Natural Answer Generation with Heterogeneous Memory

Anonymous ACL submission

Abstract

Memory augmented encoder-decoder framework has achieved promising progress for natural language generation tasks. Such frameworks enable a decoder to retrieve from a memory during generation. However, less research has been done to take care of the memory contents from different sources, which are often of heterogeneous formats. In this work, we propose a novel attention mechanism to encourage the decoder to actively interact with the memory by taking its heterogeneity into account. Our solution attends across the generated history and memory to explicitly avoid repetition, and introduce related knowledge to enrich our generated sentences. Experiments on the answer sentence generation task show that our method can effectively explore heterogeneous memory to produce readable and meaningful answer sentences while maintaining almost perfect coverage for given answer information.

1 Introduction

Most previous question answering systems focus on finding candidate words, phrases or sentence snippets from many resources, and ranking them for their users (Chu-Carroll et al., 2004; Xu et al., 2016). Typically, candidate answers are collected from different resources, such as knowledge base (KB) or textual documents, which are often with heterogeneous formats, e.g., KB triples or semi-structured results from Information Extraction (IE). For factoid questions, a single answer word or phrase is chosen as the response for users, as shown in Table 1 (A1).

However, in many real-world scenarios, users may prefer more natural responses rather than a single word. For example, as A2 in Table 1, *James Cameron directed the Titanic.* is more favorable than the single name *James Cameron.* A straightforward solution to compose an answer sentence is to build a template based model, where the answer word *James Cameron* and topic word in the question *the Titanic* are filled into a pre-defined template (Chu-Carroll et al., 2004). But such systems intrinsically lack variety, hence hard to generalize to new domains.

Q	Who is the director of the Titanic?
A1	**James Cameron**
A2	**James Cameron** directed **the Titanic**.
A3	**James Cameron** directed **it**.
A4	**James Cameron** directed it in **1999**.

Table 1: Answer sentences generated by different QA systems

To produce more natural answer sentences, (Yin et al., 2015) proposed GenQA, an encoder-decoder based model to select candidate answers from a KB styled memory during decoding to generate an answer sentence. CoreQA(He et al., 2017b) further extended GenQA with a copy mechanism to learn to copy words from the question. The application of attention mechanism enables those attempts to successfully learn sentence varieties from the memory and training data, such as usage of pronouns (A3 in Table 1). However, since they are within the encoder-decoder framework, they also encounter the well noticed repetition issue: due to loss of temporary decoder state, a RNN based decoder may repeat what has already been said during generation(Tu et al., 2016a,b).

Both GenQA and CoreQA are designed to work with a structured KB as the memory, while in most real-world scenarios, we require knowledge from different resources, hence of different formats. These knowledge may come from structured KBs, documents, or even tables. It is admittedly challenging to leverage a heterogeneous memory in a neural generation framework, and it is not well studied in previous works (Miller et al., 2016). Here in our case, the memory should

contain two main formats: KB triples and semi-structured entities from IE, forming a heterogeneous memory (HM). The former is usually organized in is a subject-predicate-object form, while, the latter is usually extracted from textual documents, in the form of keywords, sometimes associated with certain categories or tags oriented to specific tasks (Bordes and Weston, 2016).

Miller et al. (2016) discuss different knowledge representations for a simple factoid QA task, and show that classic structured KBs organized in a Key-Value Memory style work the best. However, dealing with heterogeneous memory is not trivial. Figure 1 shows an example of generating answer sentences from HM in a Key-Value style, which is indeed more challenging than only using a classic KB memory. Keys and values play different roles during decoding. A *director* key indicates this slot contains the answer. Same *James Cameron* values with different keys indicate duplication. The decoder needs these information to proactively perform memory addressing. Because keys from documents are not canonicalized, e.g., *doc directed* and *doc director*, they may lead to redundancy with the structured KB, e.g., *kb directed_by* and *doc director*. A decoder could repetively output a director twice simply because there are two different memory slots hit by the query, both indicating the same director. This will make the the repetition issue even worse.

Although many neural generation systems can produce coherent answer sentences, they often focus on how to guarantee the chosen answer words to appear in the output, while ignoring many related or meaningful background information in the memory that can further improve user experiences. In real-world applications like chatbots or personal assistants, users may want to know not only the exact answer word, but also information related to the answers or the questions. Theses information is potentially helpful to attract users' attention, and make the output sentences more natural. For example in Table 1 (A4), the extra *1999* not only enriches the answer with the movie's release year, but also can act as a clue to help distinguish ambiguous candidate answers, e.g., *Titanic (1999)* and *Titanic (HD, 2016)*.

In this paper, we propose a sequence to sequence model tailing for heterogeneous memory. In order to bridge the gap between decoder states and memory heterogeneity, we split decoder states into separate vectors, which can be used to address different memory components explicitly. To avoid redundancy, we propose the **Cumulative Attention** mechanism, which uses the context of the decoder history to address the memory, thus reduces repetition at memory addressing time. We conduct experiments on two `WikiMovies` datasets, and the experimental results show that our model is able to generate natural answer sentences composed with extra related facts about the question.

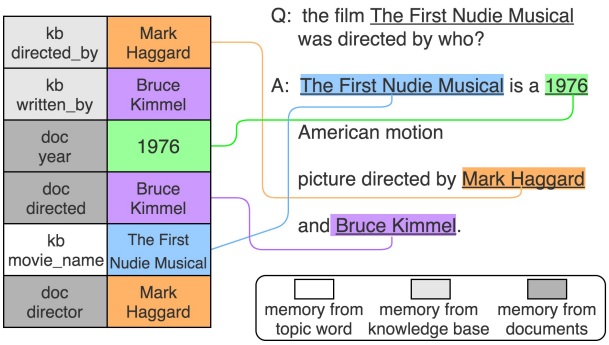

Figure 1: An example qa-pair with heterogeneous memory

2 Related Work

Natural Answer Generation with Sequence to Sequence Learning: Sequence to sequence models with attention have achieved successful results in many NLP tasks (Cho et al., 2014; Bahdanau et al., 2014; Vinyals et al., 2015; See et al., 2017). Memory is an effective way to equip seq2seq systems with external information (Weston et al., 2014; Sukhbaatar et al., 2015; Miller et al., 2016; Kumar et al., 2015). GenQA (Yin et al., 2015) apply seq2seq learning to generate natural answer sentences from a knowledge base, and CoreQA(He et al., 2017b) extend it with copying mechanism (Gu et al., 2016). But they do not consider the heterogeneity of the memory, only tackle questions with one single answer word, and do not study information enrichment.

Memory and Attention: Representation of memory and the interaction between decoder and memory (i.e. attention) is a continuous research topic. (Miller et al., 2016) propose Key-Value Memory to bridge the gap between KB and documents, but does not quite explore the attention mechanism. (Daniluk et al., 2017) split the decoder states into key and value representation, and increase language modeling performance. Multiple variants of attention mechanism have also been studied.

(Sukhbaatar et al., 2015) introduce multi-hop attention, and extend it to convolutional sequence to sequence learning(Gehring et al., 2017). (Kumar et al., 2015) further extend it by using a Gated Recurrent Unit (Chung et al., 2014) between hops. These models show that multiple hops may increase the model's ability to reason. These multi-hop attention is performed within a single homogeneous memory. Our Cumulative Attention is inspired by them, and we utilize it cross different memory, hence explicitly reason over different memory components.

Conditional Sentence Generation: Controllable sentence generation with external information is wildly studied from different views. From the task perspective, (Fan et al., 2017) utilize label information for generation, and tackle information coverage in a summarization task. (He et al., 2017a) use recursive Network to represent knowledge base, and (Bordes and Weston, 2016) track generation states and provide information enrichment, both are in a dialog setting. In terms of network architecture, (Wen et al., 2015) equip LSTM with a semantic control cell to improve informativeness of generated sentence. (Kiddon et al., 2016) propose the neural checklist model to explicitly track what has been mentioned and what left to say by splitting these two into different lists. Our model is related to these models with respect to information representation and challenges from coverage and redundancy. The most closely related one is the checklist model. But it does not explicitly study information redundancy. Also, the information we track is heterogeneous, and we track it in a different way, i.e. using Cumulative attention.

Due to loss of states across time steps, the decoder may generate duplicate outputs. Attempts have been made to address this problem. Some architectures try to utilize history attention records. (See et al., 2017) introduce a coverage mechanism, and (Paulus et al., 2017) use history attention weights to normalize new attention. Others are featured in network modules. (Suzuki and Nagata, 2017) estimate the frequency of target words and record the occurrence. Our model shows that simply attending to history decoder states can reduce redundancy. Then we use the context vector of attention to history decoder states to perform attention to the memory. Doing this enables the decoder to correctly decide what to say at memory addressing time, rather than decoding time, thus increasing answer coverage and information enrichment.

3 Task Definition

Given a question q and a memory M storing all related information that can be used to answer q, our task is to retrieve all the answer words from the memory, generate an answer sentence x, and properly introduce the rest information as enrichment.

Answer Coverage is the primary objective of our task. Since many answers contain multiple words, the system needs to cover all the target words.

Information Redundancy is a challenge of our task. It is well noticed that the decoder language model may lose track of its state, thus repeating itself. Also the decoder need to reason over the semantic gap between heterogeneous memory slots, figuring out different keys may refer to the same value. These two kinds of redundancy should both be addressed.

Information Enrichment is another challenge. It requires the decoder to interact with the memory effectively and use the right word to enrich the answer.

Tradeoff between redundancy and coverage/enrichment is one primary consideration. This is because when the decoder generate a word, it either generate a new word, or a mentioned word. The more answer words and information enrichment, the more likely the model to repeat what has already been generated. The goal is to push the decoder toward the direction of generating new words, rather than existing words.

4 Model

Our model consists of the question encoder, the heterogeneous memory, and the decoder. The encoder embeds the question into a vector representation. The decoder reads questions, retrieves the memory, and generates answer sentences.

We use a Long Short Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) for question encoding, and it encodes the question into an embedding q. It takes every word embedding $(q_1, q_2 ... q_n)$ of question words as inputs, and generates hidden states $hq_t = LSTM_{enc}(q_t, hq_{t-1})$. These hqs are later used for decoder's attention. The last hidden state hq_n is used as the vector rep-

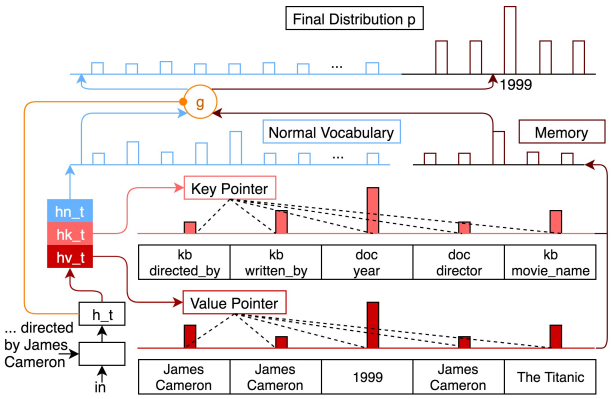

Figure 2: The Decoder with Heterogeneous States

resentation of the question, and is later put into the initial hidden state of the decoder.

We use a key-value memory M to represent the information heterogeneity. In our experiments, we study information from KB, topic words, and words extracted from documents. The memory is formated as $((mk_0, mv_0), (mk_1, mv_1)$ ... $(mk_n, mv_n))$, where mk_i and mv_i are respectively the key embedding and word embedding for the i-th memory slot. The vocabulary for keys V^{key} consists of all predicates in the KB, and all tags we use to classify the value words (e.g: director, actor, or release_year). The vocabulary for values V^{val} consists all related words from web documents, subjects and objects from the KB. This memory is later used in two ways: 1. the decoder uses its previous hidden state to perform attention and generate context vectors. 2. the decoder uses the updated hidden states as pointers (Vinyals et al., 2015) to retrieve the memory and copy the memory contents into the decoder's output.

4.1 Decoder with Heterogeneous States

As is in the standard encoder-decoder architecture with attention, the word embedding of the decoder's previous timestep x_t and context vector c_t is fed as the input of the next timestep, and the hidden state h_t is updated at the same time. The initial hidden state is the question embedding concatenated with average memory key and value

$$h_t = LSTM_{dec}(x_t, c_t, h_{t-1})$$
$$h_0 = [hq_n, avg(mk), avg(mv)]$$

where $[\cdot, \cdot]$ denotes concatenation.

As is shown in figure 2, to match the key-value memory representation, we use three linear tranformations to transform the decoder's current h_t into hn_t, hk_t, and hv_t.

$$\begin{aligned} hn_t &= W_n h_t \\ hk_t &= W_k h_t \\ hv_t &= W_v h_t \end{aligned}$$

These Ws are initialized as identity matrix $I = diag(1, 1...1)$. This bridges the decoder's semantic space with the memory's semantic space, and explicitly maintains heterogeneity.

hn_t is then projected to normal word vocabulary V^{norm} to form a distribution pn_t. hk_t and hv_t are used as pointers to perform attention to memory key Mk and value Mv respectively, and forms two distributions pmk_t and pmv_t. We use the average of the two as distribution over the memory: $pm_t = (pmk_t + pmv_t)/2$

The decoder then uses a gating mechanism $g = sigmoid(W_g h_t + b_g)$ to decide whether the output x_t comes from the normal vocabulary or the memory. The final distribution p can be formulated as:

$$\begin{aligned} p(x_t|q, M, x_i \forall i < t) = \\ gP(X_t = w_k|q, M, x_i \forall i < t) + \\ (1-g)P(X_t = m_k|q, M, x_i \forall i < t) \end{aligned} \quad (1)$$

where

$$P(X_t = w_k|q, M, x_i \forall i < t) = pn_t \text{ if } g \text{ else } 0$$
$$P(X_t = m_k|q, M, x_i \forall i < t) = pm_t \text{ if } !g \text{ else } 0$$

This is essentially a concatenation of distribution over normal vocabulary and distribution over the memory gated by the sigmoid random variable g, as is shown in the Figure 2.

The three hs are recorded for later decoder timestep to perform attention back.

4.2 Cumulative Attention

The Cumulative Attention mechanism is shown in figure 3. It is exploited similar to a multi-top fashion (Sukhbaatar et al., 2015). The difference is that the hops of attention are performed over different memories. The decoder first attend to its history hn_t, hk_t, and hv_t as is shown in the left part of figure 3.

$$\begin{aligned} c\_hn_t &= attn(h_{t-1}, hist\_hn_t) \\ c\_hk_t &= attn(h_{t-1}, hist\_hk_t) \\ c\_hv_t &= attn(h_{t-1}, hist\_hv_t) \end{aligned}$$

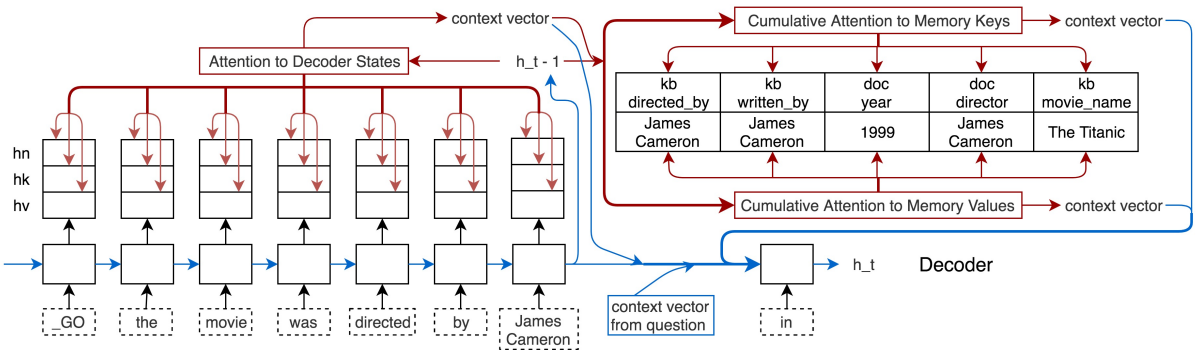

Figure 3: The Cumulative Attention Mechanism

where $attn(query, memory)$ denotes the attention function (Bahdanau et al., 2014) and

$$hist\_hn_t = (hn_0, hn_1, ...hn_{t-1})$$
$$hist\_hk_t = (hk_0, hk_1, ...hk_{t-1})$$
$$hist\_hv_t = (hv_0, hv_1, ...hv_{t-1})$$

Then the context vectors are concatenated $c\_h_t = [c\_hn_t, c\_hk_t, c\_hv_t]$, and used to perfrom attention to memory:

$$c\_mk_t = attn([h_{t-1}, c\_h_t], Mk)$$
$$c\_mv_t = attn([h_{t-1}, c\_h_t], Mv)$$

where $Mk = (mk_0, mk_1...mk_n)$ and $Mv = (mv_0, mv_1...mv_n)$ as is shown in the right part of the figure. The Cumulative attention scheme is an approach to bridge the heterogeneity of decoder states and memory. Also the decoder performs attention to the question, as is in the standard sequence to sequence attention model. All context vectors are concatenated, as well as the previous state h_{t-1} and output x_{t-1} to form the current input of decoder.

The decoder takes all the current timestep inputs, and generate a distribution p over normal vocabulary V^{norm} and the memory M under a gate g, as is shown in section 4.1. We choose cross-entropy as the loss function and optimize it with gradient descent based optimizers.

$$loss = crossent(p, \hat{p}) + crossent(g, \hat{g})$$

We use the outputs of equation 1 as answer sentences.

5 Experiments

Our experiments are designed to answer the following questions: (1) whether our model can properly deal with heterogeneous memory to generate readable answer sentences, (2) whether our model can cover all target answers, (3) whether our model can introduce related knowledge in the output while avoiding repetition.

5.1 Datasets

Our task requires a question, and a memory storing its answer words and related knowledge as input, and produce a **natural, readable** sentence as the output. Unfortunately, there is no existing dataset that naturally fits to our task. We thus tailor the `WikiMovies`[1] dataset according to our requirements. This `WikiMovies` dataset was originally constructed for answering simple factoid questions, using memory networks with different knowledge representations, i.e., structured KB (**KB entries** in Table 5.1), raw textual documents (**Doc**), or processed documents obtained through information extraction (**IE**), respectively. The first are classic structured knowledge in the subject-predicate-object format. The second contain sentences from Wikipedia. For each movie, there are also sentences automatically generated from a set of predefined templates. The third are in the subject-verb-object format, constructed by applying off-the-shell information extractor to all sentences.

As shown in Table 5.1, we treat each question in `WikiMovies` with its original answer (usually one or more words) as a QA pair, and one of the question's supportive sentences (either from Wikipedia or templates) as its gold-standard answer sentence. The memory for each question will contain all KB triples from **KB entries** for the topic movie in the question, and also entities and keywords extracted from its **IE** portion.

[1] http://fb.ai/babi

WikiMovies data format		
Question	Who directed the film Blade Runner?	
KB entries	Blade Runner *directed_by* Ridley Scott Blade Runner *release_year* 1982 Blade Runner *written_by* Philip K. Dick	
IE	*year* 1982 *starred* Harrison Ford	
Doc	Blade Runner is a **1982** American film directed by **Ridley Scott** and starring **Harrison Ford**. It is directed by **Ridley Scott** and written by **Philip K. Dick**. It comes out in **1982**.	
Answer	Ridley Scott	
Our data format		
Question	Who directed the film Blade Runner?	
Memory	*directed_by*	Ridley Scott
	release_year	1982
	written_by	Philip K. Dick
	movie	Blade Runner
	year	1982
	starred	Harrison Ford
Answer	Blade Runner is a **1982** American film directed by **Ridley Scott** and starring **Harrison Ford**.	

Table 2: Data format of WikiMovies dataset and our adaptation

Specifically, for **KB entries**, we use predicate as the key and object as value. For those from **IE**, we keep the tags as the key and entities or other expressions as the value. If an entity is not the answer, it is viewed as information enrichment. According to whether the supportive sentences are generated by templates or not, we split the dataset into WikiMovies-Synthetic and WikiMovies-Wikipedia.

The resulting WikiMovies-Synthetic includes 115 question patterns and 194 answer patterns, covering 10 main topics, e.g., director, genre, actor, release year, etc. We follow its original data split, i.e., 47,226 QA-pairs for training set, 8,895 for validation and 8,910 for testing.

In WikiMovies-Wikipedia, answer sentences are extracted from wikipedia, admittedly noisy in nature. Note that there are more than 10K Wikipedia sentences that are not paired with any questions, we thus left their questions as blank and treat it as a pure generation task to learn sentence variety. This also makes us not able to follow its original split, we therefore split the dataset randomly into 47,309 cases for training, 4,093 for testing and 3,954 for validation.

We treat normal words occurring less that 10 times as UNK, and, eventually, have 24,850 normal words and 37,898 entity words. We cut the maximum length for answer sentences to 20, and the maximum memory size to 10, which covers most cases in both synthetic and Wikipedia datasets.

5.2 Metrics

We evaluate our answer sentences in terms of answer **coverage**, information **enrichment**, and **redundancy**. For cases with only one answer word, we design $\mathbf{C}_{single}$ to indicate the percentage of cases being correctly answered. Cases with more than one answer word are evaluated by $\mathbf{C}_{part}$, i.e., percentage of answer words covered, and $\mathbf{C}_{perfect}$ is the percentage of cases whose answers are perfectly covered. Note that perfect coverage is the most difficult, while single coverage is the easiest one. For **Enrich**, we measure the number of none-answer memory items included in answer sentences. Regarding **Redundancy**, we calculate the times of repetition for memory values in the answer sentence. We also compute BLEU scores (Papineni et al., 2002) on the WikiMovies-Wikipedia, as an indicator of naturalness, to some extent.

5.3 Comparison Models

We compare our full model (HS-CumuAttn) with state-of-the-art answer generation models and constrained sentence generation models.

Our first baseline model is GenQA (Yin et al., 2015), a standard encoder-decoder model with attention mechanism. We equip it with our Key-Value style heterogeneous memory. We also compare with its two variants. HS-GenQA: we split its decoder state into heterogeneous representations. The other one, GenQA-AttnHist, is enhanced with a history attention during decoding.

CheckList (Kiddon et al., 2016) is the state-of-the-art model for generating long sentences with large agenda to mention. It keeps words that have been mentioned and words to mention with two separate records, and updates the records dynamically during decoding. To adapt to our task, we modify CheckList with a question encoder and a KV memory.

We also compare with one variant of our own model, HS-AttnHist, which does not benefit from the Cumulative Attention.

5.4 Implementation

Our model is implemented with the Tensorflow framework[2]. We use the Adam optimizer (Kingma and Ba, 2014) with default settings. The embedding dimension is set to be 256, as the LSTM state size. We set the batch size to 128 and train the model up to 80 epochs.

As mentioned, there is a tradeoff between Coverage/Enrichment and Redundancy. To set up a more fair comparison for different models, we ask the control group to reach a comparable level of **Redundancy**, i.e., approximately 0.11-0.12 on `WikiMovies-Synthetic` and 0.26-0.27 on `WikiMovies-Wikipedia`. Keeping **Redundancy** all most the same, we compare their Coverage and Enrichment.

5.5 Results and Discussion

Let us first look at the performance on the `Synthetic` set in Table 3. GenQA is originally proposed to read only one single fact during decoding, so it is not surprising it has the lowest answer coverage (38.92% $C_{perfect}$) and information enrichment (0.1535). After splitting the decoder state, HS-GenQA obtains significant improvement in both coverage (50.10% $C_{perfect}$) and enrichment (0.1952). When considering history for attention, GenQA-AttnHist achieves even better coverage (+3.% in C_{part} and +5% in $C_{perfect}$). By combining these two mechanisms, HS-AttnHist achieves the best perfect coverage, 51.55%. Although CheckList is not originally designed for our task, it still gives strong performance (50.04% $C_{perfect}$ and 0.1963 enrichment), at a lightly lower redundancy (0.1176). Finally, our full model, HS-CumuAttn, achieves the best single coverage 98.15%, and comparable partial/perfect coverage, with the lowest redundancy (0.0983). Due to the lower level of redundancy, HS-CumuAttn does not include as much enrichment as other strong models, but still outperforms GenQA.

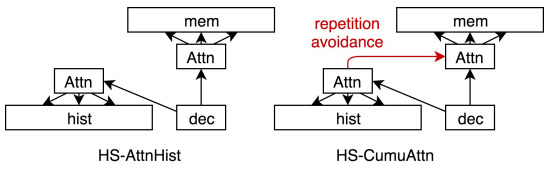

Figure 4: Two methods of using context of history to address the memory

[2]www.tensorflow.org

We further break down the contributions from different mechanisms. Compared to vanilla GenQA, HS-GenQA spilts the decoder states, thus improves the decoder's memory addressing process by performing attention separately, leading to improvements in both coverage and enrichment. Improvements of GenQA-AttnHist are of a different rationale. Looking at the history enables the decoder to avoid what are already said. Compared with HS-GenQA, GenQA-AttnHist improves Enrichment by avoiding repetition when introducing related information, while, HS-GenQA improves Enrichment by better memory addressing to select proper slots. Combining the two mechanisms together gives HS-AttnHist the best performance in Enrichment. However, HS-AttnHist still suffers from the repetition issue, to certain extent. Because when choosing memory content, there is no explicit mechanism to help the decoder to avoid repetitions according to the history (left of Figure 4). Therefore, a generated word may still be chosen again at the memory addressing step, leaving all the burden of avoiding repetition to the generation step. Our Cumulative Attention mechanism is designed to utilize the context vector of the history to address the memory, thus helps avoid choosing those already mentioned slots at memory addressing time (right of Figure 4), leading to almost the best coverage with the lowest redundancy.

Now we compare the three main models, GenQA, CheckList and our HS-CumuAttn, on `WikiMovies-Wikipedia` (Table 4), which is admittedly more challenging than `WikiMovies-Synthetic`. We skip the C_{single} metrics here, since most questions in `WikiMovies-Wikipedia` contain more than one answer word. It is not surprising that CheckList, with a lower redundancy, still outperforms GenQA in almost all metrics, except $C_{perfect}$, since CheckList is originally designed to perform well with larger agenda/memory and longer sentences. On the other hand, our model, HS-CumuAttn, achieves the best performance in all metrics. Although the BLEU score is not designed to fully reflect the naturalness, it still indicates that our model can output sentences that share more n-gram snippets with reference sentences and are more similar to those composed by humans.

Model	Redundancy	C_{single}	C_{part}	$C_{perfect}$	Enrich
GenQA	0.1109	91.25%	69.19%	38.92%	0.1535
HS-GenQA	0.1218	94.10%	76.47%	50.10%	0.1951
GenQA-AttnHist	0.1280	95.99%	73.44%	44.94%	0.1903
CheckList	0.1176	93.80%	76.32%	50.04%	0.1963
HS-AttnHist	0.1295	97.17%	**77.90%**	**51.55%**	**0.1996**
HS-CumuAttn	**0.0983**	**98.15%**	**77.28%**	50.79%	0.1665

Table 3: Results on the `WikiMovies-Synthetic` dataset

Model	BLEU	Redundancy	C_{part}	$C_{perfect}$	Enrich
GenQA	42.50	0.2603	62.80%	18.24%	0.5903
CheckList	43.69	0.2744	63.42%	18.23%	0.6094
HS-CumuAttn	**44.97**	**0.2385**	**64.06%**	**19.09%**	**0.6218**

Table 4: Results on the `WikiMovies-Wikipedia` dataset

Question	the movie Torn Curtain starred who?	
Memory	0 *actor*	Julie Andrews
	1 *starred_actors*	Julie Andrews
	2 *starred_actors*	Paul Newman
	3 *movie*	Torn Curtain
	4 *year*	1966
	5 *director*	Alfred Hitchcock
	6 *actor*	Paul Newman
GenQA	It stared (0 -> Julie Andrews) and (0 -> Julie Andrews) and and.	
CheckList	(3 -> Torn Curtain) is a (4 -> 1966) American film starring (2 -> Paul Newman) and (0 -> Julie Andrews) and (1 -> Julie Andrews).	
HS-CumuAttn	(3 -> Torn Curtain) is a (4 -> 1966) American political thriller film directed by (5 -> Alfred Hitchcock), starring (2 -> Paul Newman) and (0 -> Julie Andrews).	

Table 5: Case study

Case Study Table 5.5 provides an example question in our dataset, which expects two person names as the answer. We can see that GenQA may lose track of the decoder history, and repeat itself (*and and*), because there is not explicit mechanism to help it avoid repetition. Also it lacks informativeness and may not utilize other information stored in the memory. CheckList keeps records of what has been said and what is left to mention, thus has a good answer coverage. But its decoder is unable to explicitly address separate components within one memory slot, so it may not realize that the two *Julie Andrews*s are essentially the same person. HS-CumuAttn is able to find all the answer words properly and also include the *director* into the sentence. After generating *Paul Newman*, the Cumulative Attention mechanism enables the model to realize that *Paul Newman* in slot 2 has been said, and *Paul Newman* in slot 6 is the same as slot 2, so it should not choose the 6th slot again, and should move to *Julie Andrews*. Although the decoder may have the possibility to figure out the two *Paul Newman* are the same during decoding, Cumulative Attention can explicitly help make the clarification during memory addressing. Intuitively, the attention across memory and history induces a stronger signal for the decoder to gather the right information.

6 Conclusion and Future Work

In this paper, we propose a novel mechanism within an encoder-decoder framework to enable the decoder to actively interact with a memory by taking its heterogeneity into account. Our solution can read multiple memory slots form different sources, attend across the generated history and the memory to explicitly avoid repetition, and enrich our generated answer sentences with related information from the memory. In the future, we plan to extend our work through 1) utilizing larger and more complex memory such as knowledge graph, 2) answering more complex questions, such as questions involving deep reasoning over multiple facts.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR* abs/1409.0473.

Antoine Bordes and Jason Weston. 2016. Learning end-to-end goal-oriented dialog. *CoRR* abs/1605.07683. http://arxiv.org/abs/1605.07683.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR* abs/1406.1078. http://arxiv.org/abs/1406.1078.

Jennifer Chu-Carroll, Krzysztof Czuba, John M. Prager, Abraham Ittycheriah, and Sasha Blair-Goldensohn. 2004. Ibm's piquant ii in trec 2004. In *TREC*.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR* abs/1412.3555. http://arxiv.org/abs/1412.3555.

Michal Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel. 2017. Frustratingly short attention spans in neural language modeling. *CoRR* abs/1702.04521. http://arxiv.org/abs/1702.04521.

Angela Fan, David Grangier, and Michael Auli. 2017. Controllable abstractive summarization. *CoRR* abs/1711.05217. http://arxiv.org/abs/1711.05217.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning. *CoRR* abs/1705.03122. http://arxiv.org/abs/1705.03122.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. *CoRR* abs/1603.06393. http://arxiv.org/abs/1603.06393.

He He, Anusha Balakrishnan, Mihail Eric, and Percy Liang. 2017a. Learning symmetric collaborative dialogue agents with dynamic knowledge graph embeddings. *CoRR* abs/1704.07130. http://arxiv.org/abs/1704.07130.

Shizhu He, Cao Liu, Kang Liu, and Jun Zhao. 2017b. Generating natural answers by incorporating copying and retrieving mechanisms in sequence-to-sequence learning. pages 199–208.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

Chloé Kiddon, Luke S. Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *EMNLP*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980. http://arxiv.org/abs/1412.6980.

Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. 2015. Ask me anything: Dynamic memory networks for natural language processing. *CoRR* abs/1506.07285. http://arxiv.org/abs/1506.07285.

Alexander H. Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. *CoRR* abs/1606.03126. http://arxiv.org/abs/1606.03126.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, Stroudsburg, PA, USA, ACL '02, pages 311–318. https://doi.org/10.3115/1073083.1073135.

Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *CoRR* abs/1705.04304. http://arxiv.org/abs/1705.04304.

Abigail See, Peter Liu, and Christopher Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Association for Computational Linguistics*. https://arxiv.org/abs/1704.04368.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. MIT Press, Cambridge, MA, USA, NIPS'15, pages 2440–2448. http://dl.acm.org/citation.cfm?id=2969442.2969512.

Jun Suzuki and Masaaki Nagata. 2017. Cutting-off redundant repeating generations for neural abstractive summarization. In *EACL*.

Zhaopeng Tu, Yang Liu, Zhengdong Lu, Xiaohua Liu, and Hang Li. 2016a. Context gates for neural machine translation .

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016b. Modeling coverage for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, pages 76–85. https://doi.org/10.18653/v1/P16-1008.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. MIT Press, Cambridge, MA, USA, NIPS'15, pages 2692–2700. http://dl.acm.org/citation.cfm?id=2969442.2969540.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Association for Computational Linguistics, pages 1711–1721. https://doi.org/10.18653/v1/D15-1199.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2014. Memory networks. *CoRR* abs/1410.3916. http://arxiv.org/abs/1410.3916.

Kun Xu, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. Hybrid question answering over knowledge base and free text. In *COLING*.

Jun Yin, Xin Jiang, Zhengdong Lu, Lifeng Shang, Hang Li, and Xiaoming Li. 2015. Neural generative question answering. *CoRR* abs/1512.01337. http://arxiv.org/abs/1512.01337.