excepting additional tag "T-tag" in BERT sequence labeling model for extraction of substrings corresponding to the type of the entity.

Our model outperforms QAmp [15] and WQAqua [3][5]. We did not compare our model with [8], because their work does not consider entity detection and linking steps.

[5] http://lc-quad.sda.tech/lcquad1.0.html

Table 5: Question answering accuracy

System	Precision	Recall	F1 score
Our model	0.60	0.66	0.63
QAmp	0.25	0.50	0.33
WQAqua	0.22	0.38	0.28

5. Conclusion

In this work, we have described question answering system over Wikidata knowledge base. The system translates a natural language question into a query in SPARQL language, execution of which gives an answer. The proposed KBQA system is capable of answering complex questions which require logical or comparative reasoning. The system is the first solution to LC-QUAD2.0 dataset, and we evaluated the performance of the system for different types of questions in LC-QUAD2.0.

References

1. *Bordes, A. et al.:* Large-scale simple question answering with memory networks. CoRR. abs/1506.02075, (2015).
2. *Dai, Z. et al.:* CFO: Conditional focused neural question answering with large-scale knowledge bases. In: Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers). pp. 800–810 Association for Computational Linguistics, Berlin, Germany (2016).
3. *Diefenbach, D. et al.:* Towards a question answering system over the semantic web. CoRR. abs/1803.00832, (2018).
4. *Dubey, M. et al.:* Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In: International semantic web conference. pp. 69–78, Springer (2019).
5. *He, X., Golub, D.:* Character-level question answering with attention. In: Proceedings of the 2016 conference on empirical methods in natural language processing. pp. 1598–1607, Association for Computational Linguistics, Austin, Texas (2016).
6. *Lerer, A. et al.:* Pytorch-biggraph: A large-scale graph embedding system. arXiv preprint arXiv:1903.12287. (2019).
7. *Lukovnikov, D. et al.:* Neural network-based question answering over knowledge graphs on word and character level. In: Proceedings of the 26th international conference on world wide web. pp. 1211–1220, International World Wide Web Conferences Steering Committee, Republic; Canton of Geneva, CHE (2017).
8. *Maheshwari, G. et al.:* Learning to rank query graphs for complex question answering over knowledge graphs. In: International semantic web conference. pp. 487–504, Springer (2019).

9. *Mohammed, S. et al.:* Strong baselines for simple question answering over knowledge graphs with and without neural networks. In: Proceedings of the 2018 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 2 (short papers). pp. 291–296, Association for Computational Linguistics, New Orleans, Louisiana (2018).
10. *Rajpurkar, P. et al.:* Know what you don't know: Unanswerable questions for squad. arXiv preprint arXiv:1806.03822. (2018).
11. *Rajpurkar, P. et al.:* SQuAD: 100,000+ questions for machine comprehension of text. In: Proceedings of the 2016 conference on empirical methods in natural language processing. pp. 2383–2392, Association for Computational Linguistics, Austin, Texas (2016).
12. *Saha, A. et al.:* Complex program induction for querying knowledge bases in the absence of gold programs. Transactions of the Association for Computational Linguistics. 7, 185–200 (2019).
13. *Trivedi, P. et al.:* Lc-quad: A corpus for complex question answering over knowledge graphs. In: International semantic web conference. pp. 210–218, Springer (2017).
14. *Ture, F., Jojic, O.:* No need to pay attention: Simple recurrent neural networks work! In: Proceedings of the 2017 conference on empirical methods in natural language processing. pp. 2866–2872, Association for Computational Linguistics, Copenhagen, Denmark (2017).
15. *Vakulenko, S. et al.:* Message passing for complex question answering over knowledge graphs. In: Proceedings of the 28th acm international conference on information and knowledge management. pp. 1431–1440 (2019).
16. *Wilcke, X. et al.:* The knowledge graph as the default data model for learning on heterogeneous knowledge. Data Science. 1, 1–2, 39–57 (2017).
17. *Xiong, W. et al.:* Improving question answering over incomplete kbs with knowledge-aware reader. arXiv preprint arXiv:1905.07098. (2019).
18. *Zafar, H. et al.:* Formal query generation for question answering over knowledge bases. In: European semantic web conference. pp. 714–728, Springer (2018).