Augmenting Question Answering with Natural Language Explanations
CSCI-699 Final Report

Qinyuan Ye
qinyuany@usc.edu

Abstract

Towards building annotation-efficient question answering (QA) systems for real information seeking needs, we propose a framework that efficiently augments training data by leveraging natural language explanations to annotate unanswered questions. Explanations describing how an answer is arrived for one reference QA instance are first parsed into executable rules, and then applied to large-scale unanswered questions (e.g. questions raised in search engines, online forums). Answers with high confidence will be regarded as a training instance and potentially improve QA model performance, especially in low-resource setting. We highlight (1) the generalization ability in rule matching and (2) annotation efficiency in our proposed framework.

Currently, we have collected 200+ explanations for SQuAD instances. The parser can successfully parse and validate 42% of them. We expect the parser to be further improved with the expansion of predicate dictionary. Meanwhile, we have several reasonable observations in our case study of rule hard-matching, such as the trade-off between precision and coverage and variations in acceptable answers.

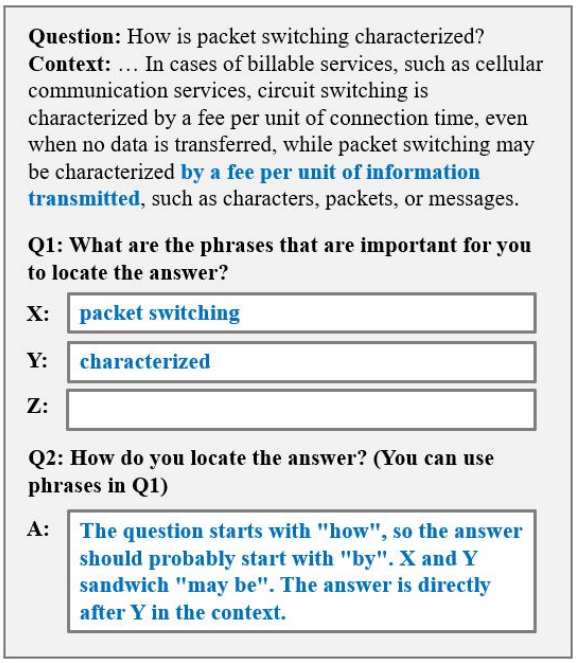

Figure 1: Annotators provide natural language explanations to describe how they arrive at the answer for a labeled QA instance. These explanations are then parsed into structured rules and used to answer similar questions. Pseudo-answered instances will enlarge the train set and allow the downstream QA model capture more information.

1 Introduction

Recent advances in natural language processing claims human-level intelligence in the task of Question Answering (QA) (Joshi et al., 2019; Lan et al., 2019). However, two important factors are often hidden behind this appealing news piece. (1) State-of-the-art results strongly rely on large-scale annotated datasets. It is impractical and time-consuming to collect such dataset for a new domain or a new language to achieve comparable, human-level performance. (2) Questions in these datasets are raised from given context via crowdsourcing, so these questions are collected on purpose, are not "naturally occuring" (Kwiatkowski et al., 2019), and do not reflect real-world needs of seeking information. These two factors imply a huge gap between QA model in research phase and QA model for application.

In this project, we aim to leverage unlabeled data with natural language (NL) explanations to build annotation-efficient, down-to-earth QA systems. Fig. 1 shows an example of NL explanations that we collect. Annotators are shown a reference QA instance and are required to explain how the answer is arrived in a few sentences. Though annotating explanations takes more time, such information can be potentially applied to numerous unanswered questions and create a larger training set, thus improving annotation efficiency. Moreover, these explanations sticks to the reference QA instance, which can provide domain-specific information, and become applicable to new instances within this domain.

The proposed framework is composed of two parts: (1) Rule Parser; (2) Matching Engine. The

rule parser will transform natural language explanations into structured and executable rules. The Matching Engine, which employs a heuristic search strategy and consists of several trainable modules in its evaluation phase, will take full advantage of the parsed rules to answer unanswered questions. One highlight of the Matching Engine is that it is dynamically constructed with the content of each rule, inspired by the notion of neural modular network. This also enables fuzzy matching to improve rule coverage by pseudo-labeling instances that slightly break the rule.

Our framework is model-agnostic, as it serves as data augmentation and provides training instance with high confidence to downstream QA models. Desirably, the performance of downstream model will be improved with large-scale, pseudo-answered instances with high confidence, comparing to the case where the training set only contains a few human-annotated instances. Extensive analysis including annotation efficiency, ablation study on modules, and case study will be conducted to demonstrate the strength of our proposed framework.

2 Related Work

Question Answering Question answering (QA) has long been considered an important benchmark task in natural language understanding (NLU). Recent advances in QA largely benefit from large-scale annotated datasets such as SQuAD (Rajpurkar et al., 2016, 2018) and WikiQA (Yang et al., 2015). Several recent datasets are designed to probe specific abilities of machine comprehension. For example, HotpotQA (Yang et al., 2018) focus on multi-hop reasoning and DROP (Dua et al., 2019) focus on discrete and arithmetic reasoning.

On the other hand, the capability of QA models is strongly enhanced with the introduction of large-scale pre-trained representations such as BERT (Devlin et al., 2019). Later on, SpanBERT (Joshi et al., 2019) extends BERT by masking spans (instead of masking single tokens) and training span boundary representations. Such technique enhance span-related tasks such as QA and co-reference resolution. More recently, ALBERT (Lan et al., 2019) pushed forward the state-of-the-art on GLUE and SQuAD benchmark with fewer parameters than BERT-large. At core of ALBERT is two parameter reduction technique: cross-layer parameter sharing and factorized embedding parameterization. BiDAF (Seo et al., 2016) is a notable model before the introduction of BERT. It employs a hierarchical multi-stage process that incorporates character-level, word-level, contextual information and allows attention flow to obtain a query-aware context representation.

QA for Real Information Need. Most of the existing datasets are collected in a context-oriented manner. That is, annotators are first shown a piece of context (usually a paragraph), and are required to ask questions accordingly. Even though the annotators are encouraged to paraphrase and use their own words, large lexical overlap between the raised questions and the context is observed. Thus the trained model may be merely learning surface patterns, and performance on these datasets does not reflect real comprehension ability. Another stream of work gather QA instances from trivia questions (quiz competitions) (Joshi et al., 2017; Dunn et al., 2017). Such quizzes already provide question and answer, and the dataset is then augmented context using information retrieval tools. One issue that remains with all the datasets mentioned above is they're not "naturally occurring" questions from human that are seeking information. Recent work start to propose datasets that based on real information need, starting with questions that people poses on search engine or web forum. Natural Questions (NQ) (Kwiatkowski et al., 2019) collects questions that users input into Google search engine and pair the question with a top-ranked Wikipedia page. One notable contribution of this work is that is studies the variations in acceptable answers and introduce new metrics to evaluate QA models. There are several domain-specific QA datasets that also reflect real information need, including legal domain (Zhong et al., 2019), biomedical domain (Jin et al., 2019), tech support (Castelli, 2019) and e-commerse (Gupta et al., 2019), etc.

Natural Language Explanation. Srivastava et al. (2017) first introduced the usage of natural language explanation in concept learning. Each statement s in set $\mathcal{S}$ is first parsed into logical form with a CCG parser and acts like a binary feature function $z = f(x) \in \{0, 1\}$. The original representation of the instance, x, is augmented with binary feature outputs z, and is later fed into a classifier. Though only a small number of

instances are required to achieve competitive performance, the task is done in a purely supervised way.

More recently, Hancock et al. (2018) proposed a BABBLELABBLE framework for training classifiers with NL explanations, and succeeded in three relation extraction tasks. BABBLELABBLE abandoned trainable CCG-based parser and adopted a simpler and fixed rule-based parser with no domain-specific predicate dictionary. Another difference is that the z is not used to augment x but is used to pseudo-label x instead. This characteristic is more similar to data programming setting (Ratner et al., 2016, 2017) and enables semi-supervised learning on unlabeled corpus. This piece of work focus on annotation efficiency, instead of pushing forward the state-of-the-art.

Wang et al. (2019) proposed NEXT framework which further extends previous work with modularized labeling functions. That is, the labeling function is constructed with small function blocks following the tree-structure of parsed NL explanations. Each block is parameterized and some of them are trainable. Compared to (Hancock et al., 2018), NEXT enables controllable and trainable fuzzy matching. This helps deal with low-coverage issue of labeling functions and enlarge the size of training set in low-resource setting.

3 Proposed Framework

The proposed framework includes two parts: (1) **Rule Parser** that transforms collected NL explanations to structured and executable form, and (2) **Matching Engine** that search for answers given a rule, a question and some context. In the following, we will first restate the problem, and introduce the parts respectively.

3.1 Problem Statement

Question Answering. Here we describe a general setting in question answering. In the training set, each instance (q, a, C) includes a question q, one valid answer a and a list of context information $C = \{c_j\}$. The valid answer can either be (1) a paragraph (long answer setting in NQ; TechQA); (2) a short span (short answer setting in NQ; SQuAD). In the test set, multiple valid answers may be acceptable to allow variations in answers. The goal of a QA model is to output one valid answer with the input of question q and context C.

$Is	$Left	$Right	$In	$AtLeast	$Contain
$ArgX	$ArgY	$ArgZ	$Question	$Answer	$Context
$Count	$Word	$Sandwich	$StartWith	$EndWith	$Similar
$Person	$Location	$Time	$Number	$Noun	$Adjective

Table 1: Example predicates. Raw explanations are first mapped to a predefined dictionary of predicates.

@AND	@EQUAL	@IS	@BETWEEN
@IN	@STARTSWITH	@NER	@CHUNK
@LEFT	@RIGHT	@WITHIN	@ENDSWITH

Table 2: Example functions. The semantic part of parsing results should be an logical form constructed with functions in a predefined list.

Natural Language Explanation. During annotation, in addition to the answer a, the annotator is required to answer questions with natural language in order to describe how they arrive at answer a. Alternatively, a may be already obtained from other sources and annotators input explanations only. One example is shown in Fig. 1. The sentences in response to Question 2 are denoted with $E = \{e_i\}$. Each sentence e_i can be potentially pared into one rule r_i, and the rule set R is defined as $R = \{r_i\}$.

Data Splitting. We use $\mathcal{S}_a$ to denote the set of (q, C, a, E) quadruples we collected from annotators. $\mathcal{S}_u$ is the set of unanswered question-context pairs (q, C). Note that $|\mathcal{S}_a| \ll |\mathcal{S}_u|$. $\mathcal{R}$ is the set of all rule sets R extracted from explanations with Rule Parser. In addition, $\mathcal{S}_p$ is used to denote the (q, C, a, s) quadruples produced by $\mathcal{R}$ with the matching engine, where s is the confidence score. The set of (q, C) in $\mathcal{S}_p$ forms a subset of $\mathcal{S}_u$.

3.2 Rule Parser

As shown in Fig. 1, annotators are required to input NL explanations $E = \{e_i\}$ for a given reference QA instance. Following previous work (Srivastava et al., 2017), we use Combinatory Categorial Grammar (CCG) based semantic parsing (Zettlemoyer and Collins, 2012) to parse these explanations into executable forms. In brief, words and phrases in each explanation e_i will be first mapped to their predicates (examples in Table 1). After that, a CCG parser will try combining neighboring predicates (e.g. combine "2" and "$Tokens" into @NUM("2", "tokens")) and finally construct the semantics of the whole sentence. An example parse is shown in Fig. 2.

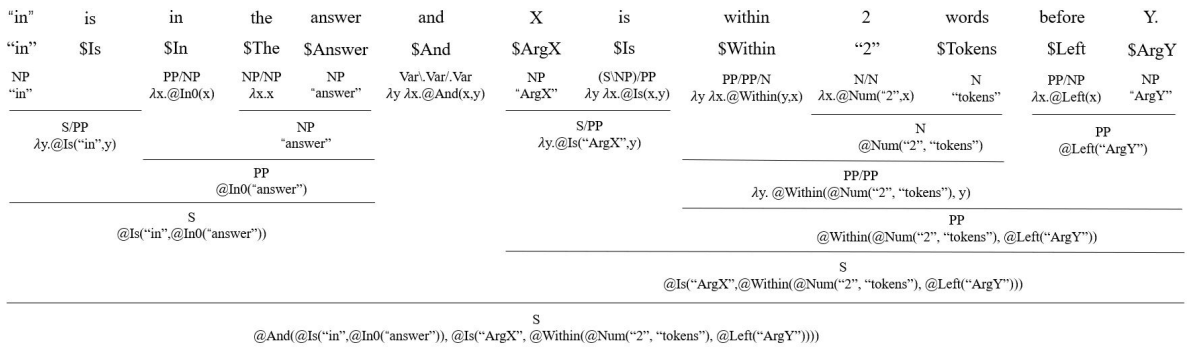

Figure 2: Parsing process of explanation "IN" IS IN THE ANSWER AND X IS WITHIN 2 WORDS BEFORE Y.

Predicate		Lexical	Semantic
$PersonNER	⇒	NP	@NER('PERSON')
$Contains	⇒	S\NP/NP	\ y x.@IS(y, @IN(x))
$Left	⇒	(S\NP)/NP	\y x. @LEFT(y,x)

Table 3: Example lexicons. One lexicon entry maps a predicate to its lexical category and semantic meaning (using the functions in Table 2)

Beam search. Each word or phrase may be mapped to multiple lexicons (e.g. "right" can be describing a position, or expressing "directly"). Moreover, built-in grammar rules in CCG allows abundant linguistic variations. These two factors results in large search space. To deal with this issue, we employ beam search. That is, in the i-th step, $i+1$ lexicons should have been already combined, and a predictive classifier will output a confidence score $p(r)$ for this intermediate parse r on $i+1$ lexicons. We control the beam size to maintain a reasonable search space size.

Predictive Classifier. We follow (Zettlemoyer and Collins, 2007) and construct a feature-based classifier to select promising (intermediate) parses. In a parse r, we count the number of occurrence that predicate f_i is an argument for f_j as g_{ij}. In the example in Fig. 2, the value for feature @WITHIN@NUM will be 1, and the value for feature @IS@NUM will be 0. All g_{ij} are listed to form the feature fector g. The classifier then computes the confidence score $p(r)$ as follows to rank the candidate parses for beam search :

$$p(r) \sim \exp(\theta^\intercal \mathbf{g}) \tag{1}$$

The parameter θ is trained by maximizing the score on parses r that are (1) complete, no variable or missing components in the result semantics; We denote this as $r \in \mathcal{Z}_e$; (2) validated on the reference QA instance; i.e., $r(q,a,C) = \texttt{True}$. Denoting $v(q,a,C) = \mathcal{Z}_e \cap \{r : r(q,a,C) = \texttt{True}\}$, the maximization objective is

$$p(r|e) = \frac{\exp(\theta^\intercal \mathbf{g})}{\sum_{r' \in v(q,a,C)} \exp(\theta^\intercal \mathbf{g}')} \tag{2}$$

3.3 Matching Engine

3.3.1 Answering Strategy

With the natural language explanations collected, we now have a list of variables and a list of answering rules with the help of Rule Parser. We now introduce how matching engine is used to provide pseudo-answers for unanswered question-context pair. The answering strategy can be roughly broken into the following three operations, and done with heuristic beam search method. The process goes in a loop of "fill-evaluate-filter" until all variables are filled. We choose the answer with the highest evaluation score as the pseudo-answer for the instance and add it to $\mathcal{S}_q$.

Fill Variables such as X, Y and Z are defined by the Turkers in Question 1. Additionally, we introduce one variable ANS (the answer) and one variable SENT (the sentence containing the answer). We require each variable (except ANS) to either have a `Chunk` type or `NER` type. We use the constituency parser and NER model in AllenNLP (Gardner et al., 2017) to extract variable candidates for one variable.

Evaluate We evaluate the feasibility for each candidate combination with the parsed logical forms. Details of the modules used in evaluation can be found in the next section. We skip the logical forms that contains pending variables (not assigned with any candidate yet).

Filter After fill and evaluate, each intermediate instance has a score, and we maintain a beam of a

pre-defined size, storing the most promising intermediate results, and start the next iteration in the loop by filling a new variable.

3.3.2 Modules in Evaluation

String Matching In previous work (Wang et al., 2019), string matching module is pre-trained by clustering key phrases in each category. Reading comprehension (RC) is significantly different from classification problems, as RC does not have pre-defined classes or categories. And thus we seek help from pre-trained representations such as BERT, which potentially learned semantic similarities in pre-training. The input to a string matching module is a phrase p of length m and a sentence s of length n. The goal of string matching module is to identify the location of p (or its equivalence) in s. The output will be two n-dimensional vectors, v_b and v_e, in which v_{bi} represent the probability that p (or its equivalence) appear in s with the starting position of i. v_e correspond to ending poisition. This is to ensure the length of p's equivalence will not be constrained. We plan to calculate v_{bi} and v_{ei} in the following way:

$$\begin{aligned} [w_1, w_2, ..., w_m] &= \texttt{BERT}(p) \\ [u_1, u_2, ..., u_n] &= \texttt{BERT}(s) \\ v_{bi} &= \texttt{sim}(w_1, u_i) \\ v_{ei} &= \texttt{sim}(w_m, u_i) \end{aligned} \tag{3}$$

There can be several modifications to the design above. For example, the `sim` function can be any kind of distance metrics, or a learned metric as in Relation Networks (Sung et al., 2018). Also, w_1 and w_m may be replaced with a mean pooling result over $\texttt{BERT}(p)$. We also have an idea in mind to train an attention on top of `BERT` to enable contextualized string matching. We will try out these variations and select the one with best performance.

Location To evaluate p_1 is within d words left/right of p_2 in s, the location module takes the output of $\texttt{String\_Matching}(p_1, s)$ and $\texttt{String\_Matching}(p_2, s)$ and output a scalar score for logical calculation. To evaluate p_0 is between p_1 and p_2 in s, the location module takes the output of $\texttt{String\_Matching}(p_i, s), \quad i = 0, 1, 2$ and output a scalar score. Detailed implementation is still in discussion.

Logical Instead of using soft logic, i.e., $\textsc{And}(p_1, p_2) = p_1 + p_2 - 1$, we plan to use graphical model to make sure the scale is not influenced by the number of constraints. The intuition is that two conditions with 70% confidence may not be as good as three conditions with 60% confidence. If there are n rules in the current answering function, m can be evaluated with the filled variables, and $S = \{s_1, s_2, ..., s_m\}$ shows the evaluation score of the m rules, this module outputs a score $s_0 = f(m, n, S)$ as the confidence of the current state. Note that we can always transform the parsed rule so that logic operation is always the root of the tree.

Dataset	#Train	#Dev	#Test
SQuAD 2.0	130,319	11,873	8,862
Natural Questions	307,373	7,830	7,842
TechQA	600	310	490

Table 4: Statistics of Datasets.

4 Experiment Setup

4.1 Datasets

We plan to do experiments on the following three QA benchmark datasets. The statistics of these datasets are summarized in Table 4. Among these three datasets, we believe NQ and TechQA relate to our problem setting more closely. Meanwhile, SQuAD 2.0 as a widely-used dataset also serves as a testbed to simulate the practical situation and evaluate the efficiency of our method.

(1) **SQuAD 2.0** (Rajpurkar et al., 2018) is a widely-acknowledged QA dataset. Annotators are first shown a paragraph from wikipedia, and then asked to raise several questions and provide answers. Due to the nature of this dataset collection process, the question and context tend to have large lexical overlap.

(2) **Natural Questions (NQ)** (Kwiatkowski et al., 2019) consists real anonymized, aggregated queries issued in Google search engine, and reflect real information seeking needs of human. The annotators are shown the question and a top-matching wikipedia page, and are required to provide long answers (a paragraph) and short answers (a span; yes/no). Questions requiring multiple paragraphs to answer will be marked as unanswerable in long answer setting.

(3) **TechQA** (Castelli, 2019) is from technical support domain and consists of questions posed by real users in IBM Develop Forum. These questions are then carefully paired with solutions in

What is	What was	How many	When did
What year	What are	What does	When was
What type	Who was	What do	How much

Table 5: Example Question Heads. Top 40 question heads cover more than half of the questions.

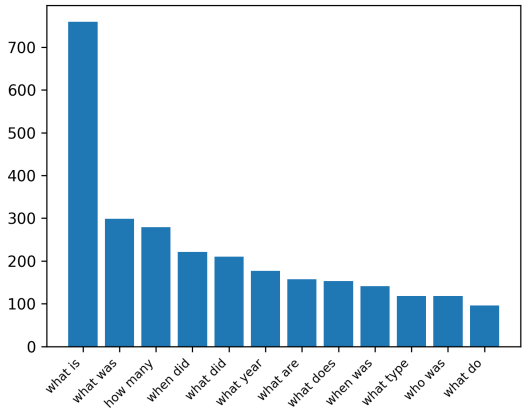

Figure 3: Question head distribution (Top 12)

IBM Technote (a user manual for technical issues) by domain experts.

4.2 Explanation Collection

We first ensure the instances sent for annotation are typical and can represent the commonly-raised questions. We use some heuristics to roughly classify the questions with "question heads" (question word such as "what", plus one word after it). Examples of question heads are shown in Table 5. In a 5% split of SQuAD, we found 1899 question heads, and the top 40 question heads takes up 53.6% of the split. This shows a majority of questions are raised in similar ways; and can potentially be answered in similar ways. We randomly sample 6 questions for each question head. This results in 240 QA instances sent for annotation.

We collect explanations with Amazon Mechanical Turk. As shown in Fig.1[1], turkers are required to answer two questions about one QA instance. The answer in the context is highlighted in red for convenience. Turkers are rewarded with $1 for each accepted responses. As of now, we have collected 215 accepted explanations. We will adjust the annotation interface and collect explanations for NQ and TechQA dataset.

[1]Fig. 1 is a simplified version of annotation interface. Turkers are shown detailed examples and suggests expressions in the actual interface.

Question	How is packet switching characterized?
X	packet switching
Y	characterized
Explanation	The question starts with "how is", so the answer probably starts with "by". X is before Y. The answer directly follows Y.

Table 6: QA instance used in Case 1.

Implied	(A)	@Is('ANS', @Chunk('P'))
Explained	(B)	@StartsWith('Question','how is')
	(C)	@StartsWith('Answer','by')
	(D)	@Is('X',@Left('Y'))
	(E)	@Is(Answer,@Direct(@Right('Y')))

Table 7: Logical Form from the example in Case 1.

Constraint	Matched	Correct	Precision
All	6	5	83.33%
-(C)	16	12	75.00%
-(D)	18	11	61.11%
-(E)	17	11	64.71%

Table 8: Hard matching result. "-(C)" means the constraint (C) in Table 7 is dropped in matching. Simple dropping strategy can extend the coverage of rules, with acceptable sacrifice in precision.

5 Preliminary Results

All results are preliminary. We're still in the process of expanding the lexicon dictionary. The results will be improved when more NL explanations becomes parse-able. Currently, 42.92% of collected explanations can be parsed and successfully validated.

5.1 Case Study with Hard-matching

Case 1: Precision and Coverage Trade-off. Similar to those labeling function in (Hancock et al., 2018) and (Wang et al., 2019), explanations tend to over-specific, which results in high precision but low coverage. Dropping one of the constraint can be considered as a straightforward way to soften the rule. Our results in Table 8 shows a larger coverage with acceptable accuracy.

When enforcing hard matching, we found 6 matches, 5 of which are acceptable in human examination. The answers provided in SQuAD may be slightly different from the hard-matched result because the varying length of span or whether prepositions such as "in" and "by" are included, which we believe are trivial.

We also tested a simple dropping strategy in rule matching. For example, if we do not require the answer to be directly following variable Y, 11 instances can be matched and answered in the whole SQuAD training set, 64.71% of the answers are

Question	Where do safari hunters usually stay?
X	safari hunters
Y	stay
Explanation	The question starts with "where", so the answer should be a place. "in" is in the answer. X is within 2 words before Y. The answer is directly after Y.
Question	Where does 112th Street start?
Answer (provided)	Morningside Heights
Answer (matched)	in Morningside Heights
Question	Where does digestion begin?
Answer (provided)	in the mouth with the secretion of saliva and its digestive enzymes
Answer (matched)	in the mouth

Table 9: Example hard match result from one rule. We observe that one question may be correctly answered in many ways. Small variations such as including "in" or dropping unnecesary details can also result in correct answers. The matched results are still useful in such cases.

correct. Existing QA models tend to be poor at few-shot learning, and thus increasing the scale of training instances while sacrificing precision can still be helpful.

Case 2: Studying variations in answer. As shown in Table 9, the answers provided in SQuAD is not the only correct answer. In our human examination, we consider an answer with one more preposition (e.g. "in", "by") as correct. Sometimes a span shorter than the provided answer is also sufficient, and we also mark this as correct. This validates the claim in (Kwiatkowski et al., 2019) that there is variability in answers, and evaluation metrics other than F1 and exact match may be needed. The authors address this problems with a 5-way, robust evaluation metric.

6 Plans for Future Research

6.1 Compared Methods

We will compare our models with existing supervised and semi-supervised QA models.

Hard-matching. We directly evaluate test set by hard-matching the rules in $\mathcal{R}$.

Supervised. The (q, C, A) triple in S_a is used as training set to supervised learning models. Compared methods include (1) ALBERT (Lan et al., 2019), the most recent published state-of-the-art model on SQuAD 2.0 as of December 12, 2019; (2) BiDAF (Seo et al., 2016), a classical model that incorporates character-level, word-level, contextual information and allows attention flow; (3) BiDAF++ (Yang et al., 2018), which is an enhanced version of (Seo et al., 2016) by incorporating character-level models, self-attention and bi-attention.

Semi-supervised. The (q, C, A) triple in S_a and the whole unanswered set S_u is used as training data. Compared methods include (1) GDAN (Yang et al., 2017); (2) Cloze-style Pretraining (Dhingra et al., 2018).

6.2 In-depth Analysis

Annotation Efficiency. To demonstrate that introducing natural language explanation is efficient compared to only providing an answer, we plan to do controlled experiments on the number of instances used and the annotation time used. For example, we check the number of instances used to train a supervised model that achieves the comparable score with our proposed method, and calculate the time used for annotation. The desired result will be our method saves annotation efforts.

Ablation Study. We plan to do ablation study on the fuzzy matching ability of `String_Matching`, `Location` and `Logical` module.

Case Study. We plan to manually examine pseudo answers with high confidence. This will certify the interpretability and generalization ability of our proposed framework.

7 Conclusion

In this report we explore and discuss a potential way to augment question answering dataset with natural language explanations. The proposed framework follows previous work in relation extraction and aspect-based sentiment analysis (Wang et al., 2019). Several challenges are raised in extending previous work to QA, including finding the anchor word, dealing with multiple sentences, and dealing with span prediction instead of traditional classification. There are still lots of work left to be done in this project, including (1) further expanding the predicate and lexicon dictionary to achieve better parsing results; (2) discussing the mathematical details for modules used in evaluation; try out different implementations and select the most suitable one; (3) crowd-sourcing high-quality explanations. We believe studying annotation efficiency for question answering has great value in bridging the gap between research and real-world applications.

References

Vittorio Castelli. 2019. The techqa dataset. *arXiv preprint arXiv:1911.02984*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Bhuwan Dhingra, Danish Danish, and Dheeraj Rajagopal. 2018. Simple and effective semi-supervised question answering. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 582–587.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proc. of NAACL*.

Matthew Dunn, Levent Sagun, Mike Higgins, V Ugur Guney, Volkan Cirik, and Kyunghyun Cho. 2017. Searchqa: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

Mansi Gupta, Nitish Kulkarni, Raghuveer Chanda, Anirudha Rayasam, and Zachary C Lipton. 2019. Amazonqa: a review-based question answering task. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 4996–5002. AAAI Press.

Braden Hancock, Martin Bringmann, Paroma Varma, Percy Liang, Stephanie Wang, and Christopher Ré. 2018. Training classifiers with natural language explanations. In *Proceedings of the conference. Association for Computational Linguistics. Meeting*, volume 2018, page 1884. NIH Public Access.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. 2019. Pubmedqa: A dataset for biomedical research question answering. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2567–2577.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. 2019. Spanbert: Improving pre-training by representing and predicting spans. *arXiv preprint arXiv:1907.10529*.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. 2017. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. *Proceedings of the VLDB Endowment*, 11(3):269–282.

Alexander J Ratner, Christopher M De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. 2016. Data programming: Creating large training sets, quickly. In *Advances in neural information processing systems*, pages 3567–3575.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*.

Shashank Srivastava, Igor Labutov, and Tom Mitchell. 2017. Joint concept learning and semantic parsing from natural language explanations. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 1527–1536.

Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. 2018. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208.

Ziqi Wang, Yujia Qin, Wenxuan Zhou, Jun Yan, Qinyuan Ye, Leonardo Neves, Zhiyuan Liu, and

Xiang Ren. 2019. Learning to annotate: Modularizing data augmentation for textclassifiers with natural language explanations. *arXiv preprint arXiv:1911.01352*.

Yi Yang, Wen-tau Yih, and Christopher Meek. 2015. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2013–2018.

Zhilin Yang, Junjie Hu, Ruslan Salakhutdinov, and William Cohen. 2017. Semi-supervised qa with generative domain-adaptive nets. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1040–1050.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687.

Luke S Zettlemoyer and Michael Collins. 2012. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*.

Haoxi Zhong, Chaojun Xiao, Cunchao Tu, Tianyang Zhang, Zhiyuan Liu, and Maosong Sun. 2019. Jec-qa: A legal-domain question answering dataset. *arXiv preprint arXiv:1911.12011*.