Multi-span Style Extraction for Generative Reading Comprehension

Junjie Yang[1,3,4], **Zhuosheng Zhang**[2,3,4], **Hai Zhao**[2,3,4*]
[1]SJTU-ParisTech Elite Institute of Technology, Shanghai Jiao Tong University
[2]Department of Computer Science and Engineering, Shanghai Jiao Tong University
[3]Key Laboratory of Shanghai Education Commission for Intelligent Interaction
and Cognitive Engineering, Shanghai Jiao Tong University, Shanghai, China
[4]MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University, Shanghai, China
jj-yang@sjtu.edu.cn, zhangzs@sjtu.edu.cn, zhaohai@cs.sjtu.edu.cn

Abstract

Generative machine reading comprehension (MRC) requires a model to generate well-formed answers. For this type of MRC, answer generation method is crucial to the model performance. However, generative models, which are supposed to be the right model for the task, in generally perform poorly. At the same time, single-span extraction models have been proven effective for extractive MRC, where the answer is constrained to a single span in the passage. Nevertheless, they generally suffer from generating incomplete answers or introducing redundant words when applied to the generative MRC. Thus, we extend the single-span extraction method to multi-span, proposing a new framework which enables generative MRC to be smoothly solved as multi-span extraction. Thorough experiments demonstrate that this novel approach can alleviate the dilemma between generative models and single-span models and produce answers with better-formed syntax and semantics.

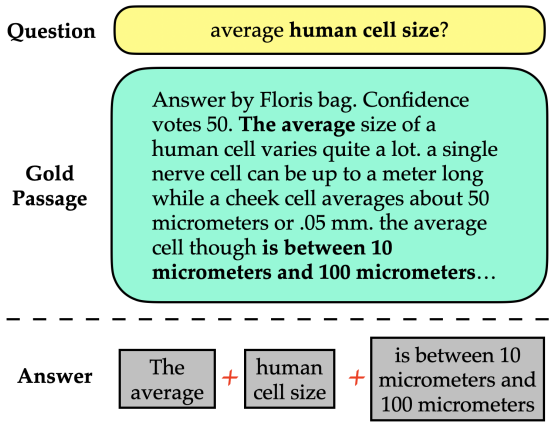

Figure 1: Example of how a well-formed answer is generated by the multi-span style extraction.

Introduction

Machine Reading Comprehension (MRC) is considered as a nontrivial challenge in natural language understanding. Recently, we have seen continuous success in this area, partially benefiting from the release of massive and well-annotated datasets from both academic (Rajpurkar, Jia, and Liang 2018; Reddy, Chen, and Manning 2019) and industry (Bajaj et al. 2018; He et al. 2018) communities.

The widely used span-extraction models (Seo et al. 2017; Ohsugi et al. 2019; Lan et al. 2020), formulate the MRC task as a process of predicting the start and end position of the span inside the given passage. They have been proven effective on the tasks which constrain the answer to be an exact span in the passage (Rajpurkar, Jia, and Liang 2018). However, for generative MRC tasks whose answers are highly abstractive, the single-span extraction based methods can easily suffer from incomplete answers or redundant words problem. Thus, there still exists a large gap between the performance of single-span extraction baselines and human performance.

In the meantime, we have observed that utilizing multiple spans appearing in the question and passage to compose the well-formed answer could be a promising method to alleviate these drawbacks. Figure 1 shows how the mechanism of multi-span style extraction works for an example from the MS MARCO task (Bajaj et al. 2018), where the well-formed answer cannot simply be extracted as a single span from the input text.

Therefore, in this work, we propose a novel answer generation approach that takes advantage of the effectiveness of span extraction and the concise spirit of multi-span style to synthesize the free-formed answer, together with a framework as a whole for the multi-passage generative MRC. We call our framework MUSST for **MU**lti-**S**pan **ST**yle extraction. Our framework is also empowered by well pre-trained language model as encoder component of our model. It provides deep understanding of both the input passage and question, and models the information interaction between

*Corresponding author. This paper was partially supported by National Key Research and Development Program of China (No. 2017YFB0304100), Key Projects of National Natural Science Foundation of China (U1836222 and 61733011), Huawei-SJTU long term AI project, Cutting-edge Machine reading comprehension and language model.

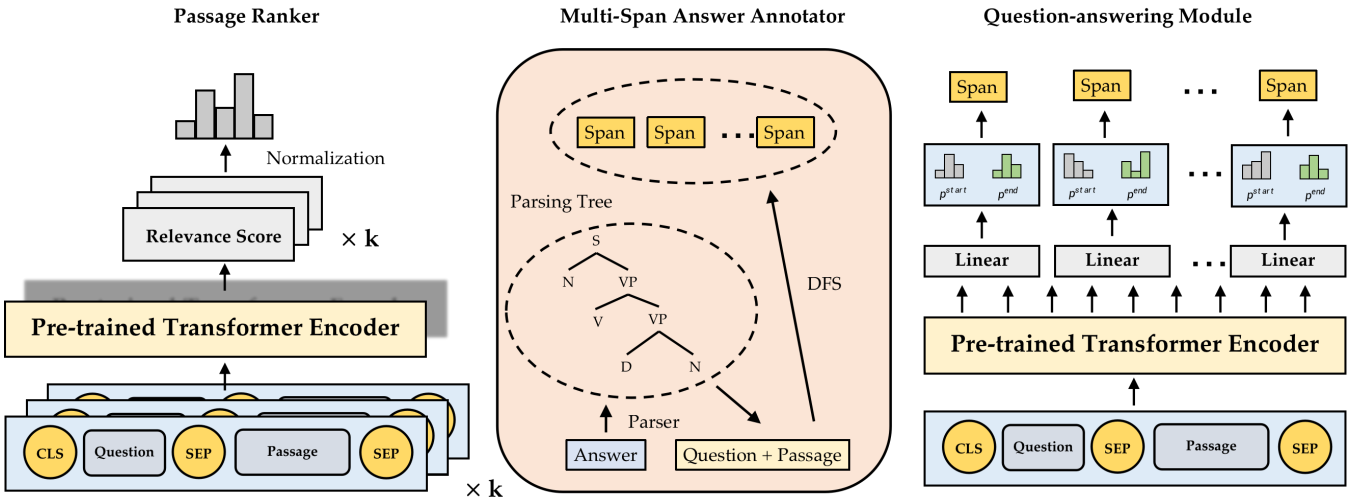

Figure 2: Our framework MUSST

them. We conduct a series of experiments and the corresponding ablations on the MS MARCO v2.1 dataset.

Our main contributions in this paper can be summarized as follows[1]:

- We propose a novel multi-span answer annotator to transform the initial well-formed answer into a series of spans that distribute in the question and passage.
- We generalize the single-span extraction based method to the multi-span style by introducing a lightweight but powerful answer generator, which supports the extraction of various number answer spans during prediction.
- To make better usage of the large dataset for the passage ranking task, we propose dynamic sampling during the training of the ranker that selects the passage most likely to entail the answer.

MUSST

In this section, we present our proposed framework, MUSST, for multi-passage generative MRC task. Figure 2 depicts the general architecture of our framework, which consists of a passage ranker, a multi-span answer annotator, and a question-answering module.

Passage ranker

Problem formulation Given a question Q and a set of k candidate passages $\mathbf{P} = \{P_1, P_2, ..., P_k\}$, the passage ranker is responsible for ranking the passages based on their relevance to the question. In other words, the model is requested to output conditional probability distribution $P(y|Q, \mathbf{P}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the model parameters and $P(y = i|Q, \mathbf{P}; \boldsymbol{\theta})$ denotes the probability that passage P_i can be used to answer question Q.

[1]The code is publicly available at: https://github.com/chunchiehy/musst

Encoder For each input question and passage pair (Q, P_i), we represent it as a single packed sequence of length n of the form "`[CLS]`Q`[SEP]`P_i`[SEP]`". We pass the whole sequence into a contextualized encoder, thereby to produce its contextualized representation $\boldsymbol{E} \in \mathbb{R}^{n \times h}$ where h denotes the hidden size of the Transformer blocks. Following the fine-tuning strategy of Devlin et al. (2019) for the classification task, we consider the final hidden vector $\boldsymbol{c} \in \mathbb{R}^h$ corresponding to the first input token (`[CLS]`) as the input's aggregate representation. Our encoder also models the interaction between the question and the passage.

Ranker The ranker is responsible for ranking the passages based on its relevance to the question. Given the output of the encoding layer $\boldsymbol{c}$, we pass it through a fully connected multi-layer perceptron which consists of two linear transformations with a Tanh activation in between:

$$\boldsymbol{s} = \text{softmax}(\boldsymbol{W}_2 \tanh(\boldsymbol{W}_1 \boldsymbol{c} + \boldsymbol{b}_1) + \boldsymbol{b}_2) \in \mathbb{R}^2$$
$$u_i = s_0 \text{ and } r_i = s_1$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{h \times h}$, $\boldsymbol{W}_2 \in \mathbb{R}^{2 \times h}$, $\boldsymbol{b}_1 \in \mathbb{R}^h$ and $\boldsymbol{b}_2 \in \mathbb{R}^2$ are trainable parameters. Here, r_i and u_i are respectively the relevance and unrelevance score for the pair (Q, P_i). The relevance scores are consequently normalized across all the candidates passages of the same question:

$$\hat{r}_i = \frac{\exp(r_i)}{\sum_{j=0}^{k} \exp(r_j)}$$

Here, $\hat{r}_i$ indicates the probability that passage P_i entails the answer Q.

Training We define the question-passage pair where the passage entails the question as a positive training sample. The positive passage is noted as P^+. During the training phase, we adopt a negative sampling with one negative sample. Specifically, for each positive instance (Q, P^+), we randomly sample a negative passage P^- from the *unselected*

passages of the same question. The model is trained by minimizing the following cost function:

$$J(\boldsymbol{\theta}) = -\frac{1}{T}\sum_{t=1}^{T}\log(r(Q_t, P_t^+)) + \log(u(Q_t, P_t^-))$$

where T is the number of questions in the training set, $r(Q_t, P_t^+)$ denotes the relevance score of (Q_t, P_t^+) and $u(Q_t, P_t^-)$ denotes the unrelevance score of (Q_t, P_t^-).

Moreover, motivated by Liu et al. (2019), we resample the negative training instances at the beginning of each training epoch, to avoid using the same training pattern for the question during each training epoch. We name it **dynamic sampling**.

Syntactic multi-span answer annotator

Algorithm 1 Syntactic Multi-span Answer Annotation

Input: Question $Q = \{q_1, q_2, \ldots, q_m\}$, passage $P = \{p_1, p_2, \ldots, p_n\}$ and gold answer $A = \{a_1, a_2, \ldots, a_k\}$
Parameter: Edit distance threshold d_{max}
Output: A list of start and end position of answer spans in the question and passage

Let M be an empty list
Pack question Q and passage P into a single sequence C in a certain way.
Get the syntactic parsing tree $\mathcal{T}$ of answer A by a constituency parser.
Let $\mathbb{S}$ be the stack of subtrees to be traversed.
Initialize $\mathbb{S}$ with the root $\mathcal{R}$ of the tree $\mathcal{T}$
while $\mathbb{S}$ is not empty **do**
    let $\mathcal{V} =$ POP($\mathbb{S}$)
    Get a list of all the leaves of subtree $\mathcal{V}$: $L = \{l_1, l_2, \cdots, l_n\}$
    if L is a sublist of C **then**
        Get the start index s and end index e of L in C by Knuth-Morris-Pratt pattern searching algorithm
        Add (s, e) into the span position list M
    else
        for childtree $\mathcal{U}$ in $\mathcal{V}$ (From right to left) **do**
            PUSH($\mathbb{S}, \mathcal{U}$)
        end for
    end if
end while
Reconstruct answer A' from span position list M
Let $d =$ EDITDISTANCE(A, A')
if $d > d_{max}$ **then**
    Empty the list M
end if
M^* = PRUNING(M)
return M^*

In this section, we introduce our syntactic multi-span answer annotator. Before the training of our question-answering module, we need to extract non-overlapped spans from the question and passage based on the original answer from the training dataset. Our annotator is responsible for transforming the original answer phrase into multiple spans that distribute in the question and passage with subject to syntactic constraints. The attempt to extract the answer spans syntactically is motivated by our first intuition that the human editors compose the original answer in an analogous way.

As shown in the middle of Figure 2, we transform the answer phrase into a parsing tree and traverse the parsing tree in a DFS (Depth-first search) way. At each visit of the subtree, we check if the span represented by the subtree appears in the question or passage text. We obtain a span list after traversing the whole parsing tree. However, in some cases, the original answer still cannot be perfectly composed by the words from the input text even in a multi-span style. We get rid of these *bad* samples by comparing their edit distances with a threshold value which is set by the model beforehand.

An important final step is to prune the answer span list. The **pruning** procedure sticks to the following principle: if two spans adjoint in the list are contiguous in the original text, we joint them together. Pruning reduces heavily the number of spans needed to recover to the original answer phrase. The more comprehensive detail of our annotator is described in Algorithm 1.

Question-answering module

Problem formulation Given a question Q and a passage P, the question-answering module is requested to answer the question based on the information provided by the passage. In other words, the model outputs the conditional probability distribution $P(y|Q, P)$, where $P(y = A|Q, P)$ denotes the probability that A is the answer.

Question-passage reader The architecture of the reader is analogous to the encoder module of the ranker in section , where we take a pre-trained language model as encoder. But instead of getting only the aggregate representation, we pass the whole output of the last layer to predict the answer spans as the follows:

$$\boldsymbol{M} = \text{Encoder}(Q, P) \in \mathbb{R}^{h\times n}$$

where n is the length of the input token sequence, and h is the hidden size of the encoder.

Multi-span style answer generator Our answer generator is responsible for composing the answer in a multi-span style extraction. Let n be the number of span to be extracted.

For each single span prediction, we treat it as the single span extraction MRC task. Following Lan et al. (2020), we adopt a linear layer to predict start and end positions of the span in the input sequence. It is worth noticing that our model is also enabled to predict the answer span from the question. The probability distribution of i-th span's start position over the input tokens is obtained by:

$$\hat{\boldsymbol{p}}^{j,\text{start}} = \text{softmax}(\boldsymbol{W}_j^s \boldsymbol{M} + b_j^s)$$

where $\boldsymbol{W}_j^s \in \mathbb{R}^{1\times h}$ and $b_j^s \in \mathbb{R}$ are trainable parameters and $\hat{\boldsymbol{p}}_k^{j,\text{start}}$ denote the probability of token k being the start of the answer span j. The end position distribution of the answer span j is obtained by using the analogous formula:

$$\hat{\boldsymbol{p}}^{j,\text{end}} = \text{softmax}(\boldsymbol{W}_j^e \boldsymbol{M} + b_j^e)$$

Training and inference During training, we add a special virtual span, with start and end position values equaling the length of the input sequence, at the end of the annotated answer span list. This approach enables our model to generate a various number of answer spans during prediction with the virtual span serving as a stop symbol. The cost function is defined as follows:

$$J(\boldsymbol{\theta}) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{m_t}\log(\hat{\boldsymbol{p}}^{j,\text{start}}_{y_t^{j,\text{start}}}) + \log(\hat{\boldsymbol{p}}^{j,\text{end}}_{y_t^{j,\text{end}}})$$

where T is the number of training samples, m_t is the number of answer span for sample t, $y_t^{j,\text{start}}$ and $y_t^{j,\text{end}}$ are the true start and end position of the t-th sample's j-th span.

During inference, at each time step j, we choose the answer span (k, l) where $k < l$ with the maximum value of $\hat{\boldsymbol{p}}_k^{j,\text{start}}\hat{\boldsymbol{p}}_l^{j,\text{end}}$. The decoding procedure terminates when the stop span is predicted. Sometimes, the model tends to generate repeatedly the same spans. In order to alleviate the repeating problem, at each prediction time step j, we mask out the predicted span positions of previous time steps ($< j$) during the calculation of probability distribution of new start and end positions. Since the masking depends on the previously predicted spans, we name it as **conditional masking**. The extracted spans are later joined together to form a final answer phrase.

Experiments

Dataset

We evaluate our framework on the MS MARCO v2.1 [2] (Bajaj et al. 2018), which is a large scale open-domain generative task. MS MARCO v2.1 provides two MRC tasks: Question Answering (**QA**) and Natural Langauge Generation (**NLG**). The statistics of the corresponding datasets' size are presented in Table 1. Both datasets consist of sampled questions from Bing's search logs, and each question is accompanied by an average of ten passages that may contain the answers. $\mathcal{QA}$ and $\mathcal{NLG}$ are subsets of $\mathcal{ALL}$, which also contains the unanswerable questions.

Distinguished with the QA task, the NLG task requires the model to provide the well-formed answer, which could be read and understood by a natural speaker without any additional context. Therefore NLG-style answers are more abstract than the QA-style answers. Table 1 shows also the percentage of examples where the answer can be extracted as a single span in the gold passage. Unsurprisingly, the answers from the $\mathcal{QA}$ set are much more likely to match a span in the passage than the ones in the $\mathcal{NLG}$ set. Moreover, Nishida et al. (2019) states that the QA task prefers the answer to be more concise than in the NLG task, averaging 13.1 words, while the latter one averages 16.6 words. Therefore, the $\mathcal{NLG}$ set is more suitable to evaluate model performance on generative MRC.

BLEU-1 (Papineni et al. 2002) and ROUGE-L (Lin 2004) are adopted as the official evaluation [3] metrics to evaluate model performance, while the official leaderboard chooses ROUGE-L as the main metric. In the meantime, we use Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR) for our ranker.

Dataset	Train	Dev	Test
$\mathcal{ALL}$	808,731	101,093	101,092
$\mathcal{QA}$	503,370 (63.39%)	55,636 (45.40%)	–
$\mathcal{NLG}$	153,725 (12.57%)	12,467 (24.99%)	–

Table 1: Statistics of MS MARCO v2.1 dataset. The numbers in parenthesis indicate the percentage of examples whose answer is single span in gold passage.

Baseline models

We compare our MUSST with the following baseline models: single-span extraction and seq2seq. For the single-span extraction baseline, we employ the model for the SQuAD dataset from ALBERT (Lan et al. 2020). The model is trained only with samples where the answer is a single span in the passage. In the meantime, We adopt the Transformer model from Vaswani et al. (2017) as our seq2seq baseline. For a fair comparison, the baseline models share the same passage ranker as the one in MUSST.

Implementation details

For the multi-span answer annotation, we use constituency parser from Standford CoreNLP (Manning et al. 2014). NLTK [4] package is also used to implement our annotator. The maximum edit distance between the answer reconstructed from the annotated spans, and the original answer is 32 and 8 respectively for the $\mathcal{NLG}$ and $\mathcal{QA}$ training sets.

The ranker and question-answering module of MUSST are implemented with PyTorch [5] and Transformers package [6]. We adopt ALBERT (Lan et al. 2020) as the encoder in our models and initialize it with the pre-trained weights before the fine-tuning. We choose ALBERT-base as the encoder of passage ranker and ALBERT-xlarge instead for question answering module.

Following Lan et al. (2020), we use SentencePiece (Kudo and Richardson 2018) to tokenize our inputs with a vocabulary size of 30,000. We adopt Adam optimizer (Kingma and Ba 2015) to minimize the cost function. Two types of regularization methods during training: dropout and L2 weight decay. Hyperparameter details for the training of the different models of our framework are presented in Table 2. MUSST-NLG and MUSST-QA are trained respectively on the $\mathcal{NLG}$ and $\mathcal{QA}$ subsets. The maximum number of

[2]The datasets can be obtained from the official site (https://microsoft.github.io/msmarco/)

[3]The official evaluation scripts can be found in https://github.com/microsoft/MSMARCO-Question-Answering/tree/master/Evaluation

[4]https://www.nltk.org

[5]https://pytorch.org

[6]https://github.com/huggingface/transformers

spans for them is set to 9 and 5, respectively. We trained the passage ranker and the question-answering module of MUSST-NLG on a machine with four Tesla P40 GPUs. The question-answering module of MUSST-QA is trained with eight GeForce GTX 1080 Ti GPUs. It takes roughly 9 hours to train the passage ranker. For the question-answering module in MUSST-NLG and MUSST-QA, the training time is about 10 hours and 17 hours respectively.

Hyperparameter	**Ranker**	**MUSST-QA**	**MUSST-NLG**
Learning rate	1e-5	3e-5	3e-5
Learning rate decay	Linear	Linear	Linear
Training epoch	3	3	5
Warmup rate	0.1	0.1	0.1
Adam ϵ	10^{-6}	10^{-6}	10^{-6}
Adam β_1	0.9	0.9	0.9
Adam β_2	0.999	0.999	0.999
MSN	256	256	256
Batch size	128	32	32
Encoder dropout rate	0	0	0
Classifier dropout rate	0.1	0.1	0.1
Weight decay	0.01	0.01	0.01

Table 2: Training hyperparameters of different modules of MUSST on MS MARCO v2.1 dataset. Here, MUSST-QA and MUSST-NLG refer to its question-answering module. MSN means maximum sequence length.

The single-span baseline is implemented with the same packages as MUSST while the seq2seq baseline is implemented with Fairseq (Ott et al. 2019).

Results

Model	$\mathcal{QA}$		$\mathcal{NLG}$	
	ROUGE-L	**BLEU-1**	**ROUGE-L**	**BLEU-1**
Single-span	47.96	**50.22**	53.10	49.08
Seq2seq	–	–	56.42	53.89
MUSST-QA	**48.44**	49.54	–	–
MUSST-NLG	–	–	**66.24**	**64.23**

Table 3: Performance comparison with our baselines on the $\mathcal{QA}$ and $\mathcal{NLG}$ development set. Here, we use the same single ranker for MUSST and the baselines.

Table 3 shows the results of our single model and the baseline models on the $\mathcal{QA}$ and $\mathcal{NLG}$ development datasets. MUSST outperforms significantly the baselines including the generative seq2seq model over the $\mathcal{NLG}$ set in terms of both ROUGE-L and BLEU-1. Even on the $\mathcal{QA}$ set, our model yields better results regarding ROUGE-L. Table 4 compares our model performance with the competing models on the leaderboard. Although our model utilizes only a standalone classifier for passage ranking, multi-span style extraction still helps us rival with state-of-the-art approaches.

Analysis and discussions

Effect of maximum number of spans

Figure 3 presents the distribution of span numbers with edit distance less than 4 over the $\mathcal{QA}$ and $\mathcal{NLG}$ training sets after the annotation procedure. It is seen that most QA-style answers are only one span, while the NLG-style answers distribute more uniformly in the range of [1, 9].

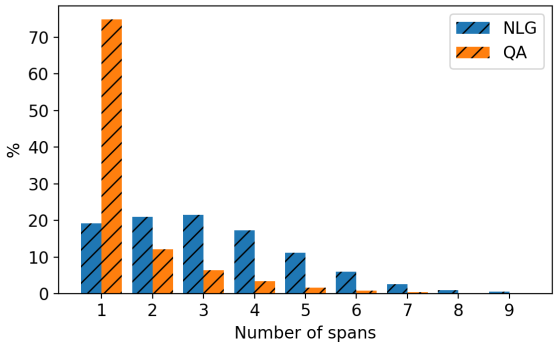

Figure 3: Distribution of training samples of edit distance less than 4 over annoted answer spans. For the purpose of better illustration, we filter the samples which include more than 9 spans.

To better understand the effect of the maximum number of spans to be generated in the answer generator, we let it vary in the range of [2, 12] and conduct experiments on the $\mathcal{NLG}$ set with our best single passage ranker. The edit distance threshold is set to be 8. The results are presented in Figure 4. Generally, increasing the number of the span will augment the token coverage rate, thus yielding better results. But the gain becomes less significant when the maximum number of span is already large enough. From Figure 4, we can see that the results vary imperceptibly when the maximum number of spans reaches 5. However, since each span only introduces 4k parameters, which is negligible before the encoder (60M), we still choose the maximum number to be 9, which corresponds to the best performance on the development set.

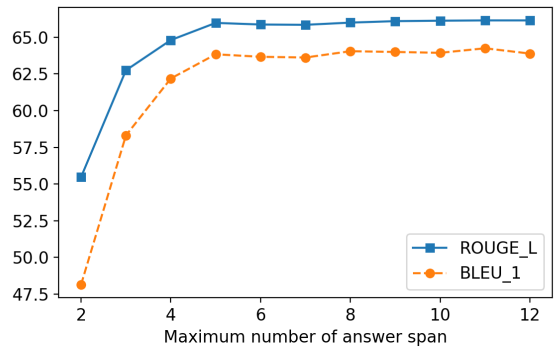

Figure 4: Effect of maximum number of spans.

Model	Answer Generation	Ranking	NLG Task R-L	NLG Task B-1	QA Task R-L	QA Task B-1	Overall Average
Human	–	–	63.2	53.0	53.9	48.5	54.65
Unpublished							
PALM	Unknown		**49.8**	49.9	51.8	50.7	**50.55**
Multi-doc Enriched BERT	Unknown		32.5	37.7	**54.0**	**56.5**	45.18
Published							
BiDAF[a] ♠	Single-span	Confidence score	16.9	9.3	24.0	10.6	15.20
ConZNet[b] ♠	Pointer-Generator	Unkonwn	42.1	38.6	–	–	–
VNET[c] ♠	Single-span	Answer verification	48.4	46.8	51.6	54.3	**50.28**
Deep Cascade QA[d] ♠	Single-span	Cascade	35.1	37.4	52.0	54.6	44.78
Masque QA[e] †	Pointer-Generator	Joint trained classifier	28.5	39.9	52.2	43.7	41.08
Masque NLG[e] †	Pointer-Generator	Joint trained classifier	49.6	**50.1**	48.9	48.8	49.35
MUSST-NLG †	Multi-span	Standalone classifier	48.0	45.8	49.0	51.6	48.60

Table 4: The performance of our framework and competing models on the MS MARCO v2.1 test set. All the results presented here reflect the MS MARCO leaderboard (microsoft.github.io/msmarco/) as of 28 May 2020. ♠ refers to the model whose results are not reported in the original published paper. BiDAF for MARCO is implemented by the official MS MARCO Team. † refers to the ensemble submission. Whether the other competing models are ensemble or not is unclear. [a] Seo et al. (2017); [b] Indurthi et al. (2018); [c] Wang et al. (2018b); [d] Yan et al. (2019); [e] Nishida et al. (2019).

Ablation study on model design choice

We perform ablation experiments that quantify the individual contribution of the design choices of MUSST. Table 5 shows the results on the $\mathcal{NLG}$ development set. Both *pruning* and *conditional masking* contribute the model performance, which indicates that pruning can help the model to converge more easily by reducing the number of spans, while conditional masking can better generate answer without suffering from the repeating problem. We also observe using the gold passage can significantly improve question-answering. It shows there still exists a great improvement space for the passager ranker.

Model	ROUGE-L	BLEU-1
MUSST	66.24	64.23
w/o pruning	64.66	60.36
w/o conditional masking	65.50	64.31
MUSST w gold passage	75.39	74.41

Table 5: Ablation study on the $\mathcal{NLG}$ development set.

Quality of multi-span answer annotator

On the $\mathcal{NLG}$ development set, we evaluate the answers generated by our syntactic multi-span annotator. The results shows our annotated answers can obtain 89.35 in BLEU-1 and 90.19 in ROUGE-L with the gold passages, which demonstrates the effectiveness of our annotator. For MUSST, the results are 74.41 and 75.39 respectively (in Table 5). So there is still much room for improvement with respect to the question-answering module.

Effect of edit distance threshold

Figure 5 shows the results of MUSST on $\mathcal{NLG}$ development set for various edit distance threshold. Interestingly, it indicates that BLEU-1 is impacted more heavily by the variation of edit distance than ROUGE-L. And setting the edit distance threshold too large may damage the model performance by introducing too many *incomplete* samples.

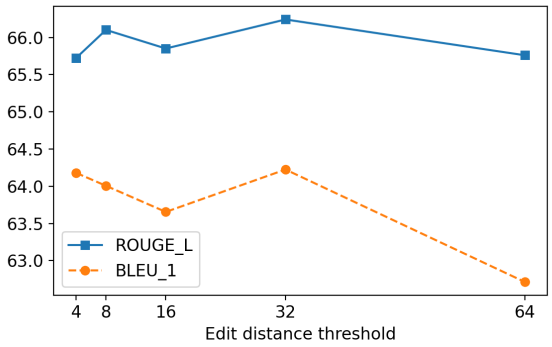

Figure 5: Effect of edit distance threshold.

Effect of encoder size

Table 6 presents experimental results on ALBERT encoder with various model sizes. Unsurprisingly, the model yields stronger results as the encoder gets larger.

Performance of the ranker

Table 7 presents our ranker performance in terms of MAP and MRR. The results show that dynamic sampling leads to slightly better results.

Encoder	Parameters	ROUGE-L	BLEU-1
ALBERT-base	12M	62.03	60.48
ALBERT-large	18M	64.93	61.67
ALBERT-xlarge	60M	**66.24**	**64.23**

Table 6: Effect of ALBERT encoder size.

Model	Training set	MAP	MRR
Bing (initial ranking)	-	34.62	35.00
MUSST (single)	$\mathcal{QA}$	**71.10**	**71.56**
w/o dynamic sampling	$\mathcal{QA}$	70.82	71.26

Table 7: The performance of ranker with various configurations on the $\mathcal{QA}$ development set.

Case study

To have an intuitive observation of the prediction ability of MUSST, we show a prediction example on MS MARCO v2.1 from the baseline and MUSST in Table 8. The comparison indicates that our model could extract effectively useful spans, yielding more complete answer that can be understood independent of question and passage context.

Question: *how long should a central air conditioner last* **Selected Passage**: *10 to 20 years - sometimes longer. You should have a service tech come out once a year for a tune up. You wouldn't run your car without regular maintenance and tune ups and you shouldn't run your a/c that way either - if you want it to last as long as possible. Source(s): 20 years working for a major manufacturer of central heating and air conditioning.* **Reference Answer**: *A Central air conditioner lasts for in between 10 and 20 years./ A central air conditioner should last for 10 to 20 years.* **Prediction (Baseline)**: *10 to 20 years.* **Prediction (MUSST)**: *a central air conditioner should last for 10 to 20 years.*

Table 8: A prediction example from the baseline and MUSST. The underlined texts are the spans predicted by our model to compose the final answer phrase.

Related work

Generative MRC

Generative MRC is considered as a more challenging task where answers are free-form human-generated text. More recently, we have seen an emerging wave of generative MRC tasks, including MS MARCO (Bajaj et al. 2018), NarrativeQA (Kočiský et al. 2018), DuReader (He et al. 2018) and CoQA (Reddy, Chen, and Manning 2019).

The most earlier approaches tried to generate the answer in a single-span extractive way (Tay et al. 2018; Tay, Luu, and Hui 2018; Wang et al. 2018b; Yan et al. 2019; Ohsugi et al. 2019). The models using a single-span extractive method show effectiveness for the dataset where abstractive behavior of answers includes mostly small modifications to spans in the context (Ohsugi et al. 2019; Yatskar 2019). Whereas, for the datasets with answers of deep abstraction, this method fails to yield promising results. The first attempt to generate the answer in a generative way is to apply an RNN-based seq2seq attentional model to synthesize the answer, such as S-NET (Tan et al. 2018), where seq2seq learning was first introduced by Sutskever, Vinyals, and Le (2014) for the machine translation. The most recent models adopt a hybrid neural network Pointer-Generator (See, Liu, and Manning 2017) to generate answer, such as ConZNet (Indurthi et al. 2018), MHPGM (Bauer, Wang, and Bansal 2018) and Masque (Nishida et al. 2019). Pointer-Generator was firtsly proposed for the abstractive text summarization, which can copy words from the source via the pointer network while retaining the ability to produce novel words through the generator. Different from ConZNet and MHPGM, Masque adopt a Transformer-based (Vaswani et al. 2017) Pointer-Generator, while the previeous ones utilizeing GRU (Cho et al. 2014) or LSTM (Hochreiter and Schmidhuber 1997).

Multi-passage MRC

For each question-answer pair, the Multi-passage MRC dataset contains more than one passage as the reading context, such as SearchQA (Dunn et al. 2017), Triviaqa (Joshi et al. 2017), MS MARCO, and DuReader.

Existing approaches designed specifically for Multi-passage MRC can be classified into two categories: pipeline and end-to-end. Pipeline-based models (Chen et al. 2017; Wang et al. 2018a; Clark and Gardner 2018) adopt a ranker to first rank all the passages based on its relevance to the question and then utilize a question-answering module to read the selected passages. The ranker can be based on traditional information retrieval methods (BM25 or TF-IDF) or employ a neural re-ranking model. End-to-end models (Wang et al. 2018b; Tan et al. 2018; Nishida et al. 2019) read all the provided passages at the same time, and produce for each passage a candidate answer assigned with a score which is consequently compared among passages to find the final answer. Passage ranking and answer prediction are usually jointly done as multi-task learning. More recently, Yan et al. (2019) proposed a cascade learning model to balance the effectiveness and efficiency of the two approaches mentioned above.

Pre-trained language model in MRC

Employing the pre-trained language models has been a common practice for tackling MRC tasks (Zhang, Zhao, and Wang 2020). The appearances of more elaborated architectures, larger corpora, and more well-designed pre-training objectives speed up the achievement of new state-of-the-art in MRC (Devlin et al. 2019; Liu et al. 2019; Yang et al. 2019; Lan et al. 2020). Moreover, Glass et al. (2019) adopts span selection, a MRC task, as an auxiliary pre-training task. Another mainstream line of research attempts

to drive the improvements during the fine-tuning, which includes integrating better verification strategies for unanswerable question (Zhang, Yang, and Zhao 2020), incorporating explicit linguistic features (Zhang et al. 2020b,c), leveraging external knowledge for commonsense reasoning (Lin et al. 2019) or enhancing matching network for multi-choice MRC (Zhang et al. 2020a; Zhu, Zhao, and Li 2020). In addition,Hu et al. (2019) introduced multi-span extraction to obtain top-k most likely spans for multi-type MRC. However, different from our work, this method is more suitable to predict a set of independent answer spans instead of generating a complete sentence.

Conclusion

In this work, we present a novel solution to generative MRC, multi-span style extraction framework (MUSST), and show it is capable of alleviating the problems of generating incomplete answers or introducing redundant words encountered by single-span extraction models. We apply our model to a challenging generative MRC dataset MS MARCO v2.1 and significantly outperform the single-span extraction baseline. This work indicates a new research line for generative MRC in addition to the existing two methods, single-span extraction and seq2seq generation. With the support of only a standalone ranking classifier, our proposed method still gives an overall performance approaching state-of-the-art, showing great potential.

References

Bajaj, P.; Campos, D.; Craswell, N.; Deng, L.; Gao, J.; Liu, X.; Majumder, R.; McNamara, A.; Mitra, B.; Nguyen, T.; Rosenberg, M.; Song, X.; Stoica, A.; Tiwary, S.; and Wang, T. 2018. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. *arXiv preprint arXiv:1611.09268* .

Bauer, L.; Wang, Y.; and Bansal, M. 2018. Commonsense for Generative Multi-Hop Question Answering Tasks. In *Empirical Methods in Natural Language Processing (EMNLP)*, 4220–4230.

Chen, D.; Fisch, A.; Weston, J.; and Bordes, A. 2017. Reading Wikipedia to Answer Open-Domain Questions. In *Association for Computational Linguistics (ACL)*, 1870–1879.

Cho, K.; van Merriënboer, 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 *Empirical Methods in Natural Language Processing (EMNLP)*, 1724–1734.

Clark, C.; and Gardner, M. 2018. Simple and Effective Multi-Paragraph Reading Comprehension. In *Association for Computational Linguistics (ACL)*, 845–855.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NACCL-HLT)*, 4171–4186.

Dunn, M.; Sagun, L.; Higgins, M.; Guney, V. 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* .

Glass, M.; Gliozzo, A.; Chakravarti, R.; Ferritto, A.; Pan, L.; Bhargav, G. P. S.; Garg, D.; and Sil, A. 2019. Span Selection Pre-training for Question Answering. *arXiv preprint arXiv:1909.04120* .

He, W.; Liu, K.; Liu, J.; Lyu, Y.; Zhao, S.; Xiao, X.; Liu, Y.; Wang, Y.; Wu, H.; She, Q.; Liu, X.; Wu, T.; and Wang, H. 2018. DuReader: a Chinese Machine Reading Comprehension Dataset from Real-world Applications. In *Proceedings of the Workshop on Machine Reading for Question Answering*, 37–46.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Comput.* 9(8): 1735–1780. ISSN 0899-7667. Place: Cambridge, MA, USA Publisher: MIT Press.

Hu, M.; Peng, Y.; Huang, Z.; and Li, D. 2019. A Multi-Type Multi-Span Network for Reading Comprehension that Requires Discrete Reasoning. 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)*, 1596–1606.

Indurthi, S. R.; Yu, S.; Back, S.; and Cuayáhuitl, H. 2018. Cut to the Chase: A Context Zoom-in Network for Reading Comprehension. In *Empirical Methods in Natural Language Processing (EMNLP)*, 570–575.

Joshi, M.; Choi, E.; Weld, D.; and Zettlemoyer, L. 2017. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In *Association for Computational Linguistics (ACL)*, 1601–1611.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations (ICLR)*.

Kočiský, T.; Schwarz, J.; Blunsom, P.; Dyer, C.; Hermann, K. M.; Melis, G.; and Grefenstette, E. 2018. The NarrativeQA Reading Comprehension Challenge. *Transactions of the Association for Computational Linguistics (TACL)* 6: 317–328.

Kudo, T.; and Richardson, J. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 66–71.

Lan, Z.; Chen, M.; Goodman, S.; Gimpel, K.; Sharma, P.; and Soricut, R. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *International Conference on Learning Representations (ICLR)*.

Lin, B. Y.; Chen, X.; Chen, J.; and Ren, X. 2019. KagNet: Knowledge-Aware Graph Networks for Commonsense Reasoning. 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)*, 2829–2839.

Lin, C.-Y. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*, 74–81.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692* .

Manning, C. D.; Surdeanu, M.; Bauer, J.; Finkel, J.; Bethard, S. J.; and McClosky, D. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, 55–60.

Nishida, K.; Saito, I.; Nishida, K.; Shinoda, K.; Otsuka, A.; Asano, H.; and Tomita, J. 2019. Multi-style Generative Reading Comprehension. In *Association for Computational Linguistics (ACL)*, 2273–2284.

Ohsugi, Y.; Saito, I.; Nishida, K.; Asano, H.; and Tomita, J. 2019. A Simple but Effective Method to Incorporate Multi-turn Context with BERT for Conversational Machine Comprehension. In *Proceedings of the First Workshop on NLP for Conversational AI*, 11–17.

Ott, M.; Edunov, S.; Baevski, A.; Fan, A.; Gross, S.; Ng, N.; Grangier, D.; and Auli, M. 2019. fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Association for Computational Linguistics (ACL)*, 311–318.

Rajpurkar, P.; Jia, R.; and Liang, P. 2018. Know What You Don't Know: Unanswerable Questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 784–789.

Reddy, S.; Chen, D.; and Manning, C. D. 2019. CoQA: A Conversational Question Answering Challenge. *Transactions of the Association for Computational Linguistics (TACL)* 7: 249–266.

See, A.; Liu, P. J.; and Manning, C. D. 2017. Get To The Point: Summarization with Pointer-Generator Networks. In *Association for Computational Linguistics (ACL)*, 1073–1083.

Seo, M.; Kembhavi, A.; Farhadi, A.; and Hajishirzi, H. 2017. Bidirectional Attention Flow for Machine Comprehension. In *International Conference on Learning Representations (ICLR)*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, 3104–3112.

Tan, C.; Wei, F.; Yang, N.; Du, B.; Lv, W.; and Zhou, M. 2018. S-Net: From Answer Extraction to Answer Synthesis for Machine Reading Comprehension. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Tay, Y.; Luu, A. T.; and Hui, S. C. 2018. Multi-Granular Sequence Encoding via Dilated Compositional Units for Reading Comprehension. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2141–2151.

Tay, Y.; Luu, A. T.; Hui, S. C.; and Su, J. 2018. Densely Connected Attention Propagation for Reading Comprehension. In *Advances in Neural Information Processing Systems (NIPS)*, 4906–4917.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems (NIPS)*, 5998–6008.

Wang, S.; Yu, M.; Guo, X.; Wang, Z.; Klinger, T.; Zhang, W.; Chang, S.; Tesauro, G.; Zhou, B.; and Jiang, J. 2018a. R3: Reinforced Ranker-Reader for Open-Domain Question Answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Wang, Y.; Liu, K.; Liu, J.; He, W.; Lyu, Y.; Wu, H.; Li, S.; and Wang, H. 2018b. Multi-Passage Machine Reading Comprehension with Cross-Passage Answer Verification. In *Association for Computational Linguistics (ACL)*, 1918–1927.

Yan, M.; Xia, J.; Wu, C.; Bi, B.; Zhao, Z.; Zhang, J.; Si, L.; Wang, R.; Wang, W.; and Chen, H. 2019. A Deep Cascade Model for Multi-Document Reading Comprehension. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 7354–7361.

Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J.; Salakhutdinov, R. R.; and Le, Q. V. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Advances in Neural Information Processing Systems (NIPS)*, 5754–5764.

Yatskar, M. 2019. A Qualitative Comparison of CoQA, SQuAD 2.0 and QuAC. In *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NACCL-HLT)*, 2318–2323.

Zhang, S.; Zhao, H.; Wu, Y.; Zhang, Z.; Zhou, X.; and Zhou, X. 2020a. DCMN+: Dual co-matching network for multi-choice reading comprehension. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9563–9570.

Zhang, Z.; Wu, Y.; Zhao, H.; Li, Z.; Zhang, S.; Zhou, X.; and Zhou, X. 2020b. Semantics-aware BERT for language understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 9628–9635.

Zhang, Z.; Wu, Y.; Zhou, J.; Duan, S.; Zhao, H.; and Wang, R. 2020c. SG-Net: Syntax-Guided Machine Reading Comprehension. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zhang, Z.; Yang, J.; and Zhao, H. 2020. Retrospective Reader for Machine Reading Comprehension. *arXiv preprint arXiv:2001.09694* .

Zhang, Z.; Zhao, H.; and Wang, R. 2020. Machine Reading Comprehension: The Role of Contextualized Language Models and Beyond. *arXiv preprint arXiv:2005.06249* .

Zhu, P.; Zhao, H.; and Li, X. 2020. Dual multi-head co-attention for multi-choice reading comprehension. *arXiv preprint arXiv:2001.09415* .