Mnemonic Reader for Machine Comprehension

Minghao Hu[*] **Yuxing Peng**
Department of Computer Science
National University of Defense Technology
{huminghao09,pengyuxing}@nudt.edu.cn

Xipeng Qiu
Department of Computer Science
Fudan University
xpqiu@fudan.edu.cn

Abstract

Recently, several end-to-end neural models have been proposed for machine comprehension tasks. Typically, these models use attention mechanisms to capture the complicated interaction between the context and the query and then point the boundary of answer. To better point the correct answer, we introduce the Mnemonic Reader for machine comprehension tasks, which enhance the attention reader in two aspects. Firstly, we use a self-alignment attention to model the long-distance dependency among context words, and obtain query-aware and self-aware contextual representation for each word in the context. Second, we use a memory-based query-dependent pointer to predict the answer, which integrates both explicit and implicit query information, such as query category. Our experimental evaluations show that our model obtains the state-of-the-art result on the large-scale machine comprehension benchmarks SQuAD.

1 Introduction

Benefiting from the rapid development of deep learning techniques (Goodfellow et al., 2016) and the large-scale benchmarks (Hermann et al., 2015; Hill et al., 2016; Rajpurkar et al., 2016), the end-to-end neural based methods have achieved promising results on machine comprehension tasks. Among the existing methods, various attention mechanisms (Bahdanau et al., 2014) play a vital role to establish the interaction between the query and the context, which allows a differentiable neural model to focus on attentive parts of context document in term of the query.

The attention mechanism is often used in two modules: the interaction layer and the pointer layer.

- In the interaction layer, the existing methods, such as Gated-Attention Reader (Dhingra et al., 2016), Multi-perspective Matching (Wang et al., 2016) and Dynamic Coattention Networks (Xiong et al., 2017), integrate the attentive information from query to construct the query-aware context representation. Such representation of each context word is expected to model its contextual information with respect to the entire context paragraph and the query. However, the existing models only adopt long short-term memory network (LSTM) (Hochreiter and Schmidhuber, 1997) or gated recurrent units (GRU) (Chung et al., 2014) to model the contextual information of contexts, therefore they actually just model the local contextual information due to the long term dependency problem.

- In the pointer layer, the pointer network (Vinyals et al., 2015) is used to predict the boundary of the answer. A neural pointer use attention mechanism again to assign a probability of being selected as start or end of an answer span to each word in context. However, most of the existing methods, such as Answer Pointer (Wang and Jiang, 2017) and Ruminating Reader (Gong and Bowman, 2017), focus on the query-aware context representation and use query-independent pointer vector to select the answer boundary. Intuitively, the pointer vector should be query

[*]Most of this work was done while MH was in Fudan University.

sensitive since the answer prediction depends on the query information, such as query category. Besides, one-hop pointing procedure may fail to fully understand the query and predict the wrong span. Therefore the pointing mechanism should also be multi-hop for refining the answer span iteratively.

In this paper, we propose the Mnemonic Reader for machine comprehension tasks, which enhances the attention reader in two aspects. Firstly, we improve the contextual representation of each word in the context with a self-alignment attention. The self-alignment attention can model the long-distance contextual information, which cannot be captured by the LSTM or GRU. Thus, the representation of each word in the context should contain the contextual information of the whole context passage and the query. Second, we enhance the answer pointer by introducing a query-sensitive pointing mechanism, which contains the implicit and explicit query information such as question type. Besides, we extend the one-hop answer pointer to a multi-hop architecture for continuing refining the answer span. The contributions of this paper can be summarized as follows.

1. The proposed mnemonic reader is a strategy to remember key terms within a text with extra information of context and query, which is helpful for deeper comprehension of text.

2. Specifically, we use two kinds of mnemonic information to improve attention reader: self-aware context representation for synthesizing contextual information and multi-hop query-sensitive answer pointer for predicting the answer span iteratively.

3. Our model obtains the state-of-the-art result on the large machine comprehension benchmarks SQuAD. Our single model obtains an F1 of 79.21% while our ensemble model obtains an F1 of 81.69%.

2 Related Work

Reading comprehension. The significant advance on reading comprehension has largely benefited from the availability of large-scale datasets. Richardson et al. (2013) released the MCTest data which was too small to train end-to-end neural models. Later large cloze-style datasets such as CNN/DailyMail (Hermann et al., 2015) and Childrens Book Test (Hill et al., 2016) were released, make it possible to solve RC tasks with deep neural architectures. The Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016) is a more recently released dataset with over 100k queries-context pairs. The biggest difference of SQuAD from previous cloze-style datasets is that the answer of SQuAD is a segment of text but not a single entity, which make it more difficult to solve.

Attention mechanism. From the perspective of attention mechanism, previous works for RC can be coarsely divided into two groups: *first-order attention* and *second-order attention*. First-order attention means that the model computes the attention in only one direction. The most popular approach is using a single fix-sized query vector to attend to contexts, and the computed attention is then either used for fusing the context into a single vector (Hermann et al., 2015; Chen et al., 2016) or acting as a ”pointer” (Vinyals et al., 2015) to indicate the position of the answer (Kadlec et al., 2016). Sordonif et al. (2016) produces the query vector and document vector by computing the first-order attention twice, followed by a multi-hop iteration.

Second-order attention means that the model computes the attention bidirectionally, which usually forms an alignment matrix containing similarity scores corresponding to all pairs of context words and query words. Cui et al. (2016) computes a query-level average attention based on the alignment matrix, which is then used to further compute a weighted sum of context-level attention. BiDAF (Seo et al., 2017) computes both the context-to-query attention and the query-to-context attention by using second-order attention. In contrast to these models, Mnemonic Reader uses second-order attention not only between the context and the query, but also inside the context, which can model the long-distance contextual information.

Reasoning mechanism. Based on how models perform the inference in RC, previous works can be divided into another two groups: *one-hop reasoning* and *multi-hop reasoning*. One-hop reasoning models use either first-order attention or second-order attention to emphasis relevant parts between the context and the query. The architecture of these models is quite shallow, usually containing only one interaction layer

(Weissenborn et al., 2017; Zhang et al., 2017).

Multi-hop reasoning models, on the other hand, try to simulate the phenomenon that human increase their understanding by reread contexts and queries. This is typically done by maintaining a memory state which incorporates the current information of reasoning with the previous information in the memory, called Memory Networks (Sukhbaatar et al., 2015; Kumar et al., 2015). ReasoNet (Shen et al., 2016) combines the memory network with reinforcement learning to dynamically determine when to stop reading. Dhingra et al. (2016) extends the attention-sum reader to multi-hop reasoning with a gating mechanism. Dynamic Coattention Networks (Xiong et al., 2017) utilizes a multi-hop pointing decoder to indicate the answer span iteratively. In contrast to these works, Mnemonic Reader contains a multi-hop query-sensitive pointing layer which is able to continue refining the answer span.

3 Model

Mnemonic Reader consists of four basic layers: *input layer*, *interactive alignment layer*, *self alignment layer* and *mnemonic pointing layer*, which are depicted in Figure 1(a). Below we discuss these layers in more details.

Input Layer: The input layer encodes each context and the corresponding query into distributed vector space.

Interactive Alignment Layer: Given a pair of encoded vector representations, the interactive alignment layer retrieves information among them to produce a query-aware context representation.

Self Alignment Layer: The self alignment layer aligns the query-aware context against itself to futher generate a self-aware context representation.

Mnemonic Pointing Layer: The mnemonic pointing layer predicts the start and end positions of the answer in a multi-hop manner, by using a query-sensitive answer pointer.

3.1 Input Layer

Embedding. In reading comprehension task, the context and query are both word sequences. The input layer is responsible for mapping each word x to its corresponding word embedding x_w, which is typically done by using pre-trained word vectors. At a more low-level granularity, the input layer also embeds each word x by encoding their character sequences with a convolutional neural network followed by max-pooling over time (Kim, 2014), resulting in a character-level embedding x_c. Each word embedding x is then represented as the concatenation of character-level embedding and word-level embedding, denoted as $x = [x_w; x_c] \in \mathbb{R}^d$, where d is the total dimensionality of word-level embedding and character-level embedding.

Similar to BiDAF (Seo et al., 2017), word embeddings of both query and context are further transformed by the Highway Network (Srivastava et al., 2015), resulting in two word embedding matrices: $X_Q = [x_1^q, x_2^q, ..., x_n^q] \in \mathbb{R}^{d\times n}$ for the query and $X_C = [x_1^c, x_2^c, ..., x_m^c] \in \mathbb{R}^{d\times m}$ for the context, where n and m denote the length of query and context respectively.

Besides the word embedding, we use a simple yet effective binary feature of exact matching as additional inputs. This binary feature indicates whether a word in context can be exactly matched to one query word, and vice versa for a word in query. We exclude those words which do not have specific meanings, such as "the", "a", "do", etc. Eq. 1 formally defines this feature:

$$\begin{aligned} \text{em}_i^q &= \mathbb{1}(\exists j : x_i^q = x_j^c) \\ \text{em}_j^c &= \mathbb{1}(\exists i : x_j^c = x_i^q) \end{aligned} \tag{1}$$

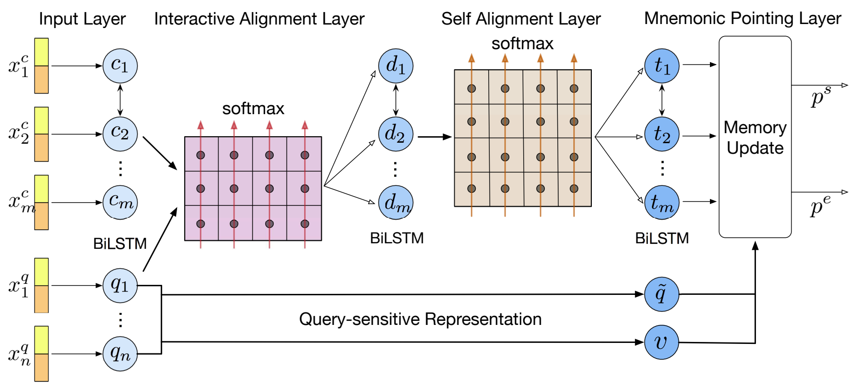
Encoding. In order to model the word sequence under its contextual information, we use a bidirectional long short-term memory network (BiLSTM) (Hochreiter and Schmidhuber, 1997) to encode both the context and the query as follows:

$$\begin{aligned} q_i &= \text{BiLSTM}([x_i^q; \text{em}_i^q]), \forall i \in [1, ..., n] \\ c_j &= \text{BiLSTM}([x_j^c; \text{em}_j^c]), \forall j \in [1, ..., m] \end{aligned} \tag{2}$$

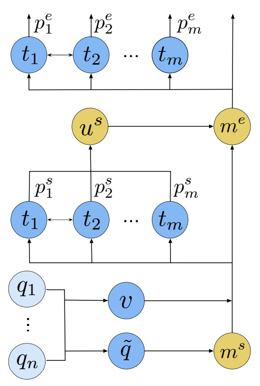
where q_i and $c_j \in \mathbb{R}^{2h}$ are the concatenated hidden states of two BiLSTMs for the i-th query word and the j-th context word, and h is the dimensionality of hidden state. Finally we obtain two encoding matrices: $Q = [q_1, q_2, ..., q_n] \in \mathbb{R}^{2h\times n}$ for the query and $C = [c_1, c_2, ..., c_m] \in \mathbb{R}^{2h\times m}$ for the context.

3.2 Interactive Alignment Layer

The interactive alignment layer attends to the query and context simultaneously so as to capture the interaction between them, and finally generates a query-aware context representation D. Such representation is constructed by fusing query information into the context. More specifically, we



(a) A high level overview of Mnemonic Reader.



(b) The detailed one-hop schematic of mnemonic pointing layer.

Figure 1: 1(a) shows the model overview. The interactive alignment layer fuses query information into context, while the self alignment layer exchanges information between each context word. 1(b) presents one hop of the mnemonic pointing layer. p^s is the estimated probability of the start position of the answer, generated by the query-memory vector m^s and the query-category vector v. u^s is an evidence vector for updating the new query-memory vector m^e. The new query memory m^e is then used to predict the end position of the answer p^e.

first compute an alignment matrix $A \in \mathbb{R}^{n \times m}$ between the query and the context by $A_{ij} = {q_i}^{\mathrm{T}} \cdot c_j$. The element A_{ij} of matrix A indicates the similarity between the i-th query word and the j-th context word.

For each context word, we intend to find the most relevant query word by computing a soft-aligned query vector, and fuse it back to the context word. Let $a_j \in \mathbb{R}^n$ denote the normalized attention distribution of query for the j-th context word, and $\tilde{Q}_{:j} \in \mathbb{R}^{2h}$ denote the corresponding soft-aligned query vector. The computation is defined in Eq. 3:

$$\begin{aligned} a_j &= \mathrm{softmax}(A_{:j}) \\ \tilde{Q}_{:j} &= Q \cdot a_j, \forall j \in [1, ..., m] \end{aligned} \tag{3}$$

Hence $\tilde{Q} \in \mathbb{R}^{2h \times m}$ contains the attended query vectors for the entire context.

To get the query-aware context representation, we combine the encoded representation of context C and the attended query vectors $\tilde{Q}$ by using an effective heuristics, and feed them into a BiLSTM for aggregation:

$$D = \mathrm{BiLSTM}([C; \tilde{Q}; C \circ \tilde{Q}; C - \tilde{Q}]) \tag{4}$$

where $D \in \mathbb{R}^{2h \times m}$ contains the contextual information of context with respect to attended query information, "$\circ$" stands for element-wise multiplication and "$-$" means the element-wise subtraction.

3.3 Self Alignment Layer

The self alignment layer aligns the query-aware context representation $D \in \mathbb{R}^{2h \times m}$ against itself to futher synthesize contextual information among context words. The insight is that the limited capability of recurrent neural networks make it difficult to model long-term dependencies. Hence, it is hard to answer questions that require synthesizing information from different part of contexts (Weissenborn et al., 2017). Fusing information between context words allows crucial contextual information to flow close to the correct answer.

Similar to the interactive alignment layer, we first compute a self alignment matrix $B \in \mathbb{R}^{m \times m}$ for the context:

$$B_{ij} = \mathbb{1}(i \neq j){d_i}^{\mathrm{T}} \cdot d_j \tag{5}$$

where B_{ij} indicates the similarity between the i-th context word and the j-th context word, and the diagonal of alignment matrix B is set to be zero in case of the word being aligned with itself.

Next, for each context word, we compute a self-aligned context vector $\tilde{D}_{:j} \in \mathbb{R}^{2h}$ in a similar way of Eq. 3. We first calculate the normalized context attention distribution $b_j \in \mathbb{R}^m$ for the j-th context word through $b_j = \text{softmax}(B_{:j})$. Then the corresponding self-aligned context vector can be computed as $\tilde{D}_{:j} = D \cdot b_j$. Hence, $\tilde{D} \in \mathbb{R}^{2h \times m}$ contains the attended context vectors for the entire query-aware context.

Finally, we use the same heuristics to combine query-aware context representation D and self-aligned context vectors $\tilde{D}$, and aggregate them through another BiLSTM:

$$T = \text{BiLSTM}([D; \tilde{D}; D \circ \tilde{D}; D - \tilde{D}]) \qquad (6)$$

where $T \in \mathbb{R}^{2h \times m}$ is the self-aware context representation, and $T_{:j}$ is expected to contain contextual information about the j-th word with respect to word-specific context and query information.

3.4 Mnemonic Pointing Layer

Unlike the cloze-style QA task in which the answer is a single token, complex QA tasks require a span of text as the final answer. The answer span (i, j) is usually decided by first predicting the start position i of answers and then the end position j, based on the implicit query representation:

$$p(i, j|C, Q) = p^s(i|C, Q) \cdot p^e(j, |i, C, Q) \qquad (7)$$

Moreover, the explicit query category (i.e., *who*, *when*, *where*, and so on) is obviously a high-level abstraction of the expression of queries, providing additional clues for searching the answer. For example, a "when" query paies more attention on temporal information while a "where" query seeks for spatial information.

Therefore, in order to utilize both the implicit and explicit query information, we propose a query-sensitive pointing mechanism to select an answer span. Firstly, the module attends over the self-aware context representation while taking into consideration a query-memory vector and a query-category vector to produce the estimate of the start position and an evidence vector. Next, the evidence vector is used, alongside the previous query-memory vector, to update the new query-memory vector, which is further used to produce the estimate of the end position. A detailed visualization of the layer is shown in Figure 1(b).

Query-sensitive representation. The query-sensitive representation consists of two parts: the *query-memory vector* and the *query-category vector*. The query-memory vector $m^s \in \mathbb{R}^{2h}$ for the start position of the answer is initialized as an implicit query summary $\tilde{q} \in \mathbb{R}^{2h}$: $m^s = \tilde{q}$. In our experiment we found that simply using the last hidden states of query representations Q as $\tilde{q}$ results in good performance.

To explicitly model the query category, we count the key word frequency of all queries and obtain top-9 most frequent query categories, which is similar in (Zhang et al., 2017): *what*, *how*, *who*, *when*, *which*, *where*, *why*, *be*, *other*, in which *be* stands for queries beginning with different forms of key word such as *is*, *am* and *are*. Each query category k is represented by a trainable embedding $x_k^v \in \mathbb{R}^{d_v}$, where d_v is the dimensionality of query-category embedding. The query-category embedding is then fed into a two-layer feedforward neural network which produces a query-category vector $v \in \mathbb{R}^{2h}$.

Prediction of the answer span. The probability distribution for the start position $p^s(i)$ is computed by using a pointer network (Vinyals et al., 2015):

$$\begin{aligned} s_i &= \text{FN}(\alpha(T_{:i}, m^s, v)) \\ p^s(i) &= \text{softmax}(w_s s_i) \end{aligned} \qquad (8)$$

where the abbreviation FN means a feedforward neural network that provides a non-linear mapping of its input. $w_s \in \mathbb{R}^h$ is a trainable weight vector, and α is a effective heuristic function that fuses its input vectors, which is defined as: $\alpha(t, m, v) = [t; m; v; t \circ m; t \circ v] \in \mathbb{R}^{10h}$

The normalized probability $p^s \in \mathbb{R}^m$ is also used as a first-order attention to aggregate information of candidate start positions, producing an evidence vector $u^s \in \mathbb{R}^{2h}$ for the start position: $u^s = T \cdot p^s$. The evidence vector and the query-memory vector are fed into a memory update function G, to produce a new query-memory vector $m^e \in \mathbb{R}^{2h}$ for the end position of the answer: $m^e = \text{G}(m^s, u^s)$.

Finally the probability distribution for the end position $p^e(j)$ is computed similarly as Eq. 8, by using m^e instead of m^s:

$$\begin{aligned} e_j &= \text{FN}(\alpha(T_{:j}, m^e, v)) \\ p^e(j) &= \text{softmax}(w_e e_j) \end{aligned} \qquad (9)$$

Memory update mechanism. The memory update function G accepts two inputs (the old query-memory vector m_{in} and the evidence vector u), and produces a new query-memory vector m_{out}. The new query memory is not necessarily identical to the original query memory, but is allowed

Model	EM	F1
Logistic Regression Baseline (Rajpurkar et al., 2016)	40.4	51.0
Dynamic Chunk Reader (Yu et al., 2016)	62.50	70.95
Fine-Grained Gating (Yang et al., 2017)	62.45	73.33
Match-LSTM with Ans-Ptr (Boundary)* (Wang and Jiang, 2017)	67.90	77.02
RaSoR (Lee et al., 2016)	69.64	77.69
FastQAExt (Weissenborn et al., 2017)	70.85	78.86
Mnemonic Reader (Ours)	**69.86**	**79.21**
Document Reader (Chen et al., 2017)	70.73	79.35
Ruminating Reader (Gong and Bowman, 2017)	70.64	79.45
jNet (Zhang et al., 2017)	70.61	79.82
Dynamic Coattention Networks* (Xiong et al., 2017)	71.63	80.38
Multi-Perspective Matching* (Wang et al., 2016)	73.77	81.26
BiDAF* (Seo et al., 2017)	73.74	81.53
SEDT+BiDAF* (Liu et al., 2017)	73.72	81.53
Mnemonic Reader* (Ours)	**73.67**	**81.69**
ReasoNet* (Shen et al., 2016)	75.03	82.55
r-net*†	76.92	84.01

Table 1: Performance comparison on the SQuAD test set as we submitted our model (April, 28, 2017). $*$ indicates ensemble models, and † indicates unpublished works on the leaderboard.

to retrieve correlative information from the evidence. In order to generate the output, the function G consists of two parts: *composition* and *gate*. The composition part produces a hidden state $z \in \mathbb{R}^{2h}$ which is a linear interpolation between the previous query memory and the evidence. The gate part generates an update gate $f \in \mathbb{R}^{2h}$ to control the composition degree to which the query memory is exposed. The new query-memory vector is calculated as follows:

$$\begin{aligned} z &= \tanh(W^z([m_{in}; u]) + b^z) \\ f &= \sigma(W^z([m_{in}; u]) + b^z) \\ m_{out} &= f \circ m_{in} + (1-f) \circ u \end{aligned} \tag{10}$$

Multi-hop extension. In complex query-context pairs, there may exist several candidate answer spans. The naive one-hop reasoning may fail to fully understand the query and predict the wrong answer. Therefore, the above one-hop answer pointing mechanism can by nature be extended to a multi-hop arhitecture, which allows the model to recover from the wrong answer span. In other word, multi-hop reasoning can help refining the predicted span to the correct position.

Let l denote the index of current hop and L denote the total number of hops. In the l-th hop, we obtain two estimated probability distributions: p_l^s for the start position of the answer and p_l^e for the end position. Next, we also utilize p_l^e as a first-order attention to aggregate information of candidate end positions, and generate an evidence vector $u_l^e \in \mathbb{R}^{2h}$ for the end position: $u_l^e = T \cdot p_l^e$. The query-memory vector for the start position in the next hop is then computed as: $m_{l+1}^s = \text{G}(m_l^e, u_l^e)$, and the following procedure for computing p_{l+1}^s and p_{l+1}^e is exactly the same as before. Such multi-hop extension allows the model to incorporate the information of both candidate start positions and end positions into the query memory, and therefore is helpful for rectifying the estimated probabilities. Finally the two probability distributions p_L^s and p_L^e are outputted for computing the answer span.

4 Evaluation

4.1 Experimental Configuration

We use the Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016) to evaluate our model. The SQuAD corpus totally contains more than 100k queries, which are manually annotated by crowdsourcing workers on 536 Wikipedia articles. The dataset includes 90k query-context tuples for training, 10k tuples for validation, and a large hidden test set. The answer to each query is a segment of text from the corresponding context. Two metrics, Exact Match (EM) and F1 score, are

Model	EM	F1
Mnemonic Reader*	72.7	81.2
Mnemonic Reader	69.7	79.1
- interactive alignment	64.1	73.7
- self alignment	67.4	77.0
- mnemonic answer pointing	68.2	77.5
- exact match feature	68.8	78.4
- character embedding	66.7	76.6

Table 2: Abalation results on the SQuAD development set.

used to evaluate models.

We use the Adam optimizer (Kingma and Ba, 2014) for training, with a minibatch size of 64 and an initial learning rate of 0.0006. We will half the learning rate when meet a bad epoch. A dropout rate (Srivastava et al., 2014) of 0.2 is used for word embeddings, all BiLSTM layers, and all linear transofrmations to prevent overfitting. Word embeddings are initialized with 100 dimensional Glove vectors (Pennington et al., 2014) and remain fixed during training, while the size of char embedding is set to be 10 and 100 1D filters are used for CNN. Out of vocabulary words are randomly sampled from Gaussian distributions. The dimension of hidden states h is 150 and the size of query-category embedding v is 100. The number of hops in mnemonic pointing layer is set to be 2. We set the max length of document to be 600 so as to efficiently train the model.

4.2 Overall Results

Table 1 shows the performance comparison of our models and other competing models on the official leaderboard of SQuAD. Our single Mnemonic Reader achieves an EM score of 69.86% and a F1 score of 79.21%. Since SQuAD is a very competitve machine comprehension benchmark, we also build an ensemble model which consists of 14 single models with the same architecture but initialized with different parameters. The answer span with the highest probability is choosen among all models for each query. Our ensemble model improves the EM score to 73.67%, and the F1 score to 81.69%, achieving competitive results on the official leaderboard.

4.3 Ablation Results

In order to evaluate the individual contribution of each model component, we run an ablation

Hops	EM	F1
0	68.2	77.5
1	69.5	79.0
2	**69.7**	**79.1**
3	69.3	78.7

Table 3: Performance comparison across different number of hops.

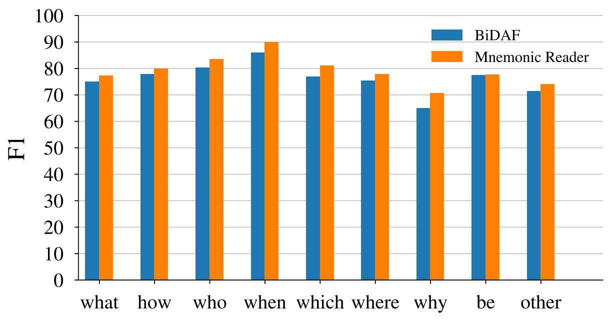


Figure 2: Comparison of F1 score divided by frequent query categories.

study on the the SQuAD development set, which is shown in Table 2. The interactive alignment between context and query is most critical to performance as ablating it results a drop of nearly 5.5% on both metrics. The reason may be that similar semantics between query and context act as strong evidences to the correct answer span. The self alignment accounts for about 2% of performance degradation, which clearly shows the effectiveness of aligning context words against themselves. We conjecture that self alignment helps information better flow through each word, alleviating long-term dependency problem in long sequence such as the context. To evaluate the performance of mnemonic pointing layer, we replace the multi-hop query-sensitive answer pointer with the standard pointer network (Vinyals et al., 2015), where the self-aware context representation T alongside the query summary $\tilde{q}$ are fed into a feedforward neural network, outputting unnormalized probability distributions followed by the softmax function. The result shows that mnemonic answer pointing outperforms the pointer network by nearly 1.5% on both metrics.

Further ablation study shows that the simple exact match feature has positive effect on the overall performace, which has been demonstrated by Weissenborn et al.(2017). Finally, character embeddings have a notable influence on the perfor-

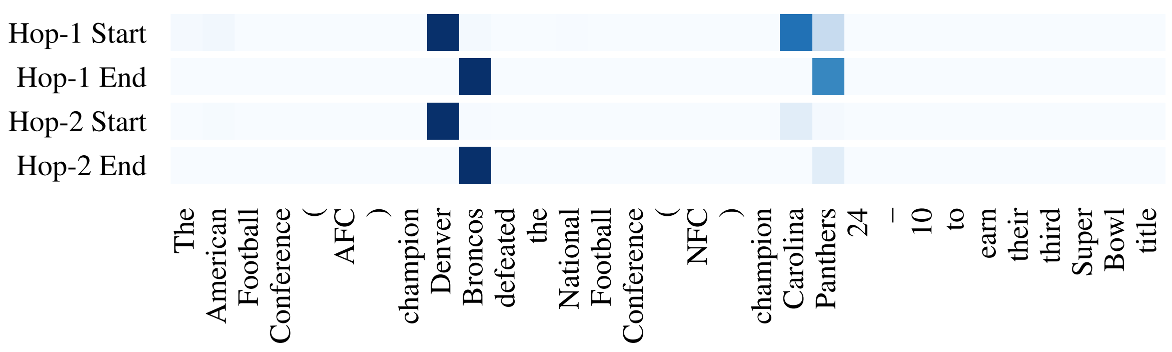


Figure 3: Examples of the start and end probability distributions estimated by the multi-hop mnemonic answer pointer. The query is ”Which NFL team represented the AFC at Super Bowl 50?”, while the correct answer is ”Denver Broncos”. The number of hops is set to be 2, and higher probability mass is indicated by darker regions.

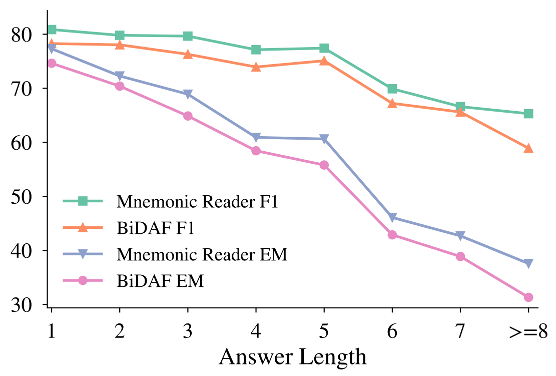


Figure 4: Performance comparison over different answer lengths.

mance which was already observed by Seo et al.(2017). We ran a simple statistics and found that 92.4% of answers in SQuAD are out-of-vocab (OOV) words. This phenomenon may explain the importance of character embeddings since character embeddings can better represent out-of-vocab (OOV) or rare words.

4.4 Result Analysis

To better understand the performance of Mnemonic Reader, we first conduct some quantitative analysis on the SQuAD development set. A natural point of interests is to analyze how much performance Mnemonic Reader improves with multi-hop reasoning, therefore we change the number of hops in mnemonic pointing layer and compare their different performances, which is shown in Table 3. As we can see, the performances on both metrics are increased gradually as the number of hops enlarges. The model with 2 hops achieves the best performance. The larger number of hops could potentially result in overfitting on the training set, therefore harming the performance.

Next we evaluate the performance of F1 score divided by frequent query types, which is shown in Figure 2. Here we use the state-of-the-art BiDAF single model as the baseline. Mnemonic Reader outperforms the baseline in every query category comfortably. The overall F1 score of our model is 79.1%, while the F1 of the reproduced BiDAF is 76.6%. Moreover, we can see that the F1 of ”when” and ”who” queries are highest and those of the ”why” query are the lowest. We conjecture that ”when” and ”who” queries only require the model to focus on specific information in the context (i.e., temporal information for ”when” and hominine information for ”who”), while the ”why” query requires the model be capable of high-level reasoning ability.

Figure 4 presents how the performance of two models degrades as the length of the answer increases. Although the performance of both models drops as the answer length increases, the performance of Mnemonic Reader is always better than BiDAF in both EM and F1 score.

Besides the quantitative analysis, we also provide a qualitative inspection of mnemonic pointing layer to show its ability of iteratively refining the answer span, as is shown in Figure 3. In the first hop the probabilities of candidate answer spans such as ”Denver Broncos” and ”Carolina Panthers” are both high. But in the subsequent hop, the model adjusts the answer span, lowering the probability of "Carolina" and "Panthers" as start position and end position respectively. This demonstrates that the model is capable of gradually locating the correct span by incorporating information of candidate answers into the query memory.

5 Conclusion

In this paper, we propose the Mnemonic Reader, an enhanced attention reader with mnemonic information such as self-aware representation and multi-hop query-sensitive pointer. Experiments on the large-scale SQuAD dataset showed that these mnemonic information leads to a significant performance improvement. On the test dataset, our single model obtains an F1 of 79.21% while our ensemble model achieves an F1 of 81.69%. For future work, we will introduce more useful mnemonic information to further augment the attention reader, such as common knowledge.

References

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

Danqi Chen, Jason Bolton, and Christopher D. Manning. 2016. A thorough examination of the cnn/daily mail reading comprehension task. In *Proceedings of ACL*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051* .

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *Proceedings of NIPS*.

Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. 2016. Attention-over-attention neural networks for reading comprehension. *arXiv preprint arXiv:1607.04423* .

Bhuwan Dhingra, Hanxiao Liu, William W. Cohen, and Ruslan Salakhutdinov. 2016. Gated-attention readers for text comprehension. *arXiv preprint arXiv:1606.01549* .

Yichen Gong and Samuel R. Bowman. 2017. Ruminating reader: Reasoning with gated multi-hop attention. *arXiv preprint arXiv:1704.07415* .

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT Press.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, , and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Proceedings of NIPS*.

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. 2016. The goldilocks principle: Reading childrens books with explicit memory representations. In *Proceedings of ICLR*.

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

Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. 2016. Text understanding with the attention sum reader network. In *Proceedings of ACL*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of EMNLP*.

Diederik P. Kingma and Lei Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *CoRR, abs/1412.6980*.

Ankit Kumar, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2015. Ask me anything: Dynamic memory networks for natural language processing ankit. In *CoRR, abs/1506.07285*.

Kenton Lee, Shimi Salant, Tom Kwiatkowski, Ankur Parikh, Dipanjan Das, and Jonathan Berant. 2016. Learning recurrent span representations for extractive question answering. *arXiv preprint arXiv:1611.01436* .

Rui Liu, Junjie Hu, Wei Wei, Zi Yang, and Eric Nyberg. 2017. Structural embedding of syntactic trees for machine comprehension. *arXiv preprint arXiv:1703.00572* .

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of EMNLP*.

Matthew Richardson, Christopher JC Burges, and Erin Renshaw. 2013. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of EMNLP*.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hananneh Hajishirzi. 2017. Bidirectional attention flow for machine comprehension. In *Proceedings of ICLR*.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2016. Reasonet: Learning to stop reading in machine comprehension. *arXiv preprint arXiv:1609.05284* .

Alessandro Sordonif, Phillip Bachmanf, and Yoshua Bengiog. 2016. Iterative alternating neural attention for machine reading. *arXiv preprint arXiv:1606.02245* .

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* pages 1929–1958.

RupeshKumar Srivastava, Klaus Greff, and Jurgen Schmidhuber. 2015. Highway networks. *arXiv preprint arXiv:1505.00387* .

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *Proceedings of NIPS*.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Proceedings of NIPS*.

Shuohang Wang and Jing Jiang. 2017. Machine comprehension using match-lstm and answer pointer. In *Proceedings of ICLR*.

Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. 2016. Multi-perspective context matching for machine comprehension. *arXiv preprint arXiv:1612.04211* .

Dirk Weissenborn, Georg Wiese, and Laura Seiffe. 2017. Fastqa: A simple and efficient neural architecture for question answering. *arXiv preprint arXiv:1703.04816* .

Caiming Xiong, Victor Zhong, and Richard Socher. 2017. Dynamic coattention networks for question answering. In *Proceedings of ICLR*.

Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W. Cohen, and Ruslan Salakhutdinov. 2017. Words or characters? fine-grained gating for reading comprehension. In *Proceedings of ICLR*.

Yang Yu, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. 2016. End-to-end answer chunk extraction and ranking for reading comprehension. *arXiv preprint arXiv:1610.09996* .

Junbei Zhang, Xiaodan Zhu, Qian Chen, Lirong Dai, Si Wei, and Hui Jiang. 2017. Exploring question understanding and adaptation in neural-network-based question answering. *arXiv preprint arXiv:1703.04617* .