HAS-QA: Hierarchical Answer Spans Model for Open-domain Question Answering

Liang Pang†, Yanyan Lan†∗, Jiafeng Guo†, Jun Xu†, Lixin Su† and **Xueqi Cheng†**
†CAS Key Laboratory of Network Data Science and Technology,
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
†University of Chinese Academy of Sciences, Beijing, China
∗Department of Statistics, University of California, Berkeley
{pangliang,lanyanyan,guojiafeng,sulixin,cxq}@ict.ac.cn, junxu@ruc.edu.cn

Abstract

This paper is concerned with open-domain question answering (i.e., OpenQA). Recently, some works have viewed this problem as a reading comprehension (RC) task, and directly applied successful RC models to it. However, the performances of such models are not so good as that in the RC task. In our opinion, the perspective of RC ignores three characteristics in OpenQA task: 1) many paragraphs without the answer span are included in the data collection; 2) multiple answer spans may exist within one given paragraph; 3) the end position of an answer span is dependent with the start position. In this paper, we first propose a new probabilistic formulation of OpenQA, based on a three-level hierarchical structure, i.e., the question level, the paragraph level and the answer span level. Then a Hierarchical Answer Spans Model (HAS-QA) is designed to capture each probability. HAS-QA has the ability to tackle the above three problems, and experiments on public OpenQA datasets show that it significantly outperforms traditional RC baselines and recent OpenQA baselines.

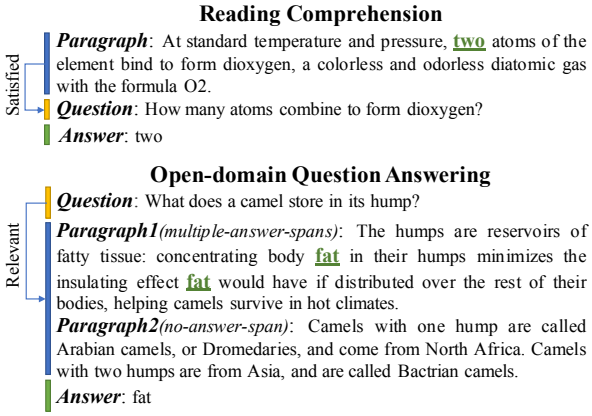


Figure 1: Examples of RC task and OpenQA task.

1 Introduction

Open-domain question answering (OpenQA) aims to seek answers for a broad range of questions from a large knowledge sources, e.g., structured knowledge bases (Berant et al. 2013; Mou et al. 2017) and unstructured documents from search engine (Ferrucci et al. 2010). In this paper we focus on the OpenQA task with the unstructured knowledge sources retrieved by search engine.

Inspired by the reading comprehension (RC) task flourishing in the area of natural language processing (Wang and Jiang 2016; Seo et al. 2016; Xiong, Zhong, and Socher 2016), some recent works have viewed OpenQA as an RC task, and directly applied the existing RC models to it (Chen et al. 2017; Joshi et al. 2017; Wang and Jiang 2016; Clark and Gardner 2018). However, these RC models do not well fit for the OpenQA task.

Firstly, they directly omit the paragraphs without answer string[1]. RC task assumes that the given paragraph contains the answer string (Figure 1 top), however, it is not valid for the OpenQA task (Figure 1 bottom). That's because the paragraphs to provide answer for an OpenQA question is collected from a search engine, where each retrieved paragraph is merely relevant to the question. Therefore, it contains many paragraphs without answer string, for instance, in Figure 1 Paragraph2. When applying RC models to OpenQA task, we have to omit these paragraphs in the training phase. However, during the inference phase, when model meets one paragraph without answer string, it will pick out a text span as an answer span with high confidence, since RC model has no evidence to justify whether a paragraph contains the answer string.

Secondly, they only consider the first answer span in the paragraph, but omit the remaining rich multiple answer spans. In RC task, the answer and its positions in the paragraph are provided by the annotator in the training data. Therefore RC models only need to consider the unique answer span, e.g., in SQuAD (Rajpurkar et al. 2016). However, the OpenQA task only provides the answer string as the ground-truth. Therefore, multiple answer spans are detected in the given paragraph, which cannot be considered by the traditional RC models. Take Figure 1 as an example, all text spans contain 'fat' are treated as answer span, so we detect two answer spans in Paragraph1.

Thirdly, they assume that the start position and end posi-

[1]The *answer string* is a piece of text that can answer the question. If the answer string is obtained in a paragraph as a consecutive text, we call it the *answer span*.

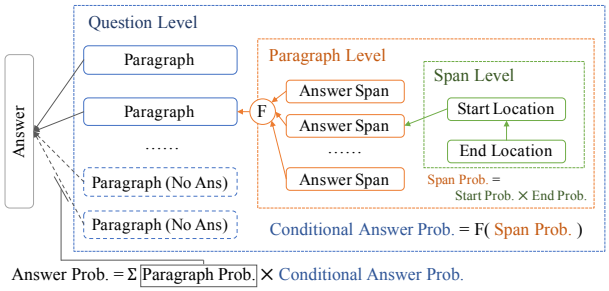


Figure 2: The three hierarchical levels of OpenQA task.

tion of an answer span is independent. However, the end position is evidently related with the start position, especially when there are multiple answer spans in a paragraph. Therefore, it may introduce some problems when using such independence assumption. For example, the detected end position may correspond to another answer span, rather than the answer span located by the start position. In Figure 1 Paragraph1, 'fat in their $\cdots$ insulating effect fat' has a high confidence to be an answer span under independence assumption.

In this paper, we propose a Hierarchical Answer Span Model, named HAS-QA, based on a new three-level probabilistic formulation of OpenQA task, as shown in Figure 2.

At the question level, the conditional probability of the answer string given a question and a collection of paragraphs, named *answer probability*, is defined as the product of the *paragraph probability* and *conditional answer probability*, based on the law of total probability.

At the paragraph level, *paragraph probability* is defined as the degree to which a paragraph can answer the question. This probability is used to measure the quality of a paragraph and targeted to tackle the first problem mentioned, i.e. identify the useless paragraphs. For calculation, we first apply bidirectional GRU and an attention mechanism on the question aware context embedding to obtain a score. Then, we normalize the scores across the multiple paragraphs. In the training phase, we adopt a negative sampling strategy for optimization. *Conditional answer probability* is the conditional probability that a text string is the answer given the paragraph. Considering multiple answer spans in a paragraph, the *conditional answer probability* can be further represented as the aggregation of several *span probability*, defined later. In this paper, four types of functions, i.e. HEAD, RAND, MAX and SUM, are used for aggregation.

At the span level, *span probability* represents the probability that a text span in a paragraph is the answer span. Similarly to previous work (Wang and Jiang 2016), *span probability* can be computed as the product of two location probability, i.e., *location start probability* and *location end probability*. Then a conditional pointer network is proposed to model the probabilistic dependences between the start and end positions, by making generation of end position depended on the start position directly, rather than internal representation of start position (Vinyals, Fortunato, and Jaitly 2015).

The contributions of this paper include:

1) a probabilistic formulation of the OpenQA task, based on the a three-level hierarchical structure, i.e. the question level, the paragraph level and the answer span level;

2) the proposal of an end-to-end HAS-QA model to implement the three-level probabilistic formulation of OpenQA task (Section 4), which tackles the three problems of direct applying existing RC models to OpenQA;

3) extensive experiments on QuasarT, TriviaQA and SearchQA datasets, which show that HAS-QA outperforms traditional RC baselines and recent OpenQA baselines.

2 Related Works

Research in reading comprehension grows rapidly, and many successful RC models have been proposed (Dhingra et al. 2017; Seo et al. 2016; Wang and Jiang 2016) in this area. Recently, some works have treated OpenQA task as an RC task and directly applied existing RC models. In this section, we first review the approach of typical RC models, then introduce some recent OpenQA models which are directly based on the RC approach.

RC models typically have two components: context encoder and answer decoder. Context encoder is used to obtain the embeddings of questions, paragraphs and their interactions. Most of recent works are based on the attention mechanism and its extensions. The efficient way is to treat the question as a key to attention paragraph (Wang and Jiang 2016; Chen et al. 2017). Adding the attention from paragraph to question (Seo et al. 2016; Xiong, Zhong, and Socher 2016), enriches the representations of context encoder. Some works (Wang et al. 2017; Pan et al. 2017; Clark and Gardner 2018) find that self-attention is useful for RC task. Answer decoder aims to generate answer string based on the context embeddings. There exist two sorts of approaches, generate answer based on the entail word vocabulary (Tan et al. 2018) and retrieve answer from the current paragraph. Almost all works in RC task choose the retrieval-based method. Some of them use two independently position classifiers (Chen et al. 2017; Weissenborn, Wiese, and Seiffe 2017), the others use the pointer networks (Wang and Jiang 2016; Seo et al. 2016; Wang et al. 2017; Pan et al. 2017). An answer length limitation is applied in these models, i.e. omit the text span longer than 8. We find that relaxing length constrain leads to performance drop.

Some recent works in OpenQA research directly introduce RC model to build a pure data driven pipline. DrQA (Chen et al. 2017) is the earliest work that applies RC model in OpenQA task. However, its RC model is trained using typical RC dataset SQuAD (Rajpurkar et al. 2016), which turns to be over-confidence about its predicted results even if the candidate paragraphs contain no answer span. R^3 (Wang et al. 2018) introduces a ranker model to rerank the original paragraph list, so as to improve the input quality of the following RC model. The training data of the RC model is solely limited to the paragraphs containing the answer span and the first appeared answer span location is chosen as the ground truth. Shared-Norm (Clark and Gardner 2018) applied a shared-norm trick which considers paragraphs without answer span in training RC models. The

trained RC model turns to be robust for the useless paragraphs and generates the lower span scores for them. However, it assumes that the start and the end positions of an answer span are independent, which is not suitable for modeling multiple answer spans in one paragraph.

Therefore, we realize that the existing OpenQA models rarely consider the differences between RC and OpenQA task. In this paper, we directly model the OpenQA task based on a probabilistic formulation, in order to identify the useless paragraphs and utilize the multiple answer spans.

3 Probabilistic Views of OpenQA

In OpenQA task, the question Q and its answer string A are given. Entering question Q into a search engine, top K relevant paragraphs are returned, denote as a list $\mathbf{P} = [P_1, \ldots, P_K]$. The target of OpenQA is to find the maximum probability of $P(A|Q, \mathbf{P})$, named *answer probability* for short. We can see the following three characteristics of OpenQA:

1) we cannot guarantee that paragraph retrieved by search engine contains the answer span for the question, so the paragraphs without answer span have to be deleted when using the above RC models. However, these paragraphs are useful for distinguishing the quality of paragraphs in training. More importantly, the quality of a paragraph plays an important role in determining the *answer probability* in the inference phase. It is clear that directly applying RC models fails to meet this requirement.

2) only answer string is provided, while the location of the answer string is unknown. That means there may be many answer spans in the paragraph. It is well known that traditional RC models are only valid for a single answer span. To tackle this problem, the authors of (Joshi et al. 2017) propose a distantly supervised method to use the *first* exact match location of answer string in the paragraph as the ground-truth answer span. However, this method omit the valuable multiple answer spans information, which may be important for the calculation of the *answer probability*.

3) the start and end positions are coupled together to determine a specific answer span, since there may be multiple answer spans. However, existing RC models usually assume that the start and end positions are independent. That's because there is only one answer span in the RC scenario. This may introduce serious problem in the OpenQA task. For example, if we do not consider the relations between the start and end position, the end position may be another answer span's end position, instead of the one determined by the start position. Therefore, it is not appropriate to assume independence between start and end positions.

In this paper, we propose to tackle the above three problems. Firstly, according to the law of total probability, the *answer probability* can be rewritten as the following form.

$$P(A|Q, \mathbf{P}) = \sum_{i=1}^{K} P(P_i|Q, \mathbf{P}) P(A|Q, P_i). \quad (1)$$

We name $P(P_i|Q, \mathbf{P})$ and $P(A|Q, P_i)$ as the *paragraph probability* and *conditional answer probability*, respectively. We can see that the *paragraph probability* measures the quality of paragraph P_i across the list $\mathbf{P}$, while the *conditional answer probability* measures the probability that string A is an answer string given paragraph P_i.

The *conditional answer probability* can be treated as a function of multiple *span probabilities* $\{P(L_j(A)|Q, P_i)\}_j$, as shown in Eq 2.

$$P(A|Q, P_i) := \mathcal{F}(\{P(L_j(A)|Q, P_i)\}_j), \quad j \in [1, |\mathcal{L}(A, P_i)|], \quad (2)$$

where the aggregation function $\mathcal{F}$ treats a list of spans $\mathcal{L}(A, P_i)$ as input, and $|\mathcal{L}(A, P_i)|$ denotes the number of the text spans contain the string A. A proper aggregation function makes use of all the answer spans information in OpenQA task. Previous work (Joshi et al. 2017) can be treated as a special case, which uses a function of selecting first match span as the aggregation function $\mathcal{F}$.

The *span probability* $P(L_j(A)|Q, P_i)$ represents the probability that a text span $L_j(A)$ in the paragraph P_i is an answer span. We further decompose it into the product of *location start probability* $P(L_j^s(A)|Q, P_i)$ and *location end probability* $P(L_j^e(A)|Q, P_i, L_j^s(A))$, shown in Eq 3.

$$\begin{aligned} P(L_j(A)|Q, P_i) =& P(L_j^s(A)|Q, P_i) \\ & \cdot P(L_j^e(A)|Q, P_i, L_j^s(A)). \end{aligned} \quad (3)$$

Some previous work such as DrQA (Chen et al. 2017) treats them as the two independently position classification tasks, thus $L^s(A)$ and $L^e(A)$ are modeled by two different functions. Match-LSTM (Wang and Jiang 2016) treats them as the pointer networks (Vinyals, Fortunato, and Jaitly 2015). The difference is that $L^e(A)$ is the function of the hidden state of $L^s(A)$, denote as $\mathbf{M^s}$. However, $L^s(A)$ and $L^e(A)$ are still independent in probabilistic view, because $L^e(A)$ depends on the hidden state $\mathbf{M^s}$, not the start position $L^s(A)$. In this paper, the span positions $L_j^s(A)$ and $L_j^e(A)$ are determined by the question Q and the paragraph P_i. Specially, end position $L_j^e(A)$ is also conditional on start position $L_j^s(A)$ directly. With this conditional probability, we can naturally remove the answer length limitation.

With above formulation, we find that RC task is a special case of OpenQA task, where we set the number of paragraph K to 1, set the *paragraph probability* to constant number 1, treat $P(A|Q, P) = P(L(A)|Q, P)$, $P(L(A)|Q, P) = P(L^s(A)|Q, P)P(L^e(A)|Q, P)$, where P is the idealized paragraph that contain the answer string A, and the right position $L(A)$ is also known.

4 HAS-QA Model

In this section, we propose a Hierarchical Answer Span Model (HAS-QA) for OpenQA task, based on the probabilistic view of OpenQA in Section 3. HAS-QA has four components: question aware context encoder, conditional span predictor, multiple spans aggregator and paragraph quality estimator. We will introduce them one by one.

4.1 Question Aware Context Encoder

The question aware context embeddings $\mathbf{C}$ is generated by the context encoder, while HAS-QA do not limit the use

of context encoder. We choose a simple but efficient context encoder in this paper. It takes advantage of previous works (Clark and Gardner 2018; Wang and Jiang 2016), which contains the character-level embedding enhancement, the bi-directional attention mechanism (Seo et al. 2016) and the self-attention mechanism (Wang et al. 2017). We briefly describe the process below [2].

Word Embeddings: use size 300 pre-trained GloVe (Pennington, Socher, and Manning 2014) word embeddings.

Char Embeddings: encode characters in size 20, which are learnable. Then obtain the embedding of each word by convolutional layer and max pooling layer.

Context Embeddings: concatenate word embeddings and char embeddings, and apply bi-directional GRU (Cho et al. 2014) to obtain the context embeddings. Both question and paragraph get their own context embeddings.

Question Aware Context Embeddings: use bi-directional attention mechanism from the BiDAF (Seo et al. 2016) to build question aware context embeddings. Additionally, we subsequently apply a layer of self-attention to get the final question aware context embeddings.

After the processes above, we get the final question aware context embeddings, denoted $\mathbf{C} \in \mathbb{R}^{n \times r}$, where n is the length of the paragraph and r is size of the embedding.

4.2 Conditional Span Predictor

Conditional span predictor defines the *span probability* for each text span in a paragraph using a conditional pointer network.

We first review the answer decoder in traditional RC models. It mainly has two types: two independently position classifiers (IndCls) and the pointer networks (PtrNet). Both of these approaches generate a distribution of start position $\mathbf{p^s} \in \mathbb{R}^n$ and a distribution of end position $\mathbf{p^e} \in \mathbb{R}^n$, where n is the length of the paragraph. Starting from the context embeddings $\mathbf{C}$, two intermedia representations $\mathbf{M^s} \in \mathbb{R}^{n \times 2d}$ and $\mathbf{M^e} \in \mathbb{R}^{n \times 2d}$ are generated using two bidirectional GRUs with the output dimension d.

$$\mathbf{M^s} = \mathrm{BiGRU}(\mathbf{C}) \tag{4}$$

$$\text{IndCls: } \mathbf{M^e} = \mathrm{BiGRU}(\mathbf{C}), \tag{5}$$

$$\text{PtrNet: } \mathbf{M^e} = \mathrm{BiGRU}([\mathbf{C}, \mathbf{M^s}]). \tag{6}$$

Then an additional Softmax function is used to generate the final positional distributions,

$$\begin{aligned} \mathbf{p^s} &= \mathrm{softmax}(\mathbf{M^s} w_s), \\ \mathbf{p^e} &= \mathrm{softmax}(\mathbf{M^e} w_e). \end{aligned} \tag{7}$$

where $w_s, w_e \in \mathbb{R}^{2d}$ denotes the linear transformation parameters.

As mentioned in Section 3, IndCls and PtrNet both treat start and end position as probabilistic independent. Given the independent start and end positions can not distinguish the different answer spans in a paragraph properly, so it is necessary to build a conditional model for them. Therefore, we proposed a conditional pointer network which directly feed the start position to the process of generating the end position:

$$\begin{aligned} \mathbf{M_j^e} &= \mathrm{BiGRU}([\mathbf{C}, \mathbf{M^s}, \mathrm{OneHot}(L_j^s)]), \\ \mathbf{p_j^e} &= \mathrm{softmax}(\mathbf{M_j^e} w_e), \end{aligned} \tag{8}$$

where L_j^s denotes the start position selected from the start positional distribution $\mathbf{p^s}$ and $\mathrm{OneHot}(\cdot)$ denotes the transformation from a position index to an one-hot vector.

In the training phase, we are given the start and end positions of each answer span, denote as L_j^s and L_j^e. The *span probability* is:

$$P(L_j(A)|Q, P_i) = s_j = \mathbf{p^s}[L_j^s] \cdot \mathbf{p_j^e}[L_j^e]. \tag{9}$$

In the inference phase, we first select the start position L_j^s from the start distribution $\mathbf{p^s}$. Then we yield its corresponding end distribution $\mathbf{p_j^e}$ using Eq 8, and select the end position L_j^e from it. Finally, we get the *span probability* using Eq 9.

4.3 Multiple Spans Aggregator

Multiple span aggregator is used to build the relations among multiple answer spans and outputs the *conditional answer probability*. In this paper, we design four types of aggregation functions $\mathcal{F}$:

$$\begin{aligned} &\text{HEAD: } P(A|Q, P_i) = s_1 \\ &\text{RAND: } P(A|Q, P_i) = \mathrm{Random}(s_j) \\ &\text{MAX: } \ P(A|Q, P_i) = \max_j(s_j) \\ &\text{SUM: } \ P(A|Q, P_i) = \sum\nolimits_j (s_j) \end{aligned} \tag{10}$$

where s_j denotes the *span probability* defined in Eq 9, s_1 denotes the first match answer span and Random denotes a stochastic function for randomly choosing an answer span.

Different aggregation functions represent different assumptions about the distribution of the oracle answer spans in a paragraph. The oracle answer span represents the answer of the question that can be merely determined by its context, e.g. in Figure 1, the first answer span 'fat' is the oracle answer span, while the second one is not, because we could retrieval the answer directly, if we have read 'concentrating body fat in their humps'.

HEAD operation simply chooses the first match *span probability* as the *conditional answer probability*, which simulates the answer preprocessing in previous works (Wang et al. 2018; Joshi et al. 2017). This function only encourages the first match answer span as the oracle, while punishes the others. It can be merely worked in a paragraph with definition, such as first paragraph in WikiPedia.

RAND operation randomly chooses a *span probability* as the *conditional answer probability*. This function assumes that all answer spans are equally important, and must be treated as oracle. However, balancing the probabilities of answer spans is hard. It can be used in paraphrasing answer spans appear in a list.

MAX operation chooses the maximum *span probability* as the *conditional answer probability*. This function assumes that only one answer span is the oracle. It can be used in a

[2] For more detailed computational steps, see reference paper (Clark and Gardner 2018).

noisy paragraph, especially for those retrieved by a search engine.

SUM operation sums all the *span probabilities* as the *conditional answer probability*. This function assumes that one or more answer spans are the oracle. It can be used in a broad range of scenarios, for its relatively weak assumption.

In the training phase, all annotated answer spans contain the same answer string A, we directly apply the Eq 10 to obtain the *conditional answer probability* in paragraph level.

In the inference phase, we treat the top K *span probabilities* s_j as the input of the aggregation function. However, we have to check all possible start and end positions to get the precise top K *span probabilities*. Instead, we use a beam search strategy (Sutskever, Vinyals, and Le 2014) which only consider the top K_1 start positions and the top K_2 end positions, where $K_1 K_2 \geq K$. Different *span probabilities* s_j represent variance answer strings A_t. Following the definition in Eq 10, we group them by different answer strings respectively.

4.4 Paragraph Quality Estimator

Paragraph quality estimator takes the useless paragraphs into consideration, which implements the *paragraph probability* $P(P_i|Q, \mathbf{P})$ directly.

Firstly, we use an attention-based network to generate a quality score, denotes as $\hat{q}_i$, in order to measure the quality of the given paragraph P_i.

$$\begin{aligned} \mathbf{M^c} &= \text{BiGRU}(\mathbf{C}), \\ \hat{q}_i &= (\mathbf{M^c}^\top \cdot \mathbf{p^s}) \cdot w_c. \end{aligned} \tag{11}$$

where $\mathbf{M^c} \in \mathbb{R}^{n \times 2d}$ is the intermedia representation obtained by applying bidirectional GRU on the context embedding $\mathbf{C}$. Then, let start distribution $\mathbf{p^s} \in \mathbb{R}^n$ as a key to attention $\mathbf{M^c}$ and transform it to 1-d value using weight $w_c \in \mathbb{R}^{2d}$. Finally, we get the quality score $\hat{q}_i$. *Paragraph probabilities* $P(P_i|Q, \mathbf{P})$ are generated by normalizing across $\mathbf{P}$,

$$P(P_i|Q, \mathbf{P}) = q_i = \frac{\exp(\hat{q}_i)}{\sum_{P_j \in \mathbf{P}} \exp(\hat{q}_j)}. \tag{12}$$

In the training phase, we conduct a negative sampling strategy with one negative sample, for efficient training. Thus a pair of paragraphs, P^+ as positive and P^- as negative, are used to approximate $q^+ \approx P(P^+|Q, [P^+, P^-])$ and $q^- \approx P(P^-|Q, [P^+, P^-])$.

In the inference phase, the probability q_i is obtained by normalizing across all the retrieved paragraphs $\mathbf{P}$.

Above all, we describe our model with Algorithm 1 in the training phase and Algorithm 2 in the inference phase.

Algorithm 1 HAS-QA Model in Training Phase

Require: Q: question; A: answer string;
$\mathbf{P}$: retrieved paragraphs;
Ensure: $\mathcal{L}$: loss function

for P^+, P^- in $\mathbf{P}$ **do**:
  Get answer locations $\mathbf{L^s}$, $\mathbf{L^e}$ for P^+;
  Get the context embedding $\mathbf{C}$;
  Compute $\mathbf{p^s}$; (Eq 7)
  for L_j^s, L_j^e in $\mathbf{L^s}, \mathbf{L^e}$ **do**:
    $p_j^s \leftarrow \mathbf{p^s}[L_j^s]$;
    Compute $\mathbf{p_j^e}$; (Eq 8)
    $p_j^e \leftarrow \mathbf{p_j^e}[L_j^e]$;
    $s_j \leftarrow p_j^s p_j^e$;
  Apply function: $p^+ \leftarrow \mathcal{F}(\{s_j\})$;
  Compute q^+ in $[P^+, P^-]$; (Eq 11, Eq 12)
  $\mathcal{L}_i \leftarrow -(\log(q^+) + \log(p^+))$;
$\mathcal{L} \leftarrow \text{Avg}(\{\mathcal{L}_i\})$.

Algorithm 2 HAS-QA Model in Inference Phase

Require: Q: question; $\mathbf{P}$: retrieved paragraphs;
Ensure: A_{best}: answer string

for P_i in $\mathbf{P}$ **do**:
  Get the context embedding $\mathbf{C}$;
  Compute $\mathbf{p^s}$; (Eq 7)
  for L_j^s in Top-K_1 $\mathbf{p^s}$ **do**:
    $p_j^s \leftarrow \mathbf{p^s}[L_j^s]$;
    Compute $\mathbf{p_j^e}$; (Eq 8)
    for L_{jk}^e in Top-K_2 $\mathbf{p_j^e}$ **do**:
      $p_{jk}^e \leftarrow \mathbf{p_j^e}[L_{jk}^e]$;
      $s_{jk} \leftarrow p_j^s p_{jk}^e$;
  Group s_{jk} by extracted answer string A_t;
  Apply function: $p_i^{A_t} \leftarrow \mathcal{F}(\{s_{jk}\}_{A_t})$;
  Compute $\hat{q}_i$; (Eq 11)
Normalize $\{\hat{q}_i\}$ get $\{q_i\}$; (Eq 12)
$S(A_t) \leftarrow \sum_i q_i \cdot p_i^{A_t}$;
$A_{best} \leftarrow \arg\max(S(A_t))$.

5 Experiments

5.1 Datasets

We evaluate our model on three OpenQA datasets, QuasarT (Dhingra, Mazaitis, and Cohen 2017), TriviaQA (Joshi et al. 2017) and SearchQA (Dunn et al. 2017).

QuasarT[3]: consists of 43k open-domain trivia questions whose answers obtained from various internet sources. ClueWeb09 (Callan et al. 2009) serves as the background corpus for providing evidences paragraphs. We choose the Long version, which is truncated to 2048 characters and 20 paragraphs for each question.

TriviaQA[4]: consists of 95k open-domain question-answer pairs authored by trivia enthusiasts and independently gathered evidence documents from Bing Web Search and Wikipedia, six per question on average. We focus on the open domain setting contains unfiltered documents.

SearchQA[5]: is based on a Jeopardy! questions and collects about top 50 web page snippets from Google search engine for each question.

As we can see in Table 1, there exist amounts of negative paragraphs which contains no answer span, especially in TriviaQA and SearchQA. For all datasets, more than

[3]https://github.com/bdhingra/quasar

[4]http://nlp.cs.washington.edu/triviaqa/

[5]https://github.com/nyu-dl/SearchQA

Dataset	Neg Para. Ratio	Avg Ans. Span Count
QuasarT	1.21%	5.09
TriviaQA	37.24%	4.20
SearchQA	25.06%	6.80

Table 1: The negative paragraph ratio and average answer span count are statistic on three datasets, in order to illustrate the problems mentioned above in OpenQA task.

4 answer spans averagely obtained per paragraph. These statistics illustrate that problems mentioned above exist in OpenQA datasets.

5.2 Experimental Settings

For RC baseline models GA (Dhingra et al. 2017), BiDAF (Seo et al. 2016) and AQA (Buck et al. 2017), their experimental results are collected from published papers (Dunn et al. 2017; Joshi et al. 2017).

The DrQA (Chen et al. 2017), R^3 (Wang et al. 2018) and Shared-Norm (Clark and Gardner 2018) are evaluated using their released code[6].

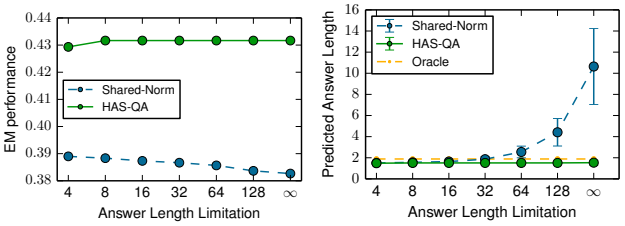
Our model [7] adopts the same data preprocessing and question context encoder presented in (Clark and Gardner 2018). In training step, we use the Adadelta optimizer (Zeiler 2012) with the batch size of 30, and we choose the model performed the best on develop set [8]. The hidden dimension of GRU is 200, and the dropout ratio is 0.8. We use 300 dimensional word embeddings pre-trained by GloVe (released by (Pennington, Socher, and Manning 2014)) and do not fine-tune in training step. Additionally, 20 dimensional character embeddings are left as learnable parameters. In inference step, for baseline models we set the answer length limitation to 8, while for our models it is unlimited. We analyze different answer length limitation settings in the Section 5.4. The parameters of beam search are $K_1 = 3$ and $K_2 = 1$.

5.3 Overall Results

The experimental results on three OpenQA datasets are shown in Table 2. It concludes as follow:

1) HAS-QA outperforms traditional RC baselines with a large gap, such as GA, BiDAF, AQA listed in the first part. For example, in QuasarT, it improves 16.8% in EM score and 20.4% in F1 score. As RC task is just a special case of OpenQA task. Some experiments on standard SQuAD dataset(dev-set) (Rajpurkar et al. 2016) show that HAS-QA yields EM/F1:0.719/0.798, which is comparable with the best released single model Reinforced Mnemonic Reader (Hu et al. 2017) in the leaderboard (dev-set) EM/F1:0.721/0.816. Our performance is slightly worse because Reinforced Mnemonic Reader directly use the accurate answer span, while we use multiple distantly supervised answer spans. That may introduce noises in the setting of SQuAD, since only one span is accurate.

2) HAS-QA outperforms recent OpenQA baselines, such as DrQA, R^3 and Shared-Norm listed in the second part. For example, in QuasarT, it improves 4.6% in EM score and 3.5% in F1 score.



Example:
About Celebrating the contributions of Louis Braille January 5th , 2009 On the 200th anniversary of Louis Braille ' s birth , people around the world are saluting a man whose tactile alphabet has provided a lifeline to people with impaired vision .
Shared-Norm HAS-QA

Figure 3: Results of Shared-Norm and HAS-QA on QuasarT. TopLeft: EM performance against answer length limitation, TopRight: predicted answer length against answer length limitation, Bottom: an example of a paragraph and the predicted answer spans of two models.

5.4 Model Analysis

In this subsection, we analyze our model by answering the following fine-grained analytic questions:

1) What advantages does HAS-QA have via modeling answer span using the conditional pointer network?

2) How much does HAS-QA gain from modeling multiple answer spans in a paragraph?

3) How does the paragraph quality work in HAS-QA?

The following three parts are used to answer these questions respectively.

Effects of Conditional Pointer Networks In order to demonstrate the effect of the conditional pointer networks, we compare Shared-Norm, which uses pointer networks, with our model. Then, we gradually remove the answer length limitation, from restricting 4 words to 128 words until no limitation (denote as ∞). Finally, we draw the tendency of the EM performance and average predicted answer length according to the different answer length limitations.

As shown in Figure 3 (TopLeft), the performance of Shared-Norm decreases when removing the answer length limitation, while the performance of HAS-QA first increases then becomes stable. In Figure 3 (TopRight), we find that the average predicted answer length increases in Shared-Norm when removing the answer length limitation. However, our model stably keeps average about 1.8 words, where the oracle average answer length is about 1.9 words. Example in Figure 3 (Bottom) illustrates that start/end pointers in Shared-Norm search their own optimal positions independently, such as two 'Louis' in paragraph. It leads to an unreasonable answer span prediction.

[6] DrQA: https://github.com/facebookresearch/DrQA.
R^3: https://github.com/shuohangwang/mprc.
Shared-Norm: https://github.com/allenai/document-qa.

[7] The code will be released at https://gitlab.com/pl8787/has-qa.

[8] QuasarT and SearchQA have official develop set and test set, while TriviaQA's test set is unknown, thus we split a develop set from train set and evaluate on official develop set.

Model	QuasarT EM	QuasarT F1	TriviaQA EM	TriviaQA F1	SearchQA EM	SearchQA F1
GA (Dhingra et al. 2017)	0.264	0.264	-	-	-	-
BiDAF (Seo et al. 2016)	0.259	0.285	0.411	0.474	0.286	0.346
AQA (Buck et al. 2017)	-	-	-	-	0.387	0.456
DrQA (Chen et al. 2017)	0.377	0.445	0.323	0.383	0.419	0.487
R^3 (Wang et al. 2018)	0.353	0.417	0.473	0.537	0.490	0.553
Shared-Norm (Clark and Gardner 2018)	0.386	0.454	0.613	0.672	0.598	0.671
HAS-QA (MAX Ans. Span)	**0.432**	**0.489**	**0.636**	**0.689**	**0.627**	**0.687**

Table 2: Experimental results on OpenQA datasets QuasarT, TriviaQA and SearchQA. EM: Exact Match.

Model	EM	F1
HAS-QA (HEAD Ans. Span)	0.372	0.425
HAS-QA (RAND Ans. Span)	0.341	0.394
HAS-QA (SUM Ans. Span)	0.423	0.484
HAS-QA (MAX Ans. Span)	0.432	0.489

Table 3: Results on QuasarT with different types of aggregation functions ($K_1 = 3, K_2 = 1$).

K_1-K_2	EM	F1	K_1-K_2	EM	F1
1-1	0.428	0.483	1-1	0.428	0.483
1-3	0.428	0.484	3-1	0.432	0.489
1-5	0.428	0.484	5-1	0.431	0.488
3-3	0.431	0.489	5-5	0.431	0.489

Table 4: Results on QuasarT with different beam search parameters K_1-K_2.

Effects of Multiple Spans Aggregation The effects of utilizing multiple answer spans lay into two aspects, 1) choose the aggregation functions in training phase, and 2) select the parameters of beam search in inference phase.

In the training phase, we evaluate four types of aggregation functions introduced in Section 4.3. The experimental results on QuasarT dataset, shown in Table 3, demonstrate the superiority of SUM and MAX operations. They take advantages of using multiple answer spans for training and improve about 6% - 10% in EM comparing to the HEAD operation. The performance of MAX operation is a little better than the SUM operation. The failure of RAND operation, mainly comes down to the conflicting training samples. Therefore, simple way to make use of multiple answer spans may not improve the performance.

In the inference phase, Table 4 shows the effects of parameters in beam search. We find that the larger K_1 yields the better performance, while K_2 seems irrelevant to the performance. As a conclusion, we choose the parameters $K_1 = 3, K_2 = 1$ to balance the performance and the speed.

Effects of Paragraph Quality The *paragraph probability* is efficient to measure the quality of paragraphs, especially for that containing useless paragraphs.

Figure 4 (Left) shows that with the increasing number of given paragraphs which ordered by the rank of a search engine, EM performance of HAS-QA sustainably grows. However, EM performance of Shared-Norm stops increasing at about 15 paragraphs and our model without paragraph quality (denotes PosOnly) stops increasing at about 5 paragraphs. So that with the help of *paragraph probability*, model performance can be improved by adding more evidence paragraphs.

We also evaluate the Mean Average Precision (MAP) score between the predicted scores and the label whether a paragraph contains answer spans (Figure 4 (Right)). The *paragraph probability* in our model outperforms PosOnly and Shared-Norm, so that it can rank the high quality paragraphs in the front of the given paragraph list.

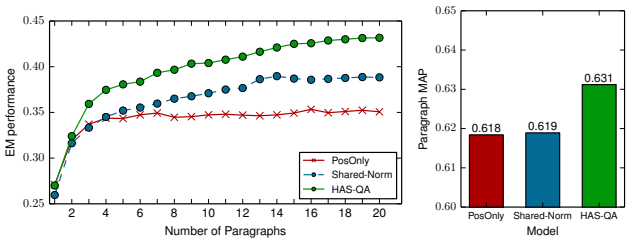


Figure 4: Results of PosOnly, Shared-Norm and HAS-QA on QuasarT. Left: EM performance against number of paragraphs, Right: paragraph MAP on different models.

6 Conclusions

In this paper, we point out three distinct characteristics of OpenQA, which make it inappropriate to directly apply existing RC models to this task. In order to tackle these problems, we first propose a new probabilistic formulation of OpenQA, where the *answer probability* is written as the question, paragraph and span, three-level structure. In this formulation, RC can be treated as a special case. Then, Hierarchical Answer Spans Model (HAS-QA) is designed to implement this structure. Specifically, a paragraph quality estimator makes it robust for the paragraphs without answer spans; a multiple span aggregator points out that it is necessary to combine the contributions of multiple answer spans in a paragraph, and a conditional span predictor is proposed to model the dependence between the start and end positions of each answer span. Experiments on public OpenQA datasets, including QuasarT, TriviaQA and SearchQA, show that HAS-QA significantly outperforms traditional RC baselines and recent OpenQA baselines.

Acknowledgments

This work was funded by the National Natural Science Foundation of China (NSFC) under Grants No. 61773362, 61425016, 61472401, 61722211, and 61872338, the Youth Innovation Promotion Association CAS under Grants No. 20144310, and 2016102, and the National Key R&D Program of China under Grants No. 2016QY02D0405.

References

Berant, J.; Chou, A.; Frostig, R.; and Liang, P. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 1533–1544.

Buck, C.; Bulian, J.; Ciaramita, M.; Gesmundo, A.; Houlsby, N.; Gajewski, W.; and Wang, W. 2017. Ask the right questions: Active question reformulation with reinforcement learning. *arXiv preprint arXiv:1705.07830*.

Callan, J.; Hoy, M.; Yoo, C.; and Zhao, L. 2009. Clueweb09 data set.

Chen, D.; Fisch, A.; Weston, J.; and Bordes, A. 2017. Reading wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 1870–1879.

Cho, K.; van Merrienboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734.

Clark, C., and Gardner, M. 2018. Simple and effective multi-paragraph reading comprehension. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 845–855.

Dhingra, B.; Liu, H.; Yang, Z.; Cohen, W.; and Salakhutdinov, R. 2017. Gated-attention readers for text comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 1832–1846.

Dhingra, B.; Mazaitis, K.; and Cohen, W. W. 2017. Quasar: Datasets for question answering by search and reading. *arXiv preprint arXiv:1707.03904*.

Dunn, M.; Sagun, L.; Higgins, M.; Guney, U.; Cirik, V.; and Cho, K. 2017. Searchqa: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*.

Ferrucci, D.; Brown, E.; Chu-Carroll, J.; Fan, J.; Gondek, D.; Kalyanpur, A. A.; Lally, A.; Murdock, J. W.; Nyberg, E.; Prager, J.; et al. 2010. Building watson: An overview of the deepqa project. *AI magazine* 31(3):59–79.

Hu, M.; Peng, Y.; Huang, Z.; Qiu, X.; Wei, F.; and Zhou, M. 2017. Reinforced mnemonic reader for machine reading comprehension. *arXiv preprint arXiv:1705.02798*.

Joshi, M.; Choi, E.; Weld, D.; and Zettlemoyer, L. 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)*, volume 1, 1601–1611.

Mou, L.; Lu, Z.; Li, H.; and Jin, Z. 2017. Coupling distributed and symbolic execution for natural language queries. In *International Conference on Machine Learning*, 2518–2526.

Pan, B.; Li, H.; Zhao, Z.; Cao, B.; Cai, D.; and He, X. 2017. Memen: Multi-layer embedding with memory networks for machine comprehension. *arXiv preprint arXiv:1707.09098*.

Pennington, J.; Socher, R.; and Manning, C. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532–1543.

Rajpurkar, P.; Zhang, J.; Lopyrev, K.; and Liang, P. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2383–2392.

Seo, M.; Kembhavi, A.; Farhadi, A.; and Hajishirzi, H. 2016. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Tan, C.; Wei, F.; Yang, N.; Lv, W.; and Zhou, M. 2018. S-net: From answer extraction to answer generation for machine reading comprehension. In *In Proceedings of the 32th AAAI Conference on Artificial Intelligence*.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, 2692–2700.

Wang, S., and Jiang, J. 2016. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*.

Wang, W.; Yang, N.; Wei, F.; Chang, B.; and Zhou, M. 2017. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, 189–198.

Wang, S.; Yu, M.; Guo, X.; Wang, Z.; Klinger, T.; Zhang, W.; Chang, S.; Tesauro, G.; Zhou, B.; and Jiang, J. 2018. R^3: Reinforced reader-ranker for open-domain question answering. In *In Proceedings of the 32th AAAI Conference on Artificial Intelligence*.

Weissenborn, D.; Wiese, G.; and Seiffe, L. 2017. Fastqa: A simple and efficient neural architecture for question answering. *arXiv preprint arXiv:1703.04816*.

Xiong, C.; Zhong, V.; and Socher, R. 2016. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*.

Zeiler, M. D. 2012. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.