A Unified Query-based Generative Model for Question Generation and Question Answering

Linfeng Song[1], **Zhiguo Wang**[2] and **Wael Hamza**[2]

[1]Department of Computer Science, University of Rochester, Rochester, NY 14627

[2]IBM T.J. Watson Research Center, Yorktown Heights, NY 10598

Abstract

We propose a query-based generative model for solving both tasks of question generation (QG) and question answering (QA). The model follows the classic encoder-decoder framework. The encoder takes a passage and a query as input then performs query understanding by matching the query with the passage from multiple perspectives. The decoder is an attention-based Long Short Term Memory (LSTM) model with copy and coverage mechanisms. In the QG task, a question is generated from the system given the passage and the target answer, whereas in the QA task, the answer is generated given the question and the passage. During the training stage, we leverage a policy-gradient reinforcement learning algorithm to overcome exposure bias, a major problem resulted from sequence learning with cross-entropy loss. For the QG task, our experiments show higher performances than the state-of-the-art results. When used as additional training data, the automatically generated questions even improve the performance of a strong extractive QA system. In addition, our model shows better performance than the state-of-the-art baselines of the generative QA task.

Introduction

Recently both question generation and question answering tasks are receiving increasing attention from both the industrial and academic communities. The task of question generation (QG) is to generate a fluent and relevant question given a passage and a target answer, while the task of question answering (QA) is to generate a correct answer given a passage and a question. Both tasks have massive industrial values: QA has been used in industrial products such as search engines, while QG is helpful for improving QA systems by automatically increasing the training data. It can also be used to generate questions for educational purposes such as language learning.

For the QG task, existing work either entirely ignores the target answer (Du, Shao, and Cardie, 2017) while generating the corresponding question, or directly hard-codes the answer positions into the passage (Zhou et al., 2017; Yang et al., 2017; Subramanian et al., 2017; Tang et al., 2017;

Wang, Yuan, and Trischler, 2017; Yuan et al., 2017), so that sequence-to-sequence model (Sutskever, Vinyals, and Le, 2014) can be simply utilized. These methods only highlight the answer positions, but neglect other potential interactions between the passage and the target answer. In addition, this kind of methods will shrivel when the target answer does not occur in the passage verbatim. For the QA task, most of the existing literatures (Wang and Jiang, 2016; Wang et al., 2016; Shen et al., 2016; Wang et al., 2017; Chen et al., 2017; Xiong, Zhong, and Socher, 2016; Seo et al., 2016; Lee et al., 2016; Yu et al., 2016; Dhingra et al., 2017) focus on the extractive QA scenario, where they assume that the target answer occurs in the passage verbatim. The task then is to extract a span of consecutive words from the passage as the final answer. However, these methods may not work well on the generative QA scenario, where the correct answer is not a span in the given passage.

In this paper, we cast both the QG and QA tasks into one process by firstly matching the input passage against the query, then generating the output according to the matching results. Our model follows the classic encoder-decoder framework, where the encoder takes a passage and a query as input, then performs query understanding by matching the query with the passage from multiple perspectives, and the decoder is an attention-based LSTM model with copy (Gulcehre et al., 2016; Gu et al., 2016; See, Liu, and Manning, 2017) and coverage (Tu et al., 2016; Mi et al., 2016) mechanisms. Here a perspective is a way of matching the query and the passage. In the QG task, the input query is the target answer, and the decoder generates a question for the target answer, whereas in the QA task, the input query is a question, and the decoder generates the corresponding answer. To the best of our knowledge, there is no existing work dealing with both tasks using the same framework. In the QG task, we are the first to investigate query understanding before generating questions. By matching the target answer against the passage from multiple perspectives, our model captures more interactions between the answer and the passage, so that it can generate more precise question for the answer. Moreover, our model does not require that the answer literally occurs in the passage. In the QA task, our model generates answers word by word, and it has the capacity to generate answers that do not literally occur in the passage. Therefore, it naturally works for the generative QA scenario.

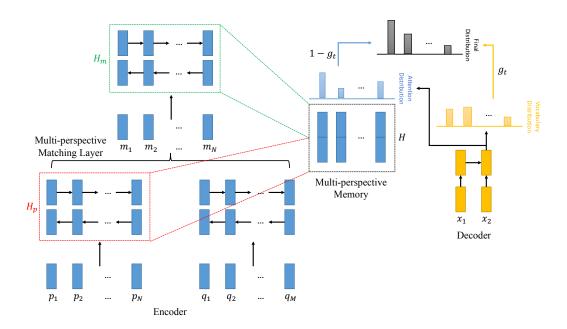arXiv:1709.01058v2 [cs.CL] 28 Aug 2018

Figure 1: Model overview.

We first pretrain the model with the cross-entropy loss, then fine tune with policy-gradient reinforcement learning to alleviate the exposure bias problem, resulting from sequence learning with the cross-entropy loss. In our policy-gradient reinforcement learning algorithm, we adopt a similar sampling strategy as the scheduled sampling strategy (Bengio et al., 2015) for generating the sampled output. We perform experiments on the SQuAD dataset (Rajpurkar et al., 2016) for the QG task, and on the "description" subset of the MS-MARCO (Nguyen et al., 2016) dataset for the generative QA task. Experimental results on the QG task show that our model outperforms previous state-of-the-art methods, and the automatically generated questions can even improve an extractive QA system. For the generative QA task, our model shows better performance than other generative systems.

Model

Figure 1 shows the architecture of our model. The model takes two components as input: a passage $P = (p_1, ..., p_j, ..., p_N)$ of length N, and a query $Q = (q_1, ..., q_i, ..., q_M)$ of length M, then generates the output sequence $X = (x_1, ..., x_L)$ word by word. Specifically, the model follows the encoder-decoder framework. The encoder matches each time-step of the passage against all time-steps of the query from multiple perspectives, and encodes the matching result into a "Multi-perspective Memory". In addition, the decoder generates the output sequence one word at a time based on the "Multi-perspective Memory".

Multi-Perspective Matching Encoder

The left-hand side of Figure 1 depicts the architecture of our encoder. Its goal is to perform comprehensive understanding of the query and the passage. The encoder first represents all

words within the passage and the query with word embeddings (Mikolov et al., 2013). In order to incorporate contextual information into the representation of each time-step of the passage or the query, we utilize a bi-directional LSTM (BiLSTM) (Hochreiter and Schmidhuber, 1997) layer to encode the passage and the query individually:

$$\overleftarrow{h_i^q} = \text{LSTM}(\overleftarrow{h_{i+1}^q}, q_i)$$
$$\overrightarrow{h_i^q} = \text{LSTM}(\overrightarrow{h_{i-1}^q}, q_i)$$
$$\overleftarrow{h_j^p} = \text{LSTM}(\overleftarrow{h_{j+1}^p}, p_j)$$
$$\overrightarrow{h_j^p} = \text{LSTM}(\overrightarrow{h_{j-1}^p}, p_j),$$

where q_i and p_j are embedding of the i-th word in the query and the j-th word in the passage. Then, the contextual vectors for each time-step of the query and the passage are constructed by concatenating the outputs from the BiLSTM layer: $h_i^q = [\overleftarrow{h_i^q}; \overrightarrow{h_i^q}]$ and $h_j^p = [\overleftarrow{h_j^p}; \overrightarrow{h_j^p}]$.

We utilize a matching layer on top of the contextual vectors to match each time-step of the passage with all time-steps of the query. Apparently, this is the most crucial layer in our encoder. Inspired by Wang, Hamza, and Florian (2017), we adopt the multi-perspective matching method for the matching layer. We define four matching strategies, as shown in Figure 2, to match the passage with the query from multiple granularities.

(1) **Full-Matching**. As shown in Figure 2 (a), each forward (or backward) contextual vector of the passage is compared with the last time-step of the forward (or backward) representation of the query.

(2) **Maxpooling-Matching**. As shown in Figure 2 (b), each forward (or backward) contextual vector of the passage is compared with every forward (or backward) contextual
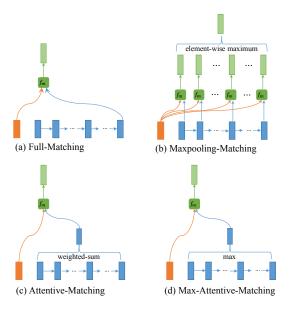
(a) Full-Matching   (b) Maxpooling-Matching

(c) Attentive-Matching   (d) Max-Attentive-Matching

Figure 2: Diagrams for different matching strategies, where f_m is a matching function between two vectors. The inputs include the contextual vector of one time-step of the passage (left orange block) and the contextual vectors of all time-steps of the query (right blue blocks). The output is a vector of matching values (top green block) calculated via f_m.

vectors of the query, and only the maximum value of each dimension is retained.

(3) **Attentive-Matching**. As shown in Figure 2 (c), we first calculate the cosine similarities between each forward (or backward) contextual vector of the passage and every forward (or backward) contextual vector of the question. Then, we take the cosine similarities as the weights, and calculate an attention vector for the entire query by computing a weighted sum of all the contextual vectors of the query. Finally, we match each forward (or backward) contextual vector of the passage with its corresponding attentive vector.

(4) **Max-Attentive-Matching**. As shown in Figure 2 (d), this strategy is similar to the Attentive-Matching strategy. However, instead of taking the weighed sum of all the contextual vectors as the attentive vector, we pick the contextual vector with the highest cosine similarity as the attentive vector. Then, we match each contextual vector of the passage with its new attentive vector.

These four match strategies require a function f_m to match two vectors. Theoretically, any functions for matching two vectors would work here. Inspired by Wang et al. (2016), we adopt the multi-perspective cosine matching function defined as:

$$m = f_m(v_1, v_2; \mathbf{W}),$$

where v_1 and v_2 are d-dimensional input vectors, $\mathbf{W} \in \mathbb{R}^{l \times d}$ is the learnable parameter of multi-perspective weight, and l is the number of perspectives. Each row $W_k \in \mathbf{W}$ represents the weights associated with one perspective, and the

similarity according to that perspective is defined as:

$$m_k = \cos(W_k \circ v_1, W_k \circ v_2),$$

where $\circ$ is the element-wise multiplication operation. So $f_m(v_1, v_2; \mathbf{W})$ represents the matching results between v_1 and v_2 from all perspectives. Intuitively, each perspective calculates the cosine similarity between two input vectors, and it is associated with a weight vector trained to highlight different dimensions of the input vectors. This can be regarded as considering different part of the semantics captured in the vector.

The final matching vector m_j for each time-step of the passage is the concatenation of the matching results of all four strategies. We also employ another BiLSTM layer on top of the matching layer to smooth the matching results. We concatenate the contextual vectors, h_j^p, of the passage and matching vectors to be the *Multi-perspective Memory* $\mathbf{H}$, which contains both the passage information and the matching information.

LSTM Decoder

The right-hand side of Figure 1 is our decoder. Basically, it is an attention-based LSTM model (Bahdanau, Cho, and Bengio, 2014) with copy and coverage mechanisms. The decoder takes the "Multi-perspective Memory" as the attention memory, and generates the output one word at a time.

Concretely, while generating the t-th word x_t, the decoder considers five factors as the input: (1) the "Multi-perspective Memory" $\mathbf{H} = \{h_0, ..., h_i, ..., h_N\}$, where each vector $h_i \in \mathbf{H}$ aligns to the i-th word in the passage; (2) the previous hidden state of the LSTM model s_{t-1}; (3) the embedding of previously generated word x_{t-1}; (4) the previous context vector c_{t-1}, which is calculated from the attention mechanism with $\mathbf{H}$ being the attentional memory; and (5) the previous coverage vector u_{t-1}, which is the accumulation of all attention distributions so far. When $t = 0$, we initialize s_{-1}, c_{-1} and u_{-1} as zero vectors, and fix x_{-1} to be the embedding of the sentence start token "<s>".

For each time-step t, the decoder first feeds the concatenation of the previous word embedding x_{t-1} and context vector c_{t-1} into the LSTM model to update the hidden state:

$$s_t = \text{LSTM}(s_{t-1}, [x_{t-1}, c_{t-1}])$$

Second, the attention distribution $\alpha_{t,i}$ for each time-step of the "Multi-perspective Memory" $h_i \in \mathbf{H}$ is calculated with the following equations:

$$e_{t,i} = v_e^T \tanh(W_h h_i + W_s s_t + W_u u_{t-1} + b_e)$$
$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^{N} \exp(e_{t,j})}$$

where W_h, W_s, W_v, b_e and v_e are learnable parameters. The coverage vector u_t is then updated by $u_t = u_{t-1} + \alpha_t$. And the new context vector c_t is calculated via:

$$c_t = \sum_{i=1}^{N} \alpha_{t,i} h_i$$

Then, the output probability distribution over a vocabulary of words at the current state is calculated by:

$$P_{vocab} = \text{softmax}(V_2(V_1[s_t, c_t] + b_1) + b_2),$$

where V_1, V_2, b_1 and b_2 are learnable parameters. The number of rows in V_2 represents the number of words in the vocabulary.

On top of the LSTM decoder, we adopt the copy mechanism (Gulcehre et al., 2016; Gu et al., 2016; See, Liu, and Manning, 2017) to integrate the attention distribution into the final vocabulary distribution. The probability distribution is defined as the interpolation between two probability distributions:

$$P_{final} = g_t P_{vocab} + (1 - g_t) P_{attn},$$

where g_t is the switch for controlling generating a word from the vocabulary or directly copying it from the passage. P_{vocab} is the generating probability distribution as defined above, and P_{attn} is calculated based on the attention distribution α_t by merging probabilities of duplicated words. Intuitively, g_t is relevant to the current decoder state, the attention results and the input. Therefore, inspired by See, Liu, and Manning (2017), we define it as:

$$g_t = \sigma(w_c^T c_t + w_s^T s_t + w_x^T x_{t-1} + b_g),$$

where vectors w_c, w_s, w_x and scalar b_g are learnable parameters.

Policy Gradient Reinforcement Learning via Scheduled Sampling

A common way of training a sequence generation model is to optimize the log-likelihood of the gold-standard output sequence $Y^* = y_0^*, ..., y_t^*, ..., y_T^*$ with the cross-entropy loss:

$$l_{ce} = -\sum_{t=1}^{T} \log p(y_t^* | y_{t-1}^*, ..., y_0^*, X; \theta),$$

where X is the model input, and θ represents the trainable model parameters.

However, this method suffers from two main issues. First, during the training stage, the ground-truth of the previous word y_{t-1}^* is taken as the input to predict the probabilities of the next word y_t. But, in the testing stage, the ground-truth y_{t-1}^* is not available, and the model has to rely on the previously generated word y_{t-1}. If the model selected a different y_{t-1} than the ground-truth y_{t-1}^*, then the following generated sequence could deviate from the gold-standard sequence. This issue is known as the "exposure bias problem". Second, models trained with the cross-entropy loss are optimized for the log-likelihood of a sequence which is different from the evaluation metrics.

In this work, we utilize a reinforcement learning method to address the exposure bias problem and directly optimize the evaluation metrics. Concretely, we adopt the "REINFORCE with a baseline" algorithm (Williams, 1992), a well-known policy-gradient reinforcement learning algorithm, to train our model, because it has shown the effectiveness for several sequence generation tasks (Paulus, Xiong, and

---

Data: gold-standard sequence Y^*
Data: greedy search sequence $\hat{Y}$
Result: sampled sequence Y^s
1   $S \leftarrow []$;
2   **for** i **in** range(len(Y^*)) **do**
3   **if** $i < \text{len}(\hat{Y})$ **then**
4   **if** random.random() $< p_{flip}$ **then**
5   S.append($\hat{Y}[i]$) ;
6   **else**
7   S.append($Y^*[i]$) ;
8   **end**
9   **else**
10   S.append($Y^*[i]$) ;
11   **end**
12 **end**

Algorithm 1: Scheduled Sampling Strategy

Socher, 2017; Rennie et al., 2016). Formally, the loss function is defined as:

$$l_{rl} = (r(\hat{Y}) - r(Y^s)) \sum_{t=1}^{T} \log p(y_t^s | y_{t-1}^s, ..., y_0^s, x; \theta),$$

where $Y^s = y_0^s, ..., y_T^s$ is the sampled sequence, $\hat{Y}$ is the sequence generated from a baseline, and the function $r(Y)$ is the reward calculated based on the evaluation metric. Intuitively, the loss function l_{rl} enlarges the log-probability of the sampled sequence Y^s, if Y^s is better than the baseline $\hat{Y}$ in terms of the evaluation metric $r(Y)$, or vice versa. In this work, for the QG task, we use the BLEU score (Papineni et al., 2002) as the reward, and for the QA task, we use the ROUGE score (Lin, 2004) as the reward.

Following Rennie et al. (2016), we take the greedy search result from the current model as the baseline sequence $\hat{Y}$. Rennie et al. (2016) generated the sampled sequence Y^s according to the probability distribution of $p(y_t^s | y_{t-1}^s, ..., y_1^s, x; \theta)$. However, this sampling strategy doesn't work well for our tasks. One possible reason is that our tasks have much larger search space. Inspired by Bengio et al. (2015), we designed a new "Scheduled Sampling" strategy to construct the sampled sequence Y^s from both the gold-standard sequence Y^* and the greedy search sequence $\hat{Y}$. As shown in Algorithm 1, it goes through the gold-standard sequence Y^* word by word (Line 2), and replaces with the corresponding word from the greedy search sequence $\hat{Y}$ with probability p_{flip} (Line 4-8). If the greedy search sequence is shorter than the gold-standard sequence, the ground-truth word is used after exceeding the end of the greedy search sequence (Line 10). Our experiments show that sampling the sequence according to the model distribution, as Rennie et al. (2016) does, usually produces outputs worse than the greedy search sequence, so it does not help very much. On the other hand, our sampling strategy usually generates better outputs than the greedy search sequence.

Experimental Setup

We conduct experiments on two tasks: question generation (QG) and generative question answering (QA).

Models	Split 1			Split 2
	BLEU-4	METEOR	ROUGE-L	BLEU-4
Du, Shao, and Cardie (2017)	12.28	16.62	39.75	–
Zhou et al. (2017)	–	–	–	13.29
w/o rich feature (baseline)	–	–	–	12.59(*)
MPQG	12.84	18.02	41.39	13.39
MPQG+R	**13.98**	**18.77**	**42.72**	**13.91**

Table 1: Results on question generation. *There is no published scores for Zhou et al. (2017) without the rich features, so we re-implemented their system and show the result.

Question Generation For the QG task, we evaluate the quality of generated questions with some automatic evaluation metrics such as BLEU (Papineni et al., 2002) and ROUGE (Lin, 2004), as well as their effectiveness in improving an extractive QA system. We conduct experiments on the SQuAD dataset (Rajpurkar et al., 2016) by comparing our model with Du, Shao, and Cardie (2017) and Zhou et al. (2017) in terms of BLEU, METEOR (Banerjee and Lavie, 2005) and ROUGE. The dataset contains 536 articles and over 100k questions related to the articles. Here, we follow Du, Shao, and Cardie (2017) and Zhou et al. (2017) to conduct experiments on the accessible part as our entire dataset. Since Du, Shao, and Cardie (2017) and Zhou et al. (2017) conducted their experiments using different training/dev/test split, we conduct experiments on both splits, and compare with their reported performance.

In addition, we also evaluate our model from a more practical aspect by examining whether the automatically generated questions are helpful for improving an extractive QA system. We use the data split of Du, Shao, and Cardie (2017), and conduct experiments on low-resource settings, where only (10%, 20%, or 50%) of the human-labeled questions in the training data are available. For example, in the 10% setting, we first train our QG model with the 10% available training data, then generate questions for the remaining 90% instances in the training data, where the human-labeled questions are abandoned.[1] Finally, we train an extractive QA system with the 10% human-labeled questions and the 90% automatically generated questions. The extractive QA system we choose is Wang et al. (2016), but our framework does not make any assumptions about the extractive QA systems being used.

Generative QA For this task, we conduct experiments on the MS MARCO dataset (Nguyen et al., 2016), which contains around 100k queries and 1M passages. The design purpose of this dataset is to generate the answer given the top 10 returned documents from a search engine, where the answer is not necessary in the documents. Even though the answers in this dataset are human generated rather than extracted from candidate documents, we found that the answers of around 66% questions can be exactly matched in the passage, and a large number of the remaining answers

just have a small difference with the content in the passages.[2] Among all types of questions ("numeric", "entity", "location", "person" and "description"), the "description" subset has the most percentage of answers that can not be exactly matched in the passage. Therefore, for the generative QA experiments, we follow Nguyen et al. (2016) to conduct experiments on the "description" subset, and compare with their reported results.

For both tasks, our model is first trained for 15 epochs with the cross-entropy loss, then fine-tuned for 15 epochs using our policy gradient algorithm. Adam (Kingma and Ba, 2014) is used for parameter optimization, and the learning rate is set to 0.005 and 0.0001 for cross entropy and policy gradient phases respectively. The encoder and decoder share the same pre-trained word embeddings, which are the 300-dimensional GloVe (Pennington, Socher, and Manning, 2014) word vectors pre-trained from the 840B common crawl corpus, and the embeddings are not updated during training. For all experiments, the flip probability p_{flip} is set to 0.1, the number of perspectives l is set to 5, and the weight for the coverage loss η is set to 0.1. For all experiments, the model yielding the best performance on the dev set is picked for evaluation on the test set.

Experimental Results

Question Generation

We compare our model with Du, Shao, and Cardie (2017) and Zhou et al. (2017) on the question generation task, and show the results in Table 1. Since Zhou et al. (2017) adopts rich features (such as named entity tags and part-of-speech tags), we re-implement a version without these rich features (w/o rich feature) for fair comparison. We also implement two versions of our model: (1) MPQG is our model only trained with the cross-entropy loss, and (2) MPQG+R is our model fine-tuned with the policy gradient reinforcement learning algorithm after pretraining.

First, our MPQG model outperforms the comparing systems on both data splits, which shows the effectiveness of our multi-perspective matching encoder. Our MPQG model, which only takes word features, shows even better performance than the feature-rich (with POS and NE tags) system of Zhou et al. (2017). Du, Shao, and Cardie (2017) utilized the sequence-to-sequence (Peng et al., 2016) model to take the passage as input and then generated the questions, where

---

[1] We assume the gold answers are available when generating questions for the remaining 90% instances, and leave automatic answer selection as future work, since the primary goal here is to evaluate the quality of automatically generated questions.

[2] They are generated by dropping or paraphrasing one span from the supporting sentence (containing answer) in the passage.

Passage: nikola tesla -lrb- serbian cyrillic : Никола Тесла ; 10 july [*1856*] – 7 january 1943 -rrb- was a serbian american inventor , electrical engineer , mechanical engineer , physicist , and futurist best known for his contributions to the design of the modern alternating current -lrb- ac -rrb- electricity supply system .
Target Answer: 1856
Reference: when was nikola tesla born ?
Baseline: when was nikola tesla 's inventor ?
MPQG: when was nikola tesla born ?
MPQG+R: when was nikola tesla born ?

Passage: zhéng -lrb- chinese : 正 -rrb- meaning " [*right*] " , " just " , or " true " , would have received the mongolian adjectival modifiers , creating " jenggis " , which in medieval romanization would be written " genghis " .
Target Answer: right
Reference: what does zhéng mean ?
Baseline: what are the names of the " jenggis " ?
MPQG: what does zhéng UNK mean ?
MPQG+R: what does zhéng mean ?

Passage: kenya is known for its [*safaris , diverse climate and geography , and expansive wildlife reserves*] and national parks such as the east and west tsavo national park , the maasai mara , lake nakuru national park , and aberdares national park .
Target Answer: safaris , diverse climate and geography , and expansive wildlife reserves
Reference: what is kenya known for ?
Baseline: what are the two major rivers that are known for the east and west tsavo national park ?
MPQG: what is kenya known for ?
MPQG+R: what is kenya known for ?

Table 2: Examples of generated questions. In each passage, the target answer is italic and is within brackets. The baseline is our implementation of Zhou et al. (2017) without rich features.

they entirely ignored the target answer. Therefore, the generated questions are independent of the target answer. Zhou et al. (2017) hard-coded the target answer positions into the passage, and employed the sequence-to-sequence model to consume the position-encoded passages, then generated the questions. This method only considered the target answer positions, but neglected the relations between the target answer and other parts of the passage. If the target answer does not literally occur in the passage, this method will shrivel. Conversely, our MPQG model matches the target answer against the passage from multiple perspectives. Therefore, it can capture more interactions between the target answer and the passage, and result in a more suitable question for the target answer.

Second, our MPQG+R model works better than the MPQG model on both splits, showing the effectiveness of our policy gradient training algorithm.

To better illustrate the advantage of our model, we show some comparative results of different models in Table 2, where the *Baseline* system is our implementation of Zhou et al. (2017) without rich features. Generally, our MPQG model generates better questions than Zhou et al. (2017). Taking the first case as an example, the baseline fails to recognize that "1856" is the year when "nikola tesla" is born, while our MPQG learns that from the pattern "day month year - day month year", which frequently occurs in the train-

ing data. For the third case, the baseline fails to generate the correct output as the query is very long and complicated. On the other hand, our MPQG model is able to capture that, because it performs comprehensive matching between the target answer and the passages. In addition, our MPQG+R model fixes some small mistakes of MPQG by directly optimizing the evaluation metrics, such as the second case in Table 2.

Question Generation for Extractive QA

Table 3 shows the results on improving an extractive QA system with automatically generated questions. Here F1 measures the overlap between the prediction and the reference in terms of bags of tokens, and exact match (EM) measures the percentage where the prediction is identical to the reference (Rajpurkar et al., 2016). The baseline is trained only on the part where gold questions are available, while the others are trained on the combination of the gold questions and the automatically generated questions, but with different methods of generating questions: (1) *w/ window*, a strong baseline from Yang et al. (2017), uses the previous and the following 5 tokens of the target answer as the pseudo question, and (2) *w/ MPQG+R* generates questions with our MPQG+R model.

First, we can see that *w/ MPQG+R* outperforms the baseline under all settings in terms of both F1 and EM scores, especially under the 10% setting, where we observe 3 and 5 points gains in terms of F1 and EM scores. This shows the effectiveness of our model. Second, the comparing results between *w/ MPQG+R* and *w/ window* show that the improvements of *w/ MPQG+R* are not due to simply enlarging the training data, but because of the higher quality of the generated questions. Yang et al. (2017) showed that *w/ window* can significantly improve their baseline, while it is not true in our experiment. One reason could be that our baseline is much stronger than theirs. For example, our system achieves 50.54% EM score under 10% setting, while theirs only got an EM score of 24.92%.

Generative QA

For the generative QA experiment, we compare our model with the generative models in Nguyen et al. (2016) on the "description" subset of MS-MARCO dataset. Table 4 shows the corresponding performance. Among all the comparing methods, *Best Passage* selects the best passage in terms of the ROUGE-L score, and obviously it accesses the reference. *Passage Ranking* ranks the passage by a deep structured semantic model of Huang et al. (2013). *Sequence to Sequence* is a vanilla sequence-to-sequence model (Sutskever, Vinyals, and Le, 2014). *Memory Network* adopts the end-to-end memory network (Sukhbaatar et al., 2015) as the encoder, and a vanilla RNN model as the decoder. We also implement a baseline system "vanilla-cosine", which only apply the vanilla cosine similarity for the matching function f_m in our encoder, and is only trained with the cross-entropy loss.

First, we can see that our *MPQG+R* model outperforms all other systems by a large margin, and is close to *Best Passage*, even though *Best Passage* accesses the reference.

Methods	F1			Exact Match (EM)		
	10%	20%	50%	10%	20%	50%
baseline	61.61	68.38	73.67	50.54	57.63	64.13
w/ window	61.23	66.80	73.35	50.48	56.31	64.00
w/ MPQG+R	**64.52**	**69.28**	**74.50**	**55.44**	**59.66**	**65.30**

Table 3: Results on improving extractive QA with automatically generated questions.

Models	ROUGE-L
Best Passage	35.1
Passage Ranking	17.7
Sequence to Sequence	8.9
Memory Network	11.9
vanilla-cosine	19.9
MPQG	31.5
MPQG+R	32.9

Table 4: Results on the "description" subset of MS-MARCO.

Besides, our *MPQG* model outperforms the *vanilla-cosine* model showing the effectiveness of our multi-perspective matching encoder. Finally, *MPQG+R* outperforms *MPQG* by around 1.4 ROUGE-L points, showing the effectiveness of our policy-gradient learning strategy.

Related Work

For question generation (QG), our work extends previous work (Du, Shao, and Cardie, 2017; Zhou et al., 2017; Yang et al., 2017; Subramanian et al., 2017; Tang et al., 2017; Wang, Yuan, and Trischler, 2017; Yuan et al., 2017) by performing query understanding. Tang et al. (2017); Yang et al. (2017) joins the QG task with the QA task, but they still conduct the QG task. The only difference is that they directly optimize the QA performance rather than a general metric (such as BLEU). On the other hand, our model can conduct both tasks of QG and QA.

For question answering (QA), most previous works (Wang and Jiang, 2016; Wang et al., 2016; Shen et al., 2016; Wang et al., 2017; Chen et al., 2017; Xiong, Zhong, and Socher, 2016; Seo et al., 2016; Lee et al., 2016; Yu et al., 2016; Dhingra et al., 2017) focus on the extractive QA scenario, which predicts a continuous span in the passage as the answer. Obviously, they rely on the assumption that the answer can be exactly matched in the passage. On the other hand, our model performs generative QA, which generates the answer word-by-word, and does not rely on this assumption. The generative QA is valuable for studying, as we can not guarantee the assumption being true for all scenarios. Tan et al. (2017) claims to perform generative QA, but it still relies on an extractive QA system by generating answers from the extractive results. One notable exclusion is Yin et al. (2015), which generate factoid answers from a knowledge base (KB). One significant difference is that their method matches the query against a KB, whereas ours performs matching against unstructured texts. Besides, we leverage policy gradient learning to alleviate the exposure bias problem, which they also suffer from.

Conclusion

In this paper, we introduced a query-based generative model, which can be used on both question generation and question answering. Following the encoder-decoder framework, a multi-perspective matching encoder is designed to perform query and passage understanding, and an LSTM model with coverage and copy mechanisms is leveraged as the decoder to generate the target sequence. In addition, we leverage a policy gradient learning algorithm to alleviate the exposure bias problem, which generative models suffer from when training with the cross-entropy loss. Experiments on both question generation and question answering tasks show superior performances of our model, which outperforms the state-of-the-art models. From the results we conclude that query understanding is important for question generation, and that policy gradient is effective on tackling the exposure bias problem resulted by sequence learning with cross-entropy loss. For the future work, we will consider adding adversarial data, which has been shown successful on plenty of areas(Peng et al., 2018).

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Banerjee, S., and Lavie, A. 2005. METEOR: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the ACL workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*.

Bengio, S.; Vinyals, O.; Jaitly, N.; and Shazeer, N. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In *NIPS 2015*, 1171–1179.

Chen, D.; Fisch, A.; Weston, J.; and Bordes, A. 2017. Reading wikipedia to answer open-domain questions. In *Proceedings of ACL 2017*.

Dhingra, B.; Liu, H.; Yang, Z.; Cohen, W.; and Salakhutdinov, R. 2017. Gated-attention readers for text comprehension. In *Proceedings of ACL 2017*.

Du, X.; Shao, J.; and Cardie, C. 2017. Learning to ask: Neural question generation for reading comprehension. *arXiv preprint arXiv:1705.00106*.

Gu, J.; Lu, Z.; Li, H.; and Li, V. O. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of ACL 2017*.

Gulcehre, C.; Ahn, S.; Nallapati, R.; Zhou, B.; and Bengio, Y. 2016. Pointing the unknown words. In *Proceedings of ACL 2017*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Huang, P.-S.; He, X.; Gao, J.; Deng, L.; Acero, A.; and Heck, L. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of CIKM 2013*, 2333–2338.

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Lee, K.; Salant, S.; Kwiatkowski, T.; Parikh, A.; Das, D.; and Berant, J. 2016. Learning recurrent span representations for extractive question answering. *arXiv preprint arXiv:1611.01436.*

Lin, C.-Y. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out: Proceedings of the ACL-04 workshop.* Barcelona, Spain.

Mi, H.; Sankaran, B.; Wang, Z.; and Ittycheriah, A. 2016. A coverage embedding model for neural machine translation. In *EMNLP 2016.*

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

Nguyen, T.; Rosenberg, M.; Song, X.; Gao, J.; Tiwary, S.; Majumder, R.; and Deng, L. 2016. MS MARCO: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268.*

Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of ACL 2002.*

Paulus, R.; Xiong, C.; and Socher, R. 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304.*

Peng, X.; Feris, R. S.; Wang, X.; and Metaxas, D. N. 2016. A recurrent encoder-decoder network for sequential face alignment. In *European conference on computer vision*, 38–56. Springer, Cham.

Peng, X.; Tang, Z.; Yang, F.; Feris, R. S.; and Metaxas, D. 2018. Jointly optimize data augmentation and network training: Adversarial data augmentation in human pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2226–2234.

Pennington, J.; Socher, R.; and Manning, C. 2014. Glove: Global vectors for word representation. In *Proceedings of EMNLP 2014*, 1532–1543.

Rajpurkar, P.; Zhang, J.; Lopyrev, K.; and Liang, P. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP 2016*, 2383–2392.

Rennie, S. J.; Marcheret, E.; Mroueh, Y.; Ross, J.; and Goel, V. 2016. Self-critical sequence training for image captioning. *arXiv preprint arXiv:1612.00563.*

See, A.; Liu, P. J.; and Manning, C. D. 2017. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368.*

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

Shen, Y.; Huang, P.-S.; Gao, J.; and Chen, W. 2016. Reasonet: Learning to stop reading in machine comprehension. *arXiv preprint arXiv:1609.05284.*

Subramanian, S.; Wang, T.; Yuan, X.; and Trischler, A. 2017. Neural models for key phrase detection and question generation. *arXiv preprint arXiv:1706.04560.*

Sukhbaatar, S.; Weston, J.; Fergus, R.; et al. 2015. End-to-end memory networks. In *NIPS 2015*, 2440–2448.

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

Tan, C.; Wei, F.; Yang, N.; Lv, W.; and Zhou, M. 2017. S-net: From answer extraction to answer generation for machine reading comprehension. *arXiv preprint arXiv:1706.04815.*

Tang, D.; Duan, N.; Qin, T.; and Zhou, M. 2017. Question answering and question generation as dual tasks. *arXiv preprint arXiv:1706.02027.*

Tu, Z.; Lu, Z.; Liu, Y.; Liu, X.; and Li, H. 2016. Modeling coverage for neural machine translation. *arXiv preprint arXiv:1601.04811.*

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

Wang, Z.; Mi, H.; Hamza, W.; and Florian, R. 2016. Multi-perspective context matching for machine comprehension. *arXiv preprint arXiv:1612.04211.*

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 ACL 2017.*

Wang, Z.; Hamza, W.; and Florian, R. 2017. Bilateral multi-perspective matching for natural language sentences. In *IJCAI 2017.*

Wang, T.; Yuan, X.; and Trischler, A. 2017. A joint model for question answering and question generation. *arXiv preprint arXiv:1706.01450.*

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

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

Yang, Z.; Hu, J.; Salakhutdinov, R.; and Cohen, W. W. 2017. Semi-supervised qa with generative domain-adaptive nets. *arXiv preprint arXiv:1702.02206.*

Yin, J.; Jiang, X.; Lu, Z.; Shang, L.; Li, H.; and Li, X. 2015. Neural generative question answering. *arXiv preprint arXiv:1512.01337.*

Yu, Y.; Zhang, W.; Hasan, K.; Yu, M.; Xiang, B.; and Zhou, B. 2016. End-to-end answer chunk extraction

and ranking for reading comprehension. *arXiv preprint arXiv:1610.09996*.

Yuan, X.; Wang, T.; Gulcehre, C.; Sordoni, A.; Bachman, P.; Subramanian, S.; Zhang, S.; and Trischler, A. 2017. Machine comprehension by text-to-text neural question generation. *arXiv preprint arXiv:1705.02012*.

Zhou, Q.; Yang, N.; Wei, F.; Tan, C.; Bao, H.; and Zhou, M. 2017. Neural question generation from text: A preliminary study. *arXiv preprint arXiv:1704.01792*.