SliQA-I: TOWARDS COLD-START DEVELOPMENT OF END-TO-END SPOKEN LANGUAGE INTERFACE FOR QUESTION ANSWERING

Yilin Shen, Yu Wang, Abhishek Patel, Hongxia Jin

Samsung Research America, Mountain View, CA, USA
{yilin.shen,yu.wang1,abhishek.p,hongxia.jin}@samsung.com

ABSTRACT

Question answering (QA) has become a key capability for voice enabled personal assistants to automatically answer various user questions. However, the development of a spoken language interface for QA in a new domain is time consuming and requires a lot of human labors. Thus, it is crucially desirable to design an end-to-end system, referred to as SliQA, that can facilitate developers to easily and quickly build a QA interface from scratch and output a high quality plug-and-play QA engine. In this paper, we take the first step of SliQA system design, named SliQA-I, to support answering factoid questions regarding an entity over existing knowledge graphs. SliQA-I incorporates a novel iterative human-in-the-loop question generator and an enhanced deep coupled QA engine, thereby requiring light human workload. We implement the real system and evaluate it on three domains from different aspects. The results show that the QA performance of SliQA-I achieves up to 3.58% accuracy gain compared with baseline approaches which use existing QA engine on human generated data. More importantly, we show that SliQA-I only takes as low as 0.025 second to generate a question which has similar quality as human generated ones in terms of both naturalness and grammatical correctness.

Index Terms: spoken language interface, question generation, question answering, cold start, end-to-end system

1. INTRODUCTION

Artificially intelligent voice-enabled personal assistants (PA) have been emerging in our daily life, such as Alexa, Google Assistant, Siri, etc. The capabilities of answering user questions, referred to as *QA skills*, are one of key business drivers for PAs and the number grows rapidly [1]. The goal of building a QA skill is to develop a natural/spoken language interface such that users can use it to interact with PAs using voice to ask questions. However, the development of QA skills in existing PAs has two drawbacks: first, it requires developers to manually generate a large amount and varieties of questions; secondly, it only supports question understanding and heavily relies on third-party services to derive answers. Thus, it is highly desirable to help developers build new QA skills easily and quickly without depending on other third-parties.

With the rising of structured knowledge graphs (KG) such as DBpedia [2] and Freebase [3], automatic question answering using KG has attracted increasing attention recently from both the industrial and academic communities. The most relevant work [4] developed an end-to-end QA system, which however requires an input of corpora with already disambiguated entities. Other works, including question generation [4, 5, 6] and automatic question answering engine [7, 8, 9, 10], require heavy human workload to generate questions or handcrafted templates, making it hard to adapt into new domains. More importantly, in practice, these approaches cannot correctly answer user questions due to the failure of understanding various user expressions.

To overcome the above limitations, we design an *Spoken language interface for Question Answering* (SliQA-I) system to *guide* developers to quickly build up a new QA skill *from scratch*. Taking a new *domain name* as input from developers, SliQA-I generates factoid questions with their corresponding answers iteratively via a novel hybrid rule-based and data-driven approach; and uses them to train an enhanced QA engine with novel skip connection layers. A software developer only needs to iteratively *prune* incorrect generated sentences and then can directly use the well-trained QA engine output, rendering SliQA-I practically very useful.

2. RELATED WORK

Question answering thrived with IBM Watson [11] and attracted rising research attentions over knowledge graphs.

Question generator: Olney *et al.* [5] and Duma *et al.* [12] predefined set of relationship specific question templates. Ngonga Ngomo *et al.* [13] tackled such query verbalization problem for SPARQL. Seyler *et al.* [4] improved the query verbalization which however requires corpora with annotations of already disambiguated entities. Serban *et al.* [6] trained a recurrent neural network on large-scale question-answer pair dataset for question generation. Unfortunately, these approaches cannot be utilized in practice due to either heavy human workload or lack of required inputs.

QA engine: Most recent works [7, 8, 9, 10, 14] are designed based on predefined templates. However, they usually fail to understand and correctly answer many various

expressions of a question. Another closely relevant deep learning based approach [15] using RNN is the state-of-the-art spoken language understanding (SLU) engine to best understand the user expressions and answer factoid questions. Yet, the assumption of the existence of large-scale annotated training data is unrealistic in practice.

3. SliQA-I SYSTEM

The goal of SliQA-I system is to facilitate software developers to build the QA skill in new domains from scratch. The output QA engine can answer user questions, in the format of spoken language, about the attributes of entities in this new domain over knowledge graphs (KG). A KG, such as DBpedia [2], YAGO [16], Freebase [3], consists of entities that are linked to each other with relationships, i.e., a set of triples ($\langle subject \rangle, \langle predicate \rangle, \langle object \rangle$) representing the facts, e.g., ($\langle$*the Louvre*$\rangle, \langle$*LocatedIn*$\rangle, \langle$*Paris*$\rangle$). The focus of our SliQA-I system is to answer basic factoid questions (questions concerned with the subject and relationship of a fact and answered by the object of the fact).

3.1. Iterative Tagged QA Generator

As one may have inferred from the title, our SliQA-I system is designed based on two key ideas: *First, to tackle cold start, we iteratively generate new sentences with human-in-the-loop pruning of incorrect ones and finally generate question answer pairs.* Technically, our approach decomposes the complex question generation task into small subtasks and solves them individually. From developers' perspective, the total amount of their workload will be largely reduced since the next iteration generates more sentences *only* based on the previously selected *correct* ones. *Second, to overcome the semantic ambiguity of entities, we generate sentences with tagged entities as well as tagged answers.*

Next, we introduce the details of our proposed iterative QA generation algorithm, from the rule-based iteration flow to a data-driven subroutine.

Rule-based Iteration Design. Given an answer (tagged attribute), we consider generating two types of questions containing the main tagged entity, *wh-questions* and *commands*. The key idea of our iterative generator is to decompose the task into subtasks (in Figure 1) and tackle each one by one. In each iteration i, *we generate sentences S_i using a hybrid rule-based and data-driven approach and output the correct ones S_i' after human pruning.*

Iteration 1. Entity Expansion. We start with the tagged entity $@E$ defined by the input domain name and expand it into statements and phrases with each of its tagged attributes in KG, where the tagged attributes are the common concepts of entities which connects to one of the entity instances of $@E$. For each new tagged attribute $@A$, we insert $@A$ after $@E$ to generate statements (e.g., @LANDMARK _ @LOCATION) and insert it type keyword A before $@E$ to generate phrases (e.g., *the location* _ @LANDMARK). Here, the joiner words are considered as word placeholders "_" (or called blanks) which will be generated using tagged sentence filler algorithm. For each statement or phrase $s \in S_1$, the expanded tagged attribute $@A$ will be the answer of questions later generated on top of s.

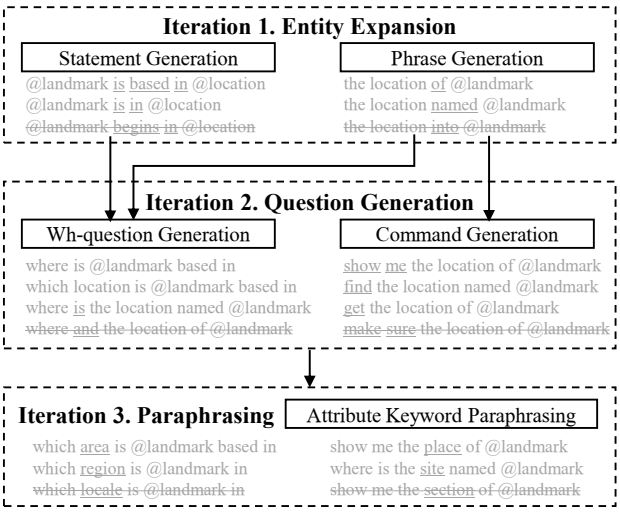

Fig. 1: Iterative QA Generator in SliQA-I System. Grey parts are the sample outputs of each component, in which the strikethrough parts are bad sentences pruned by software developers. The underlined parts are the filled words using data-driven tagged sentence filler.

Iteration 2. Question Generation. As shown in Figure 1, we further generate the wh-questions from both labeled sentences and phrases from iteration 1. For each $s \in S_1'$ associated with an answer $@A$, in addition to using "*which*" and its answer type keyword A, we first determine which other wh words to use based on the following mapping of the answer or its concepts: @INFORMATION: *what*; @LOCATION: *where*; @QUANTITY: *how much*. For each statement, its corresponding wh-questions can be derived via subject-auxiliary inversion by replacing its tagged attribute with the corresponding wh word. For each phrase, we generate its predicate for wh-questions and commands, by utilizing the tagged sentence filler algorithm to concatenate the phrase with its corresponding wh word (e.g., *where* _ *the location of* @LANDMARK) and identify verbs (e.g., _ *the location of* @LANDMARK).

Iteration 3. Paraphrasing. This iteration aims to generate the varieties of type keyword A for each attribute $@A$. Specifically, we consider two approaches: (1) we utilize the tagged sentence filler to capture the semantics of a sentence and find the replacement of type keyword A (e.g., *which* _ *is* @LANDMARK *in*). The result is ranked by the frequency of appearance in all questions w.r.t. the same answer. (2) we also find the $N(= 10)$ nearest neighbors of A in the word vector space (GloVe [17]) via cosine similarity, so as to detect the alternative words semantically related to A.

Data-driven Tagged Sentence Filler. As a key data-driven subroutine, this module takes a tagged sentence (sentence with tagged entities) with blanks (number of blanks as parameters) U as the input and outputs the complete tagged sentence with all blanks filled by natural language words, called *filler words*. We first instantiate the tagged entities in U by selecting the most frequent entities in vocabulary V of the language model, which have their concepts matching the tagged entity and is in vocabulary of the pre-trained language model to avoid tackling out of vocabulary problem. Based on the underlying idea that good filler words should be generated repeatedly by different entity instances, we fill up the blanks in all sentences with different instantiated entities and rank the filler words by the frequency of appearances in all instantiated sentences.

3.2. Deep Coupled QA Engine

Thanks to SliQA-I generated large-scale training data, we then design a novel deep learning based QA engine, called *Deep Coupled QA Engine* (in Figure 2). It contains two parts via *additional tightly coupled skip connections*: (1) a bi-directional LSTM based *query-understanding network*; and (2) a LSTM based *answer generator*.

Query-Understanding Network (QUN). Once a question (represented as a word sequence) comes into our QA Engine, it will first pass through an QUN, which is the bottom part under the dotted line in Figure 2. Each word x_i will be read in one by one and transferred into an word vector v_i through an embedding layer. This word vector v_i will be used twice, one is as an input to a bidirectional LSTM, in order to generate the encoded hidden state vector h_i; the other is as part of the input to our answer generator (AG) by skip connection to convey word level information for deriving a better answer.

The bi-directional LSTM reads in the word vector sequence $\mathbf{v} = \{v_1, \cdots, v_n\}$ in two directions, where n is the number of words in a sequence. Forward and backward LSTMs take in the sequence forwardly and backwardly to generate two groups of forward and backward hidden states, $\mathbf{hf} = \{hf_1, \cdots, hf_n\}$ and $\mathbf{hb} = \{hb_1, \cdots, hb_n\}$. The hidden state h_i is a concatenation of hf_i and hb_i, *i.e.*, $h_i = [hf_i, hb_i]$, which will be used as one part of the input to the answer generator (AG). An attention hidden state h_{att} is generated as an augmented signal to capture the contextual information from the word sequence, as $h_{i,att} = \Sigma_{j=1}^{n} \alpha_{i,j} h_j$ where $\alpha_{i,j}$ is a standard weighted coefficient computed by $\alpha_{i,j} = \mathrm{e}^{(q_{i,j})} / \sum_{k=1}^{n} \mathrm{e}^{(q_{i,k})}$, and $q_{i,j}$ is obtained by a separate feed-forward network f_{nn} as $q_{i,j} = f_{nn}(h_j, o_{i-1})$. The concatenation of word vector from v_i, hidden state h_i and attention signal $h_{i,att}$, as $s_i = [v_i, h_i, h_{i,att}]$ will be the output of the query-understanding network (QUN) as well as the input to the following answer generator (AG) network.

Answer Generator (AG). AG aims to derive the answer as a pair of tag and instance. In the question example *'How large is @landmark'* of Figure 2, *'@area: 72,735 square meters'* is used as the answer pair when the entity instance is *'the Louvre'*.

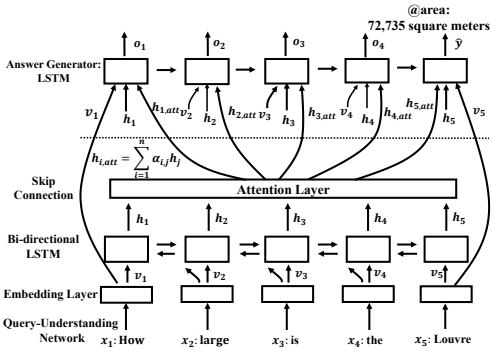

Fig. 2: Deep Coupled QA Engine

AG is a LSTM which takes the output of QUN as its input. Its hidden state o_i is generated as:

$$o_i = g_{rnn}(o_{i-1}, v_i, h_i, h_{i,att}) \quad (1)$$

where $g_{rnn}(\cdot)$ is AG's RNN network. The output $\hat{y}$ of AG is the best answer with the highest probability based on the last state of LSTM:

$$\hat{y} = \arg\max_{y'} P(y' | o_{n-1}, v_n, h_n, h_{n,att}) \quad (2)$$

where n is the length of the word sequence; and y' represents the predicted distribution at the last time step n. The loss function for training our QA engine is the cross entropy for the softmax of $P(y' | o_{n-1}, v_n, h_n, h_{n,att})$.

Remarks: The major improvement of our QA engine is the design of novel skip connection directly from input word vectors to answer generator such that the word level semantics between question and answer can be directly captured.

4. EXPERIMENTAL EVALUATION

We evaluate SliQA-I system from three aspects: (1) evaluate the quality of our SliQA-I generated dataset against the baseline dataset collected from real human developers; (2) evaluate the human workload to generate the dataset; and (3) evaluate the performance of our deep coupled QA engine against the modified state-of-the-art RNN SLU model [15].

4.1. Data Schema and Collection

We collect datasets in three different domains: (a) landmark, (b) restaurant and (c) food. We choose these three domains in the travel user scenario where a user is likely to ask questions about a landmark, local restaurant and food information. In each domain, we determine the set of tagged attributes based on their availability in FreeBase. These tagged attributes are further divided into two types, categorical and scalar. We collect two types of datasets using the same set of entities, attributes and their instances: (1) *SliQA-I questions* generated by our SliQA-I system; and (2) *human questions* generated by crowd sourced human volunteers manually.

4.2. Evaluation of QA Generator

4.2.1. Data Quality Evaluation

Subjective Evaluation. We evaluate the collected question answer pairs by soliciting judgments from another group of crowd sourced raters, following the experimental design in [18]. Each turker rater is presented 10 questions w.r.t. the same answer. Turkers rate each question on a 5 point Likert scale [19] as to whether the question is *natural* and *grammatically correct*. We then asked them to comparatively evaluate 10 questions at a time for their *overall goodness*, also on a 5 point Likert scale. We observe that SliQA-I generated dataset achieves quite promising performance in terms of all three metrics, and with more unique questions.

Table 1: Human Evaluation Results

Datasets	Naturalness	Grammar	Overall	Unique Questions
SliQA-I	3.39	3.71	3.40	4,004
Human	3.43	3.70	3.42	272

Objective Evaluation. We conform to the ultimate goal of our SliQA-I system to objectively evaluate the quality of generated QA dataset based on the performance of QA engine. Again, to maximally mitigate the bias, we test on both state-of-the-art model [15] and our QA engine. As shown in Table 2, in all three domains, both QA engines trained on SliQA-I dataset outperform those trained on human dataset. The accuracy gain is up to 2.14 and 2.32 in SliQA-I and RNN QA engines respectively. This is because SliQA-I dataset has larger varieties of questions but with similar qualities compared with the questions in human dataset.

4.2.2. Human Workload Evaluation

First, we analyze the cognitive load via preliminary qualitative analysis from turker generators who use our SliQA-I system. Specifically, the turkers filled out the survey regarding different types of cognitive load [20]. In terms of *intrinsic cognitive load* about the inherent difficulty level to use SliQA-I system, the turkers concluded SliQA-I as a much easier system to use compared with existing industrial tools which require heavy human workload to manually generate questions as well as academic solution which requires natural language expertise the turkers do not usually possess. For *extraneous cognitive load*, the thumb up/down interface to make human pruning much easier. The turkers can also mark all questions correct by pressing one thumb up/down to select all sentences. At last, the turkers are also satisfied with the reduced *germane cognitive load* due to the iterative pruning design in SliQA-I which dramatically minimizes the whole pruning effort by generating more questions only based on the correct ones in previous iteration. As shown in Figure 3, the time consumption for human pruning is reduced drastically iteration by iteration.

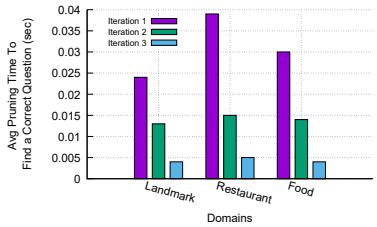

Fig. 3: Time Consumption for Human Pruning

4.3. Evaluation of QA Engine

We train the BiLSTM RNN [15] and our QA engines separately on these three datasets. One may notice in Table 2 that we use a larger size of SliQA-I data than human data for training. This is benefited from the quick generation capability of our SliQA-I generator. To compare it fairly, we also randomly sampled the same number of training data from the SliQA-I generated data as that of the human collected ones. The results are shown in the 5^{th} column in Table 2.

Table 2: QA Engine Performance Comparison

Training data		Human		SliQA-I		
Dataset	QA engine	Size	ACC	ACC	Size	ACC
Landmark	SliQA-I	356	96.21%	**97.56%**	6584	**98.35%**
	RNN	356	95.53%	96.24%	6584	97.85%
Restaurant	SliQA-I	648	97.15%	**97.45%**	11404	**98.78%**
	RNN	648	95.20%	96.90%	11404	97.23%
Food	SliQA-I	628	97.98%	**98.54%**	6036	**98.65%**
	RNN	628	96.29%	97.12%	6036	97.12%

Table 2 report the results. We observe that all QA engines trained on SliQA-I generated data achieve better test accuracy than those trained on human generated baseline data. It can be observed that the augmented dataset does help on improving the model accuracy. However, even we eliminate the performance impact due to more generated training data, our SliQA generated data still gives a better performance using different training engines. This is because SliQA-I generates a dataset which has a better coverage and balance of the varieties of questions. We also observe that our deep coupled QA engine outperforms RNN based engine in all domains, since our skip connection helps to better capture direct relations between question words and its answer.

5. CONCLUSION

This paper takes the first step, called SliQA-I, towards the development of a cold-start end-to-end system (SliQA) for question-answering spoken language interface. SliQA-I, consisting of a novel iterative human-in-the-loop question generator and an enhanced deep coupled QA engine, is designed to build up a QA skill in a new domain that supports answering basic factoid questions, i.e., questions regarding the attributes of an entity. The results show its significant improvement over existing approaches in terms of QA engine and data quality as well as human labor efficiency.

References

[1] "http://www.businessinsider.com/amazon-alexa-how-many-skills-chart-2017-7," .

[2] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer, "Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web – Interoperability, Usability, Applicability*, 2013.

[3] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor, "Freebase: A collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 2008, SIGMOD '08, pp. 1247–1250.

[4] Dominic Seyler, Mohamed Yahya, and Klaus Berberich, "Knowledge questions from knowledge graphs," in *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, 2017, ICTIR '17, pp. 11–18.

[5] Andrew Olney, Arthur C. Graesser, and Natalie K. Person, "Question generation from concept maps," *Dialogue and Discourse*, vol. 3, pp. 75–99, 2012.

[6] Iulian Vlad Serban, Alberto García-Durán, Çaglar Gülçehre, Sungjin Ahn, Sarath Chandar, Aaron C. Courville, and Yoshua Bengio, "Generating factoid questions with recurrent neural networks: The 30m factoid question-answer corpus," *CoRR*, vol. abs/1603.06807, 2016.

[7] Saeedeh Shekarpour, Axel-Cyrille Ngonga Ngomo, and Sören Auer, "Question answering on interlinked data," in *Proceedings of the 22nd International Conference on World Wide Web*, 2013, WWW '13, pp. 1145–1156.

[8] Hannah Bast and Elmar Haussmann, "More accurate question answering on freebase," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015, CIKM '15, pp. 1431–1440.

[9] Kun Xu, Sheng Zhang, Yansong Feng, Songfang Huang, and Dongyan Zhao, "What is the longest river in the usa? semantic parsing for aggregation questions," in *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2015, AAAI'15, pp. 4222–4223.

[10] Wanyun Cui, Yanghua Xiao, and Wei Wang, "Kbqa: An online template based question answering system over freebase," in *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, 2016, IJCAI'16, pp. 4240–4241.

[11] D. A. Ferrucci, "Introduction to "this is watson"," *IBM Journal of Research and Development*, vol. 56, no. 3.4, pp. 1:1–1:15, May 2012.

[12] Daniel Duma and Ewan Klein, "Generating natural language from linked data: Unsupervised template extraction," in *Proceedings of the 10th International Conference on Computational Semantics*, 2013, IWCS '13, pp. 83–94.

[13] Axel-Cyrille Ngonga Ngomo, Lorenz Bühmann, Christina Unger, Jens Lehmann, and Daniel Gerber, "Sorry, i don't speak sparql: Translating sparql queries into natural language," in *Proceedings of the 22nd International Conference on World Wide Web*, 2013, WWW '13, pp. 977–988.

[14] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano, "Template-based question answering over rdf data," in *Proceedings of the 21st International Conference on World Wide Web*, 2012, WWW '12, pp. 639–648.

[15] Bing Liu and Ian Lane, "Attention-based recurrent neural network models for joint intent detection and slot filling," in *17th Annual Conference of the International Speech Communication Association, San Francisco, CA, USA, September 8-12, 2016*, 2016, Interspeech '16, pp. 685–689.

[16] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum, "Yago: A core of semantic knowledge," in *Proceedings of the 16th International Conference on World Wide Web*, 2007, WWW '07, pp. 697–706.

[17] Jeffrey Pennington, Richard Socher, and Christopher Manning, "Glove: Global vectors for word representation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014, EMNLP '14, pp. 1532–1543.

[18] Neha Nayak, Dilek Hakkani-Tür, Marilyn A. Walker, and Larry P. Heck, "To plan or not to plan? discourse planning in slot-value informed sequence to sequence models for language generation," in *18th Annual Conference of the International Speech Communication Association, Stockholm, Sweden, August 20-24, 2017*, 2017, Interspeech '17, pp. 3339–3343.

[19] "https://en.wikipedia.org/wiki/likert_scale," .

[20] John Sweller, Jeroen J. G. Van Merrienboer, and Fred G. W. C. Paas, "Cognitive architecture and instructional design," *Educational Psychology Review*, vol. 10, pp. 251–296, 1998.