Interactive Instance-based Evaluation of Knowledge Base Question Answering

Daniil Sorokin and **Iryna Gurevych**
Ubiquitous Knowledge Processing Lab (UKP) and Research Training Group AIPHES
Department of Computer Science, Technische Universität Darmstadt
www.ukp.tu-darmstadt.de

Abstract

Most approaches to Knowledge Base Question Answering are based on semantic parsing. In this paper, we present a tool that aids in debugging of question answering systems that construct a structured semantic representation for the input question. Previous work has largely focused on building question answering interfaces or evaluation frameworks that unify multiple data sets. The primary objective of our system is to enable interactive debugging of model predictions on individual instances (questions) and to simplify manual error analysis. Our interactive interface helps researchers to understand the shortcomings of a particular model, qualitatively analyze the complete pipeline and compare different models. A set of sit-by sessions was used to validate our interface design.

1 Introduction

Knowledge base question answering (QA) is an important natural language processing problem. Given a natural language question, the task is to find a set of entities in a knowledge base (KB) that constitutes the answer. For example, for a question "Who played Princess Leia?" the answer, "Carrie Fisher", could be retrieved from a general-purpose KB, such as Wikidata[1]. A successful KB QA system would ultimately provide a universally accessible natural language interface to factual knowledge (Liang, 2016).

KB QA requires a precise modeling of the question semantics through the entities and relations available in the KB in order to retrieve the correct answer. It is common to break down the task into three main steps: entity linking, semantic parsing or relation disambiguation and answer retrieval. We show in Figure 1 how the outlined steps lead to an answer on an example question and how the output of each step is re-used in the next one. This approach has been exhibited by the most of the recent works on the KB QA (Berant and Liang, 2014; Reddy et al., 2016; Yih et al., 2015; Peng et al., 2017; Sorokin and Gurevych, 2018b). The multi-step pipeline poses particular challenges for error analysis, since many unique errors can arise at different processing stages. With our tool, we aim to support the evaluation of QA systems and to help to identify problems that do not necessary form generalisable error patterns, but hinder the overall system performance nonetheless.

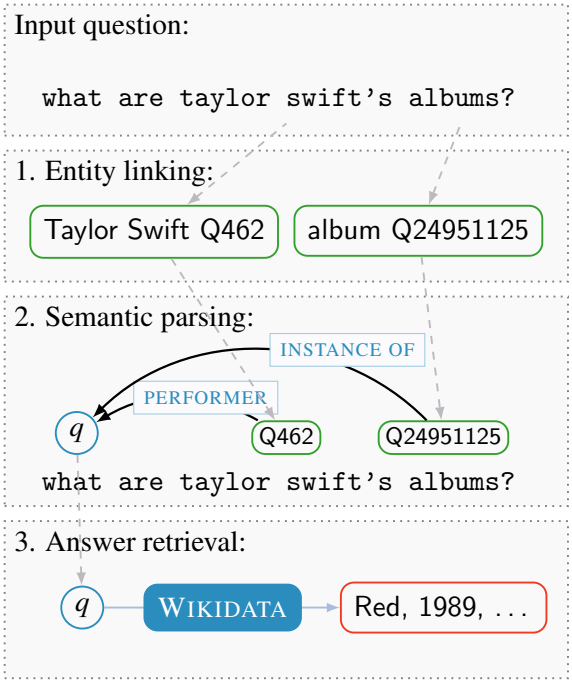

Figure 1: Typical steps undertaken by a QA system. Gray dashed arrows show how the output of the previous step is passed into the next one. Qxxx stands for a KB identifier

Some frameworks have been introduced recently to streamline the evaluation of KB QA systems. The ParlAI framework focuses on building a uni-

[1] https://www.wikidata.org/

fied interface for multiple QA data sets (Miller et al., 2017), while GerbilQA[2] introduces evaluation of individual steps of a QA pipeline. However, none of them addresses an interactive debugging scenario, that can be used by the researchers to do instance-base error analysis. This is especially relevant in the context of such benchmarks as QALD, where each individual question is meant to test a particular aspect of the system and debugging individual instances is crucial for understanding of the system performance (Unger et al., 2016).

Another set of tools has focused on building an infrastructure to support the development for KB QA. Ask Wikidata[3] offers an easy way to post queries to Wikidata via a web-based interface, though the tool relies on manual disambiguation to understand a question. The WDAqua project[4] has produced a speech-based plug-in interface (Kumar et al., 2017) and the Qanary specification for QA systems (Singh et al., 2016). These tools follow the described steps of the QA pipeline, but do not facilitate the interactive instance-based evaluation that is the main aspect of this work.

Main contribution In this paper, we present a modular debugging application for KB QA that can be used to manually evaluate the main steps of a QA pipeline. Our system focuses on the analysis of individual examples and a detailed view of possible causes of errors, so that individual error propagation cases can be identified.

Demo system and the code A demo instance with the default QA model is running at the following url: `http://semanticparsing.ukp.informatik.tu-darmstadt.de:5000/question-answering/`. Our system is freely available: `https://github.com/UKPLab/emnlp2018-question-answering-interface`.

2 A prototypical QA pipeline and the requirements

The first stage of every QA approach is entity linking (EL), that is the identification of entity mentions in the question and linking them to entities in KB. In Figure 1, two entity mentions are detected and linked to the KB referents. According to multiple error analyses, entity linking is a common source of errors in a QA system (Berant and Liang, 2014; Reddy et al., 2016).

[2] `http://aksw.org/Projects/GERBIL.html`
[3] `https://tools.wmflabs.org/bene/ask/`
[4] `http://wdaqua.eu`

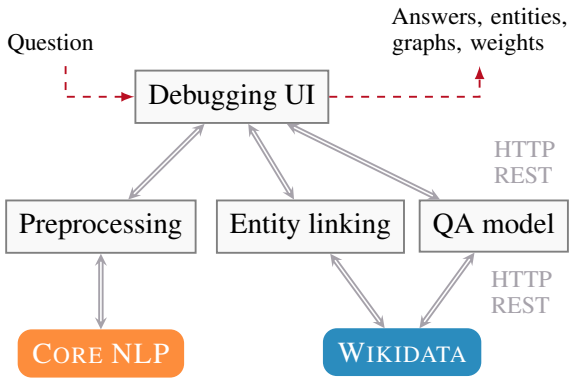

Figure 2: Overview of the system architecture

The entity linking stage is followed by semantic parsing that consists of combining the extracted entities into a single structured meaning representation. The entities are connected with semantic relations to a special question variable that denotes the answer to the question (Yih et al., 2015).

Given the ambiguity of the natural language, a semantic parsing model constructs multiple representations that can match the question and assigns probabilities to them (Liang, 2016). It is common to learn a vector encoding for the question and the structured representations and then use a similarity function to compute the probabilities (Yih et al., 2015; Reddy et al., 2016). The most probable structured representation is then translated into a query and used to extract the answer from the KB.

Some approaches circumvent building a structured representation and instead directly compose vector encodings of the potential answers (Dong et al., 2015). Since this is a less common architecture type for KB QA, we focused on semantic parsing approaches while developing our interface.

The described pipeline lets us outline the main requirements for an interactive debugging tool:

1. It needs to represent all stages of the QA pipeline in a sequential manner to let the user identify where the error occurs and how it propagates.

2. It needs to account for the specific properties of semantic parsing approaches to KB QA, such as structured semantic representations.

3. It needs to include an analysis block that shows if the model has learned meaningful vector representations.

```
entity.linking:
  model: models/elmodel.pkl
  max.entity.options: 5
  max.ngram.len: 4

model:
  model.file: models/qamodel.pkl

evaluation:
  max.num.entities: 3
  beam.size: 10
  ...

wikidata:
  backend: localhost:8890/sparql
  timeout: 10
  ...
```

Listing 1: A snippet of the YAML configuration file with the main modules and settings

3 System overview

Our system consists of a web-based front-end and a set of back-end services that communicate through HTTP REST API (see Figure 2). The front-end contains the interactive debugging user interface (Section 4). A separate request is sent to the back-end service for each processing step of the QA pipeline. Thus, we are able to show the results to the user as they are being delivered by the back-end services. The front-end is responsible for aggregating and visualizing the information after each step. In case any service fails, a partial result from the previous steps would be available to the user.

The back-end services include the pre-processing, the entity linking and the semantic parsing modules. The pre-processing module performs tokenization and part-of speech tagging using the Stanford CoreNLP toolkit (Manning et al., 2014). The entity linking module recognizes mentions of the KB entities in the input question. We provide a default model for entity linking on Wikidata that is freely available feature-based implementation of Sorokin and Gurevych (2017).

The semantic parsing module includes the construction of structured semantic representations and a learned model that selects the correct representation. The integrated model uses a convolutional neural network architecture to learn vector encodings for questions and semantic representations (Sorokin and Gurevych, 2018a). We provide an integrated model for demonstartion purposes, while the main purpose of the tool is to enable manual evaluation and comparison of new models. We define an interface that a user needs to implement in order to integrate their own model into the tool.

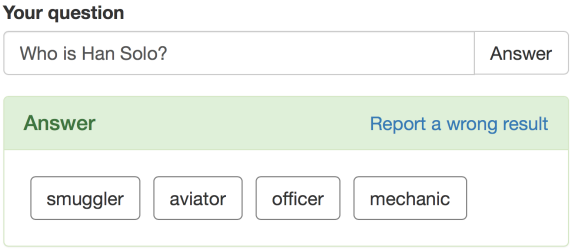

Figure 3: The main input field and the answer block

Finally, using the KB query provided by the semantic parsing module, the front-end retrieves the answers from the KB. The back-end modules can be configured using a YAML properties file (see Listing 1 for an example configuration).

3.1 Implementation details

We implement the user interface and the back-end services with modern web technologies, such as JQuery, D3.js and Bootstrap. The back-end services are implemented in Python with Flask. It is configurable and can be further easily extended with other models for entity linking and QA.

A new QA model can be integrated either as a Python module or as a separate REST service. To communicate the results to the front-end, the service has to send the response in the defined JSON format. We refer to the published code repository for additional details on the implementation.

4 User interface

The interactive web-based user interface (UI) is the central element of our system. We have designed the interface for an expert user and have considered the following user traits (Raskin, 2000): a background in KB QA, a knowledge of programming languages, an interest in manual error analysis.

The UI is modeled after the prototypical QA pipeline as described in Section 2. Each step of the pipeline is represented as a separate block in the interface (see the complete UI depicted in Figure 5). That is, the represented model of the UI directly corresponds to the implementation model of the prototypical QA system. Such design choice is appropriate for tools aimed at domain experts who already have a clear mental model of the underlying processes and it makes the UI understandable for the first time (expert) users (Cooper et al., 2007).

Consequently, the interface is divided into blocks

that correspond to the steps of the QA pipeline. There are several base blocks that are fixed and can not be removed, such as the input question block, while the other ones can be hidden if needed.

Input question and answer block The first block consists of the input question field and the answer area (Figure 3). When the user first loads the interface, only the input question field is presented. This avoids the confusion as to what is the starting point of the interactions with the system. Further elements appear only when the corresponding results are returned. For example, the answers area is only shown when the processing is finished.

Although the answer retrieval from the KB is the last step of the pipeline, we put the answer area right below the input question field. This design choice makes it easy to see right away if the system has processed the question correctly.

Entity linking block In this block, we list all identified entity mentions in the input question and the top 5 entity disambiguation candidates. The entity candidate with the highest score is automatically selected and forwarded to the QA model.

The list of entity disambiguation candidates is interactive and the user can select all or none candidates for each entity mention. In case multiple candidates are selected, all of them are sent to the QA system as separate entities. This lets the user correct potential entity linker mistakes by selecting some other than the top disambiguation candidate and continue to debug the rest of the pipeline.

Semantic graphs We visualize the structured semantic representations that the QA model generates as graphs. That is the most common way to visually depict structured representations (Yih et al., 2015; Reddy et al., 2016; Sorokin and Gurevych, 2018a). A semantic graph consists of a question variable node that denotes the answer to the question, KB entities and KB relation types.

We use circles to depict entities and solid lines between them to show relations. In each graph the question variable node is represented with a high-contrast blue circle. Since most relations are attached to the question node, this makes it easier to parse the structure of the graph. Additionally, since each graph has only one high contrast node, the user can identify at the first sight how many graphs have been composed for the input question. For example, Figure 4 shows four semantic graphs for a question "Who played Princess Leia in Star Wars?", that are clearly visible. In this instance, the model selects an incorrect graph (highlighted in green) and retrieves all cast members of the Star Wars movie. The correct graph would be the second one, that also uses the entity Princess Leia.

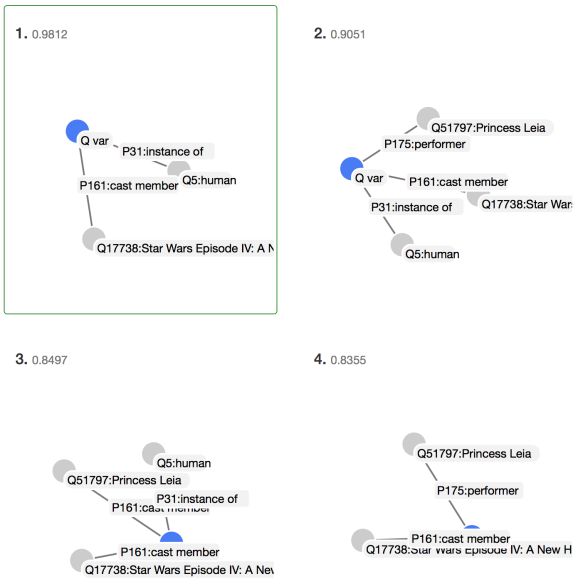

Figure 4: The semantic graphs block for the question "Who played Princess Leia in Star Wars?"

Representation analysis The visual inspection of the learned vector representation in the two final blocks makes it possible to identify implementation or training errors in the QA model. Once the error is attributed to the learned model, a researcher can continue to inspect the model in the tool that would be the most appropriate to inspect the weights of a particular model architecture (e.g. TensorBoard[5]).

The token-level representations block visualizes the weights computed by the model for each input token of the question. This kind of visual analysis is helpful to identify if the model is learning meaningful word representations. We rely on the shade and saturation visual variables to encode the computed vectors. Each vertical line corresponds to a vector dimension and the darker saturated colors denote a higher numerical value. In Figure 5, one can see that the model is assigning the highest weights to the main entity in the question.

The second representation analysis block places the vector representation of the question and of the semantic relations on a 2D-plane using

[5] https://www.tensorflow.org/programmers_guide/summaries_and_tensorboard

t-SNE (van der Maaten and Hinton, 2008). The vectors of the relations that appear in the generated semantic graphs are rendered as red circles. The user can zoom in and out to compare the position of the question vector to the surrounding relations.

5 User study

The interface was designed with the requirements that were outlined in Section 2. As domain experts, we were personally interested in applying the developed tool to QA systems. To verify that the interface and the designed interactions are in line with expectations of the first-time users, we have conducted a set of brief sit-by testing sessions. Sit-by sessions are usually used for exploratory situations and gathering first impressions about the design of a product (Rubin and Chisnell, 2008).

In the user study, we were aiming to evaluate: can a person familiar with KB QA use the tool independently? Does the developed tool make it possible to manually identify errors in a QA pipeline? Two participants with background in natural language processing and linguistics were asked to perform a simple analysis task while a moderator was sitting near them and monitoring the progress. The participants were asked to input a list of three questions into the tool and tell if the model succeeded in answering them. In case of an incorrect answer, we have expected participants to be able to identify the stage of the pipeline that caused the error. During the sit-by sessions, we were able to confirm that the interface is intuitive and easy to use. All the participants were able to complete the task in under 10 minutes and could point out at what stage an error has occurred for all input questions. The representation analysis instruments, on the contrary, have proven to be the least intuitive element of the interface. Although the participants could attribute the error to the model, they were unable to say if the learned vector representations were meaningful based on the provided visualization.

6 Conclusions

In this work, we have presented an interactive debugging tool for semantic parsing approaches to KB QA. We have started by defining the main requirements for an instance-based evaluation tool and then demonstarted how the different aspects of the designed interface fulfill them. Our tool enables researchers to explore and qualitatively analyse a developed QA pipeline. We used sit-by sessions to verify the design choices and to assess the usability of the tool. Our architecture includes default models for entity linking and question answering, which makes it easy to replace only one of the components with a new module.

Acknowledgments

This work has been supported by the German Research Foundation as part of the Research Training Group AIPHES (grant No. GRK 1994/1), and via the QA-EduInf project (grant GU 798/18-1 and RI 803/12-1). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X GPU.

References

Jonathan Berant and Percy Liang. 2014. Semantic Parsing via Paraphrasing. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1415–1425, Baltimore, MD, USA.

Alan Cooper, Robert Reimann, and David Cronin. 2007. *About Face 3: The Essentials of Interaction Design*. John Wiley & Sons.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question Answering over Freebase with Multi-Column Convolutional Neural Networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 260–269.

Ashwini Jaya Kumar, Christoph Schmidt, and Joachim Khler. 2017. A knowledge graph based speech interface for question answering systems. *Speech Communication*, 92:1–12.

Percy Liang. 2016. Learning Executable Semantic Parsers for Natural Language Understanding. *Communications of the ACM*, 59(9):68–76.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing High-Dimensional Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605.

Christopher D. Manning, John Bauer, Jenny Finkel, Steven J. Bethard, Mihai Surdeanu, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics (ACL): System Demonstrations*, pages 55–60, Baltimore, MD, USA.

A. H. Miller, W. Feng, A. Fisch, J. Lu, D. Batra, A. Bordes, D. Parikh, and J. Weston. 2017. ParlAI: A Dialog Research Software Platform. *arXiv preprint arXiv:1705.06476*.

Haoruo Peng, Ming-Wei Chang, and Wen-Tau Yih. 2017. Maximum Margin Reward Networks for Learning from Explicit and Implicit Supervision. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2358–2368, Copenhagen, Denmark.

J Raskin. 2000. *The Humane Interface: New Directions for Designing Interactive Systems*. Addison-Wesley.

Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. 2016. Transforming Dependency Structures to Logical Forms for Semantic Parsing. *Transactions of the Association for Computational Linguistics*, 4:127–140.

J Rubin and D Chisnell. 2008. *Handbook of Usability Testing: Howto Plan, Design, and Conduct Effective Tests*, 2nd edition. Wiley Publishing.

Kuldeep Singh, Andreas Both, Dennis Diefenbach, Saedeeh Shekarpour, Didier Cherix, and Christoph Lange. 2016. Qanary – The Fast Track to Creating a Question Answering System with Linked Data Technology. In *The Semantic Web*, pages 183–188, Cham. Springer International Publishing.

Daniil Sorokin and Iryna Gurevych. 2017. End-to-end representation learning for question answering with weak supervision. In *Semantic Web Challenges: 4th SemWebEval Challenge at ESWC 2017*, volume 769 of *Communications in Computer and Information Science*, pages 70–83, Cham. Springer International Publishing.

Daniil Sorokin and Iryna Gurevych. 2018a. Mixing Context Granularities for Improved Entity Linking on Question Answering Data across Entity Categories. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics (*SEM)*, pages 65–75, New Orleans, LA, USA. Association for Computational Linguistics.

Daniil Sorokin and Iryna Gurevych. 2018b. Modeling semantics with gated graph neural networks for knowledge base question answering. In *Proceedings of COLING 2018, the 27th International Conference on Computational Linguistics*, pages 3306–3317. Association for Computational Linguistics.

Christina Unger, Axel-Cyrille Ngonga Ngomo, and Elena Cabrio. 2016. 6th Open Challenge on Question Answering over Linked Data (QALD-6). In *Semantic Web Challenges*, pages 171–177, Cham. Springer International Publishing.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic Parsing via Staged Query Graph Generation: Question Answering with Knowledge Base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, pages 1321–1331, Beijing, China.

Figure 5: The complete unrolled UI for the question "Who played Luke Skywalker in Star Wars?"