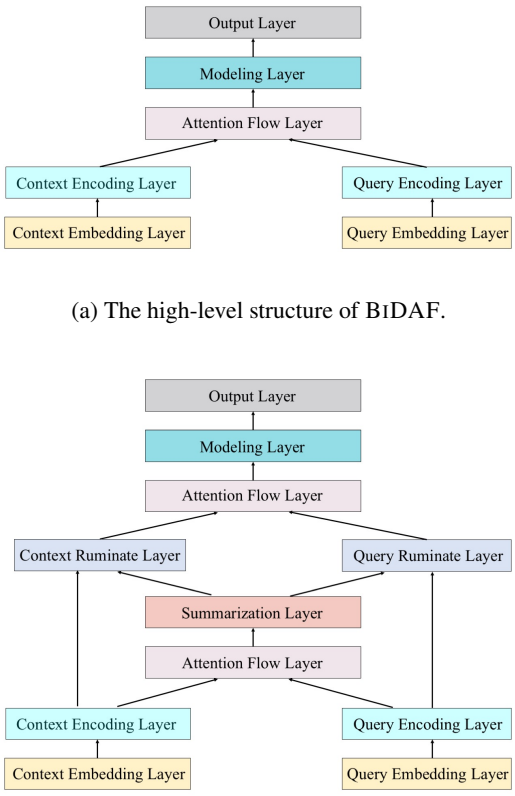
Ruminating Reader: Reasoning with Gated Multi-Hop Attention

Yichen Gong and **Samuel R. Bowman**
New York University
New York, NY
{yichen.gong,bowman}@nyu.edu

Abstract

To answer the question in machine comprehension (MC) task, the models need to establish the interaction between the question and the context. To tackle the problem that the single-pass model cannot reflect on and correct its answer, we present Ruminating Reader. Ruminating Reader adds a second pass of attention and a novel information fusion component to the Bi-Directional Attention Flow model (BIDAF). We propose novel layer structures that construct an query-aware context vector representation and fuse encoding representation with intermediate representation on top of BIDAF model. We show that a multi-hop attention mechanism can be applied to a bi-directional attention structure. In experiments on SQuAD, we find that the Reader outperforms the BIDAF baseline by a substantial margin, and matches or surpasses the performance of all other published systems.



(a) The high-level structure of BIDAF.

(b) The high-level structure of Ruminating Reader.

Figure 1: The high-level comparison between BIDAF and Ruminating Reader.

1 Introduction

The majority of recorded human knowledge is circulated in unstructured natural language. It is tremendously valuable to allow machines to read and comprehend the text knowledge. Machine comprehension (MC)—especially in the form of question answering (QA)—is therefore attracting a significant amount of attention from the machine learning community. Recently introduced large-scale datasets like CNN/Daily Mail (Hermann et al., 2015), the Stanford Question Answering Dataset (SQuAD; Rajpurkar et al., 2016) and the Microsoft MAchine Reading COmprehension Dataset (MS-MARCO; Nguyen et al., 2016) have allow data-driven methods, including deep learning, to become viable.

Recent approaches toward solving machine comprehension tasks using neural networks can be viewed as falling into two broad categories: single-pass reasoners and multiple-pass reasoners. Single-pass models read a question and a source text once and often adopt the differentiable attention mechanism that emphasizes important parts of the context related to the question.

BIDAF (Seo et al., 2017) represents one of

the state-of-the-art single-pass models in Machine Comprehension. BIDAF uses a bi-directional attention matrix which calculates the correlations between each word pair in context and query to build query-aware context representation. However, BIDAF and some similar models miss some questions because they don't have the capacity to reflect on problematic candidate answers and revise their decisions.

When humans are reading a text with the goal of answering a question, they tend to read it multiple times to get a better understanding of the context and question, and to give a better response.

With this intuition, recent multi-pass models revisit the question and the context passage (or *ruminate*) to infer the relations between the context, the question and the answer.

We propose an extension of BIDAF, called Ruminating Reader, which uses a second pass of reading and reasoning to allow it to learn to avoid mistakes and to ensure that it is able to effectively use the full context when selecting an answer. In addition to adding a second pass, we also introduce two novel layer types, the *ruminate layers*, which use gating mechanisms to fuse the obtained from the first and second passes. We observe a surprising phenomenon that when an LSTM layer in the context ruminate layer takes same input in each timestep, it can produce useful representation for the gates. In addition, we introduce an answer–question similarity loss to penalize overlap between question and predicted answer, a common feature in the errors of our base model. This allows us to achieve an F1 score of 79.5 and Exact Match (EM) score of 70.6 on hidden test set,[1] an improvement of 2.2 F1 score and 2.9 EM on BIDAF. Figure 1 shows a high-level comparison between BIDAF and Ruminating Reader.

This paper is organized as follows: In Section 2 we define the problem to be solved and introduce the SQuAD task. In Section 3 we introduce Ruminating Reader, focusing on the information-extracting and information-digesting components and how they integrate. Section 4 discusses related work. Section 5 presents the experimental setting, results and analysis. Section 6 concludes.

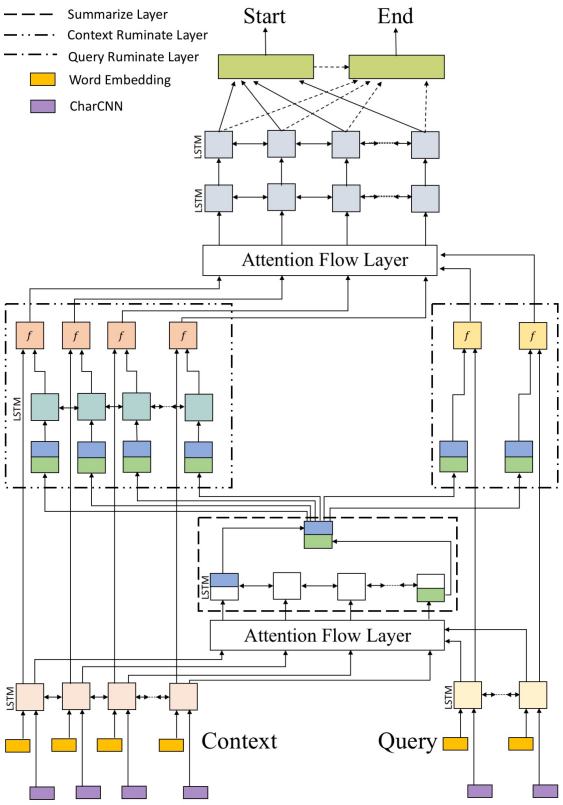


Figure 2: The model structure of our Ruminating Reader.

2 Question Answering

The task of the Ruminate Reader is to answer a question by reading and understanding a paragraph of text and selecting a span of words within the context. Formally, the Training and development data consist of tuples (Q, P, A), where Q = $(q_1, ..., q_i, ...q_{|Q|})$ is the question, a sequence of words with length $|Q|$, C = $(c_1, ...c_j, ..., c_{|C|})$ is the context, a sequence of words with length $|C|$, and A = (a_b, a_e) is the answer span marking the beginning and end indices of the the answer in the context ($1 <= a_b <= a_e <= |C|$).

SQuAD The SQuAD corpus is built using 536 articles randomly selected from English Wikipedia. Images, figures, tables are stripped and any paragraphs shorter than 500 characters are discarded. Unlike other datasets that such as CNN/Daily Mail whose questions are synthesized, Rajpurkar et al. (2016) uses a crowdsourcing platform to generate realistic question and answer pairs. SQuAD contains 107,785 question-

[1]The latest results are listed at https://rajpurkar.github.io/SQuAD-explorer/

answer pairs. The typical context length spans from 50 tokens to 250 tokens. The typical length of a question is around 10 tokens. The answer be any span of words from the context, resulting in $O(|C|^2)$ possible outputs.

3 Our Model

3.1 Ruminating Reader

In this section, we review the BIDAF model (Seo et al., 2017) and introduce our extension, the Ruminating Reader.

Our additions to the base model are motivated by the intuition that adding an additional pass of reading will allow the model to better integrate information from the question and answer and to better weigh possible answers, and that by interpolating the results of the second pass with those of the first pass through gating, we can prevent the additional complexity that we add to the model from substantially increasing the difficulty of training. The structure of our model is shown in Figure 2 and explained in the following sections.

Character Embedding Layer Just as in the base BIDAF model, the character embedding layer maps each word to a high dimensional vector using character features. It does so using a convolutional neural network with max pooling over learned character vectors (Lee et al., 2017; Kim et al., 2016). Thus we have a context character representation $\boldsymbol{M} \in \mathbb{R}^{f \times C}$ and a query representation $\boldsymbol{N} \in \mathbb{R}^{f \times Q}$, where C is the sequence length of the context, Q is the sequence length of the query and f is the number of 1D convolutional neural network filters.

Word Embedding Layer Again as in the base model, the word embedding layer uses pretrained word vectors (the 6B GloVe vectors of Pennington et al., 2014) to map the word into a high dimensional vector space. We do not update the word embeddings during training. The character embedding and the word embedding are concatenated and passed into a two-layer highway network (Srivastava et al., 2015) to obtain a d dimensional vector representation of each single word. Hence, we have a context representation $\boldsymbol{H} \in \mathbb{R}^{d \times C}$ and a query representation $\boldsymbol{U} \in \mathbb{R}^{d \times Q}$.

Sequence Encoding Layers As in BIDAF, we use two LSTM RNNs (Hochreiter and Schmidhuber, 1997) with d-dimensional outputs to encode the context and query representations in both directions. Therefore, we obtain a context encoding matrix $\boldsymbol{C} \in \mathbb{R}^{2d \times C}$, and a query encoding matrix $\boldsymbol{Q} \in \mathbb{R}^{2d \times Q}$.

Attention Flow Layer As in BIDAF, the attention flow layer constructs a query-aware context representation $\boldsymbol{G}$ from inputs $\boldsymbol{C}$ and $\boldsymbol{Q}$. This layer takes two steps. In the first step, an interaction matrix $\boldsymbol{I} \in \mathbb{R}^{C \times Q}$ is computed, which indicates the affinities between each context word encoding and each query word encoding. $\boldsymbol{I}_{cq}$ indicates the correlation between the c-th word in context and q-th word in query. The interaction matrix is computed by

$$\boldsymbol{I}_{cq} = \boldsymbol{w}_{(\boldsymbol{I})}^{\top}[\boldsymbol{C}_c; \boldsymbol{Q}_q; \boldsymbol{C}_c \circ \boldsymbol{Q}_q] \quad (1)$$

where $\boldsymbol{w}_I \in \mathbb{R}^{6d}$ is a trainable parameter, $\boldsymbol{C}_c$ is c-th column of context encoding and $\boldsymbol{Q}_q$ is q-th column of query encoding, $\circ$ is elementwise multiplication, and $[;]$ is vector concatenation.

Context-to-query Attention As in BIDAF, the context-to-query attention component generates, for each context word, an attention-weighted sum of query word encodings. Let $\tilde{\boldsymbol{Q}} \in \mathbb{R}^{2d \times C}$ represent the context-to-query attention matrix. For column c in $\tilde{\boldsymbol{Q}}$ is defined by $\tilde{\boldsymbol{Q}}_c = \sum(\boldsymbol{a}_{cq}\boldsymbol{Q}_q)$, where $\boldsymbol{a}$ is the attention weight. $\boldsymbol{a}$ is computed by $\boldsymbol{a}_c = softmax(\boldsymbol{I}_c) \in \mathbb{R}^Q$.

Query-to-context Attention Query-to-context attention indicates the most relevant context words to query. The most relevant word vector representation is an attention-weighted sum defined by $\tilde{\boldsymbol{c}} = \sum \boldsymbol{b}_c \boldsymbol{C}_c$ where $\boldsymbol{b}$, is an attention weight which is calculated by $\boldsymbol{b} = softmax(max_{col}(\boldsymbol{I})) \in \mathbb{R}^C$. $\tilde{\boldsymbol{c}}$ is replicated C times across the column, therefore giving $\tilde{\boldsymbol{C}} \in \mathbb{R}^{2d \times C}$.

We then obtain the final query-aware context representation by

$$\boldsymbol{G}_c = [\boldsymbol{C}_c; \tilde{\boldsymbol{Q}}_c; \boldsymbol{C}_c \circ \tilde{\boldsymbol{Q}}_c; \boldsymbol{C}_c \circ \tilde{\boldsymbol{C}}_c] \quad (2)$$

where $\boldsymbol{G}_c \in \mathbb{R}^{8d \times C}$.

Summarization Layer We propose summarization layer which produces a vector representation that summarizes the information in the query-aware context representation. The input to summarization layer is $\boldsymbol{G}$. We use one bi-directional LSTM network to model the learned information. We select the final states from both directions and concatenate them together as $\boldsymbol{s} = [\boldsymbol{s}_f; \boldsymbol{s}_b]$. where

$\boldsymbol{s} \in \mathbb{R}^{2d}$ represents the representation summarized from the reading of context and query, $\boldsymbol{s}_f$ is the final state of LSTM in forward direction, and $\boldsymbol{s}_b$ is the final state of LSTM in backward direction.

Query Ruminate Layer The query ruminate layer fuses the summarization vector representation with the query encoding $\boldsymbol{Q}$, helping reformulate the query representation in order to maximize the chance of retrieving the correct answer. The input to this layer is $\boldsymbol{s}$ tiled Q times ($\boldsymbol{S}_Q \in \mathbb{R}^{2d \times Q}$). A gating function then fuses this with the existing query encoding:

$$\boldsymbol{z}_i = tanh(\boldsymbol{W}_{Qz}^{1\top} \boldsymbol{S}_{Qi} + \boldsymbol{W}_{Qz}^{2\top} \boldsymbol{Q}_i + \boldsymbol{b}_{Qz}) \quad (3)$$

$$\boldsymbol{f}_i = \sigma(\boldsymbol{W}_{Qf}^{1\top} \boldsymbol{S}_{Qi} + \boldsymbol{W}_{Qf}^{2\top} \boldsymbol{Q}_i + \boldsymbol{b}_{Qf}) \quad (4)$$

$$\tilde{\boldsymbol{Q}}_i = \boldsymbol{f}_i \circ \boldsymbol{Q}_i + (1 - \boldsymbol{f}_i) \circ \boldsymbol{z}_i \quad (5)$$

where $\boldsymbol{W}_{Qz}^1, \boldsymbol{W}_{Qz}^2, \boldsymbol{W}_{Qf}^1, \boldsymbol{W}_{Qf}^2 \in \mathbb{R}^{2d \times 2d}$ and $\boldsymbol{b}_{Qz}, \boldsymbol{b}_{Qf} \in \mathbb{R}^{2d}$ are trainable parameters, $\boldsymbol{S}_{Qi}$is the i-th column of the $\boldsymbol{S}_Q$, $\boldsymbol{Q}_i$ is the i-th column of $\boldsymbol{Q}$.

Context Ruminate Layer Context ruminate layer digests the summarization and integrates it with the context encoding $\boldsymbol{C}$ to facilitate answer extraction. In this layer, we tile $\boldsymbol{s}$ C times and we have $\boldsymbol{S}_C \in \mathbb{R}^{2d \times C}$. To incorporate the positional information into this relatively long tiled sequence, we feed it into an additional bidirectional LSTM with output size d in each direction. This approach, while somewhat inefficient, proves to be an valuable addition to the model and allows it to better track position information, loosely following the positional encoding strategy of Sukhbaatar et al. (2015). Hence we obtain $\tilde{\boldsymbol{S}}_C \in \mathbb{R}^{2d \times C}$, which is fused with context encoding $\boldsymbol{C}$ via a gate:

$$\boldsymbol{z}_i = tanh(\boldsymbol{W}_{Cz}^{1\top} \tilde{\boldsymbol{S}}_{Ci} + \boldsymbol{W}_{Cz}^{2\top} \boldsymbol{C}_i + \boldsymbol{b}_{Cz}) \quad (6)$$

$$\boldsymbol{f}_i = \sigma(\boldsymbol{W}_{Cf}^{1\top} \tilde{\boldsymbol{S}}_{Ci} + \boldsymbol{W}_{Cf}^{2\top} \boldsymbol{C}_i + \boldsymbol{b}_{Cf}) \quad (7)$$

$$\tilde{\boldsymbol{C}}_i = \boldsymbol{f}_i \circ \boldsymbol{C}_i + (1 - \boldsymbol{f}_i) \circ \boldsymbol{z}_i \quad (8)$$

where $\boldsymbol{W}_{Cz}^1, \boldsymbol{W}_{Cz}^2, \boldsymbol{W}_{Cf}^1, \boldsymbol{W}_{Cf}^2 \in \mathbb{R}^{2d \times 2d}$ and $\boldsymbol{b}_{Cz}, \boldsymbol{b}_{Cf} \in \mathbb{R}^{2d}$ are trainable parameters, $\tilde{\boldsymbol{S}}_{Ci}$is the i-th column of the $\tilde{\boldsymbol{S}}_C$, $\boldsymbol{C}_i$ is the i-th column of $\boldsymbol{C}$.

Context: The Broncos took an early lead in Super Bowl 50 and never trailed. Newton was limited by Denver's defense, which sacked him seven times and forced him into three turnovers, including a fumble which they recovered for a touchdown. Denver linebacker Von Miller was named Super Bowl MVP, recording five solo tackles, 2 sacks, and two forced fumbles.

Question: Which Newton turnover resulted in seven points for Denver?

Ground Truth: {a fumble, a fumble, Fumble}

Prediction: three turnovers

Table 1: An error of the type that motivated the answer–question similarity loss.

Second Hop Attention Flow Layer We take $\tilde{\boldsymbol{Q}} \in \mathbb{R}^{2d \times Q}$ and $\tilde{\boldsymbol{C}} \in \mathbb{R}^{2d \times C}$ as the input to another attention flow layer with the same structure as described above, yielding $\boldsymbol{G}^{(2)} \in \mathbb{R}^{8d \times C}$.

Modeling Layer We use two layers of bi-directional LSTM with output size d in each direction to aggregate the information in $\boldsymbol{G}^{(2)}$, yielding a pre-output matrix $\boldsymbol{M}^s \in \mathbb{R}^{2d \times C}$.

Output Layer As in BIDAF, our output layer independently models the probability of each word being selected as the start or end of an answer span. We calculate the probability distribution of the start index of the answer span by

$$\boldsymbol{p}^s = softmax(\boldsymbol{w}_{\boldsymbol{p}^1}^\top [\boldsymbol{G}; \boldsymbol{M}^s]) \quad (9)$$

where $\boldsymbol{w}_{(\boldsymbol{p}^1)} \in \mathbb{R}^{10d}$ is a trainable parameter. We pass the matrix $\boldsymbol{M}^s$ to another bi-directional LSTM with output size d in single direction yielding $\boldsymbol{M}^e$. We obtain the probability distribution of the end index of the answer span by

$$\boldsymbol{p}^e = softmax(\boldsymbol{w}_{(\boldsymbol{p}^2)}^\top [\boldsymbol{G}; \boldsymbol{M}^e]) \quad (10)$$

Training Loss We define the training loss as the sum of three components: negative log likelihood loss, L2 regularization loss, and a novel answer–question similarity loss.

Answer–Question Similarity Loss We observe that a version of our model trained only on the two standard loss terms often selects answers that overlap substantially in content with their corresponding questions, and that this nearly always results

in an error. A sample error of this kind is shown in Table 1. This motivates an additional loss term at training time: We penalize the similarity between the question and the selected answer. Formally, the answer question similarity loss is defined as

$$s = Argmax(\boldsymbol{p}^1) \tag{11}$$

$$e = Argmax(\boldsymbol{p}^2) \tag{12}$$

$$\vec{\boldsymbol{q}}_{BoW} = \frac{Sum_{row}(\boldsymbol{Q})}{Q} \tag{13}$$

$$AQSL(\theta) = cos(\boldsymbol{C}_s, \vec{\boldsymbol{q}}_{BoW}) + cos(\boldsymbol{C}_e, \vec{\boldsymbol{q}}_{BoW}) \tag{14}$$

where s refers to the start index of answer span, e refers to the end index of the answer span, $\vec{\boldsymbol{q}}_{BoW}$ is the bag of words representation of query encoding, $cos(a, b)$ is the cosine similarity between a and b, $\boldsymbol{C}_s$ and $\boldsymbol{C}_e$ are the s-th and e-th vector representation of context encoding.

Prediction During prediction, we use a local search strategy that for token indices a and a', we maximize $\boldsymbol{p}^s_a \times \boldsymbol{p}^e_{a'}$, where $0 \leq a' - a \leq 15$. Dynamic programming is applied during search, resulting in $O(C)$ time complexity.

4 Related Work

Recently, both QA and Cloze-style machine comprehension tasks like CNN/Daily Mail have seen fast progress. Much of this recent work has been based on end-to-end trained neural network models, and within that, most have used recurrent neural networks with soft attention (Bahdanau et al., 2015), which emphasizes one part of the data over the others. These models can be coarsely divided into two categories: single-pass and multi-pass reasoners.

Most papers on single-pass reasoning systems propose novel ways to use the attention mechanism: Wang and Jiang (2016) propose match-LSTM to model the interaction between context and query, as well as introducing the use of a pointer network (Vinyals et al., 2015) to extract the answer span from the context. Xiong et al. (2017) propose the Dynamic Coattention Network, which uses co-dependent representations of the question and the context, and iteratively updates the start and end indices to recover from local maxima and to find the optimal answer span. Wang et al. (2016) propose the Multi-Perspective Context Matching model that matches the encoded context with query by combining various matching strategies, aggregates matching vector with bi-directional LSTM, and predict start and end positions. In order to merge the entity score during its multiple appearence, Kadlec et al. (2016) propose attention-sum reader who computes dot product between context and query encoding, does a softmax operation over context and sums the probability over the same entity to favor the frequent entities over rare ones. Chen et al. (2016) propose to use a bilinear term to calculate the attentional alignment between context and query.

Among multi-hop reasoning systems: Hill et al. (2015) apply attention on window-based memory, by extending multi-hop end-to-end memory network (Sukhbaatar et al., 2015). Dhingra et al. (2016) extend attention-sum reader to multi-turn reasoning with an added gating mechanism. The Iterative Alternative (IA) reader (Sordoni et al., 2016) produces query glimpse and document glimpse in each iterations and uses both glimpses to update recurrent state in each iteration. Shen et al. (2017) propose a multi-hop attention model that used reinforcement learning to dynamically determine when to stop digesting intermediate information and produce an answer.

5 Evaluation

5.1 Implementation details

Our model configuration closely follows that of Seo et al. (2017) did: In the character encoding layer, we use 100 filters of width 5. In the remainder of the model, we set the hidden layer dimension (d) to 100. We use pretrained 100D GloVe vectors (6B-token version) as word embeddings. Out-of-vocobulary tokens are represented by an UNK symbol in the word embedding layer, but treated normally by the character embedding layer. The BiLSTMs in context and query encoding layers share same weights. We use the AdaDelta optimizer (Zeiler, 2012) for optimization.

We selected hyperparameter values through random search (Bergstra and Bengio, 2012). Batch size is 30. Learning rate starts at 0.5, and decreases to 0.2 once the model stops improving. The L2-regularization weight is 1e-4, AQSL weight is 1 and dropout with a drop rate of 0.2 is

Model	Test	
	F1	EM
Logistic Regression[a]	51.0	40.4
Dynamic Chunk Reader[b]	70.956	62.499
Fine-grained Gating[c]	73.327	62.446
Match-LSTM[d]	73.743	64.744
Dynamic Coattention Network[e]	75.896	66.233
Bidirectional Attention Flow[f]	77.323	67.974
RaSoR[g]	77.696	69.642
Multi-perspective Matching[h]	77.771	68.877
FastQAExt[i]	78.857	70.849
Document Reader[j]	79.353	70.733
ReasoNet[k]	79.364	70.555
jNet[l]	79.821	70.607
Interactive AoA Reader[‡]	79.937	71.153
QFASE[‡]	79.989	71.898
r-net[‡]	**80.717**	**72.338**
Ruminating Reader	79.456	70.639

Table 2: The Official SQuAD leaderboard performance on test set for single model section from April 23, 2017, the time of submission. There are other unpublished systems shown on leaderboard, including Document Reader and r-net.

[a] Rajpurkar et al. (2016); [b] Yu et al. (2016);
[c] Yang et al. (2017); [d] Wang and Jiang (2016);
[e] Xiong et al. (2017); [f] Seo et al. (2017);
[g] Lee et al. (2016); [h] Wang et al. (2016);
[i] Weissenborn et al. (2017); [j] Chen et al. (2017);
[k] Shen et al. (2017); [l] Zhang et al. (2017);
[‡] Unpublished

applied to all forward connections in the CNN, the LSTMs, and all feedforward layers.

A typical model run converges in about 40k steps. This takes two days using Tensorflow (Abadi et al., 2015) and a single NVIDIA K80 GPU .

5.2 Evaluation Method

Rajpurkar et al. (2016) provide an official evaluation script that allows us to measure F1 score and EM score by comparing the prediction and ground truth answers. Three answers are provided for each question. The prediction is compared to each of the answer and best score is selected. F1 score is defined by recall and precision of words and EM score, as Exact Match score, is defined as the score of 100% accuracy in prediction. We do not use any kind of ensembling, and compare our results primarily with other single-model (non-ensemble) results. The test set performance is evaluated at CodaLab by administrator.

Ablation Variant	Dev	
	F1	EM
1. BIDAF	77.3	67.7
2. BIDAF w/ L2 Reg., AQSL, LS	77.7	68.6
3. RR w/o query ruminate layer	78.7	69.6
4. RR w/o context ruminate layer	78.9	70.0
5. RR w/ BiLSTM in QRL	**79.4**	70.4
6. RR w/o BiLSTM in CRL	74.0	64.2
7. RR w/o query input at $\boldsymbol{s}$,$\boldsymbol{f}$ in QRL	78.8	70.1
8. RR w/o context input at $\boldsymbol{s}$,$\boldsymbol{f}$ in CRL	78.9	70.3
9. RR w/o query input in QRL	63.3	54.1
10. RR w/o context input in CRL	27.0	9.4
11. RR w/o summ. input in QRL	79.2	70.1
12. RR w/o summ. input in CRL	79.2	70.3
Ruminating Reader	**79.5**	**70.6**

Table 3: Layer ablation results. The order of the listing corresponds to the description in Appendix A.1. CRL refers to context ruminate layer and QRL refers to query ruminate layer.

5.3 Results

At the time of submission, our model is tied in accuracy on the hidden test set with the best-performing published single model (Zhang et al., 2017). We achieve an F1 score of 79.5 and EM score of 70.6. The current leaderboard is displayed in Table 2. The leaderboard is listed in descending order of F1 score, but if an entry's F1 score is better than the adjacent entry's, while its EM score is worse, then these two entries are considered tied.

5.4 Analysis

Layer Ablation Analysis To analyze how each component contribute to the model, we run a layer ablation experiment. We present results for twelve versions of the model on the development set, each missing some or all of the major components of the full Ruminating Reader. The precise definition of each of the twelve ablated models can be found in Appendix A.1.

The results of the ablation experiment are shown in Table 3. The ablation experiments show how each component contribute to the model. Experiments 3 and 4 show that the two ruminate layers are both important and helpful in contributing performance. It is worth noting that the BiLSTM in the context ruminate layer contributes substantially to model performance. We find this somewhat surprising, since it takes the same input in each timestep, but it nonetheless successfully digests the summarization information representation and produces a useful input for the gating component. Experiments 7 and 8 show that the

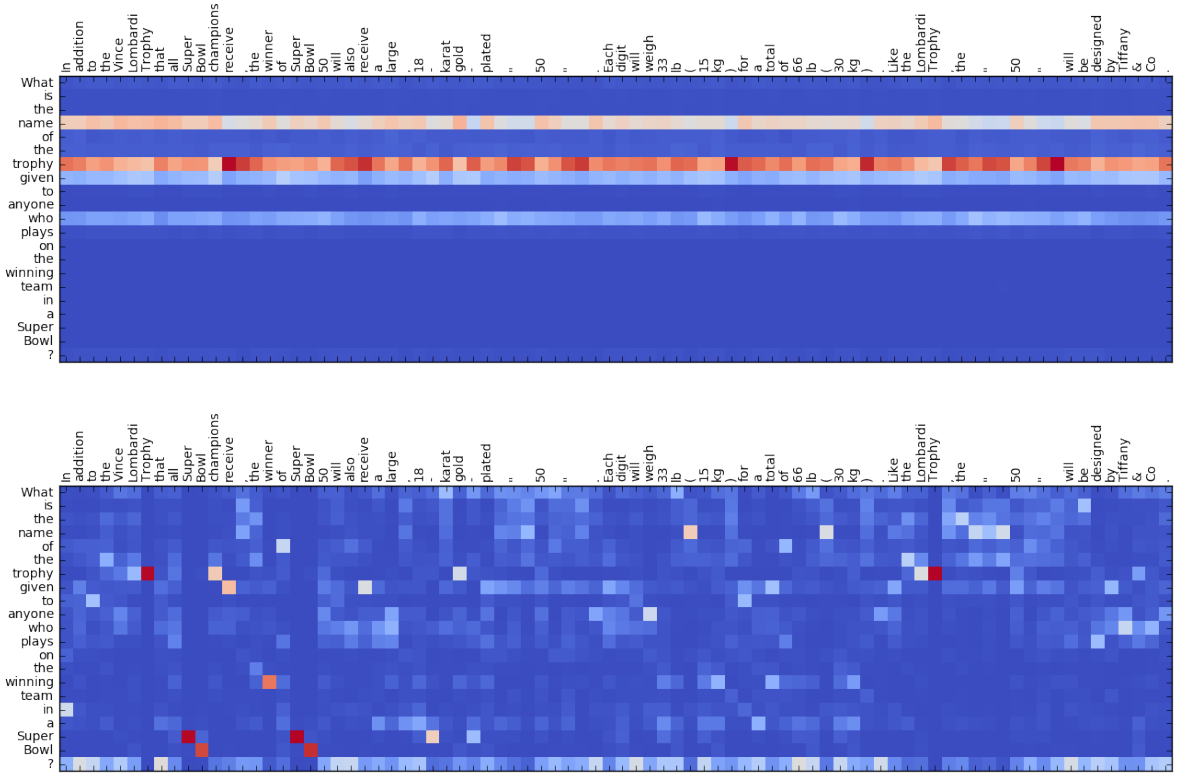


Figure 3: The visualization of first hop (top) and second hop (bottom) attention interaction matrix. We use coolwarm colormap, where red is close to 1 and blue is close to 0. In the question "What is the name of the trophy given to anyone who plays on the winning team in a super Bowl?", the key words *name*, *trophy*, *given*, *who* are strongly attended to in the first hop.

modeled summarization vector representation can provide information to gates reasonably well. The drop in performance in both experiments 9 and 10 shows that the key information for new query and context representation are the are first stage query and context encodings. Experiments 11 and 12 shows that the summarization vector representation does help the later stage of reasoning.

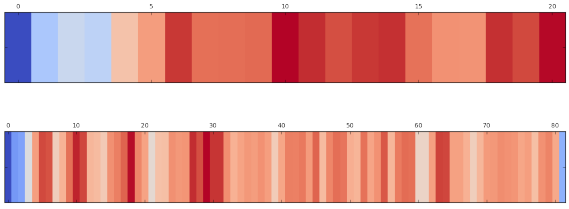


Figure 4: Visualizations of gate values for the query (top) and context (bottom) ruminate layers. The order of words is the same as in Figure 3. We use coolwarm colormap, where red means the gate uses more information from intermediate representation, and blue from encoding representation.

Visualization Figure 3 provides a visualization of the first hop and second hop attention interaction matrix $\boldsymbol{I}$. We also provide a sample of visualization for the L2 sum of gate value in context and query ruminate layers in Figure 4.

From Figure 3 we see that though the structures of two hops of attention flow layer are the same, they function quite differently in typical cases. The first hop attention appears to be primarily concerned with identifying the key informative word (or words, as here) in the query. Though in Figure 3 four key words are signified, one or two words are attended to in the first hop in the common case. The second hop is then responsible for finding candidate answers that are relevant to those key words and generating a query-aware context representation. We observe the first hop attention shows a consistent attention pattern across context words, suggesting that there may be room to make the first hop component more efficient in future work.

From Figure 4, we see the gate value on both query ruminate layer and context ruminate layer

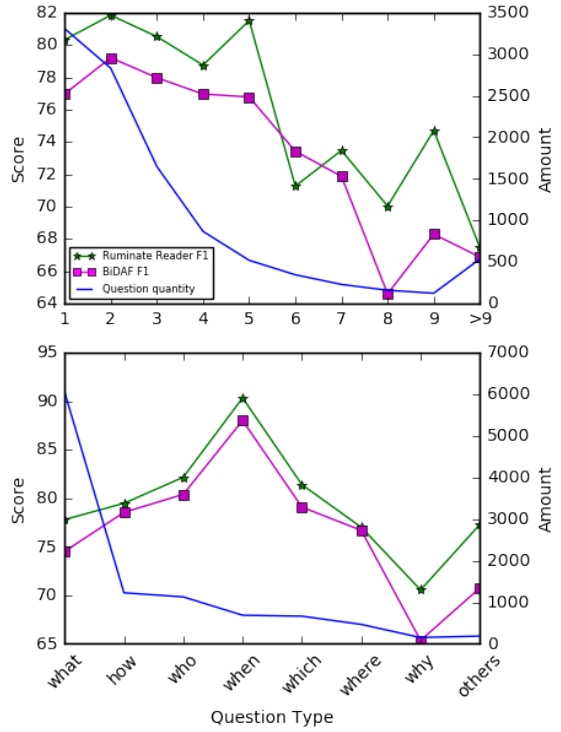


Figure 5: An analysis according to answer length of questions and question types. The top graph shows the comparison of F1 score between Ruminating Reader and the BIDAF model on the development set by answer length. The blue line shows the distribution of questions by answer length. The bottom graph shows a corresponding comparison BIDAF by question type (*wh*-word).

shows that the gates are working to fuse information to original query encoding and context encoding. We observe that in most of the case the gates in ruminate layers uses more information from encoding than from summarization representation. The observation matches our expectation that the gates modify and improve on the encoding representation.

We also provide a comparison of F1 score between BIDAF and Ruminating Reader on question with different ground truth answer length and different types of questions in Figure 5. Exact match score is highly correlated with F1 score so we omit it for clarity. We observe that the Ruminating Reader outperforms BIDAF on most of the questions with respect of different answer length. On the question with long answer length, of 5, 8 and 9, Ruminating Reader outperforms BIDAF by a great margin. Questions with longer reference answers appear to be more difficult to answer. In addition, the Ruminating Reader does better on each type of question. Both models work best for *when* questions—these question are answerable by temporal expressions, which are relatively easy to recognize. The *Why* questions are hardest to answer—they tend to have long answers with no purely lexical cues marking their beginnings or ends. Ruminating Reader outperforms BIDAF model on *why* questions by a substantial margin.

Performance Breakdown Following Zhang et al. (2017), we break down Ruminating Reader's 79.5% F1 score on the development set into three sub-scores, representing failures, partial successes, and successes. On 13.5% of development set examples, Ruminate Reader fails, yielding 0% F1. On 70.6% of examples, Ruminate Reader achieves a perfect F1 score. On the remaining 15.9%, Ruminate Reader got only partial matches (i.e., answers that partially overlapped with reference answers), with an average F1 score of 56.0%. Comparing to the jNet (Zhang et al., 2017) whose success answers occupy 69.1% of all answers, failure score answers 14.9% and partial success 16.01% with an average F1 score of 58.0%, our model works better on increasing successes and reducing failures.

6 Conclusion

We propose the Ruminating Reader, an extension to the BIDAF model with two-hop attention. The model surpasses the original BIDAF model's performance on Stanford Question Answering Dataset (SQuAD) by a large margin, and ties with the best published system. These results and our qualitative analysis both suggest that the model successfully fuses the information from two passes of reading using gating and uses the result to identify appropriate answers to Wikipedia questions. An ablation experiment shows that each of components of this complex model contribute substantially. In future work, we aim to find ways to simplify this model without impacting performance, to explore the possibility of yet deeper models, and to expand our study to machine comprehension tasks more broadly.

Acknowledgments

We thank Pranav Rajpurkar for testing Ruminate Reader on SQuAD hidden test set.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proc. ICLR*.

James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. In *Proc. JMLR*.

Danqi Chen, Jason Bolton, and Christopher D Manning. 2016. A Thorough Examination of the CNN/Daily Mail Reading Comprehension Task. In *Proc. ACL*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to Answer Open-Domain Questions. In *Proc. ACL*.

Bhuwan Dhingra, Hanxiao Liu, William W Cohen, and Ruslan Salakhutdinov. 2016. Gated-Attention Readers for Text Comprehension. In *CoRR*.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching Machines to Read and Comprehend. In *Proc. NIPS*.

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. 2015. The Goldilocks Principle: Reading Children's Books with Explicit Memory Representations. In *Proc. ICLR*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. volume 9, pages 1735–1780.

Rudolf Kadlec, Martin Schmid, Ondrej Bajgar, and Jan Kleindienst. 2016. Text Understanding with the Attention Sum Reader Network. In *Proc. ACL*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-Aware Neural Language Models. In *Proc. AAAI*.

Jason Lee, Kyunghyun Cho, and Thomas Hofmann. 2017. Fully Character-Level Neural Machine Translation without Explicit Segmentation. In *Proc. ACL*.

Kenton Lee, Shimi Salant, Tom Kwiatkowski, Ankur Parikh, Dipanjan Das, and Jonathan Berant. 2016. Learning Recurrent Span Representations for Extractive Question Answering. ArXiv:1611.01436.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. In *Proc. CoCo@NIPS*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. In *Proc. EMNLP*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD - 100, 000+ Questions for Machine Comprehension of Text. In *Proc. EMNLP*.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2017. Bidirectional Attention Flow for Machine Comprehension. In *Proc. ICLR*.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2017. ReasoNet: Learning to Stop Reading in Machine Comprehension. In *Proc. NIPS*.

Alessandro Sordoni, Phillip Bachman, and Yoshua Bengio. 2016. Iterative Alternating Neural Attention for Machine Reading. In *CoRR*.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway Networks. In *Proc. ICML*.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-To-End Memory Networks. In *Proc. NIPS*.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer Networks. In *Proc. NIPS*.

Shuohang Wang and Jing Jiang. 2016. Machine Comprehension Using Match-LSTM and Answer Pointer. ArXiv:1608.07905.

Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. 2016. Multi-Perspective Context Matching for Machine Comprehension. ArXiv:1612.04211.

Dirk Weissenborn, Georg Wiese, and Laura Seiffe. 2017. Making Neural QA as Simple as Possible but not Simpler. ArXiv:1703.04816.

Caiming Xiong, Victor Zhong, and Richard Socher. 2017. Dynamic Coattention Networks For Question Answering. In *Proc. 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 *Proc. 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:1610.09996.

Matthew D Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. ArXiv:1212.5701.

Junbei Zhang, Xiaodan Zhu, Qian Chen, Lirong Dai, and Hui Jiang. 2017. Exploring Question Understanding and Adaptation in Neural-Network-Based Question Answering. ArXiv:1703.04617.

A Appendix

A.1 Layer Ablation Experiments setup

In this section we show the setup of layer ablation experiment details in Table 3.

1. Vanilla BIDAF

2. Ruminating Reader without context and query ruminate layer. Therefore, the model is equivalent to original BIDAF model with L2-regurization, answer-question similarity penalization and local search prediction feature.

3. Ruminating Reader without query ruminate layer. The query encoding $\boldsymbol{Q}$ is directly fed into the second hop attention flow layer.

4. Ruminating Reader without context ruminate layer. The context encoding $\boldsymbol{C}$ is directly connected to the second hop attention flow layer without digesting newly acquired information.

5. Ruminating Reader with BiLSTM modeling in query ruminate layer. Formally, we have $\tilde{\boldsymbol{S}}_Q = BiLSTM(\boldsymbol{S}_Q)$ in query ruminate layer. Therefore, the query ruminate layer is defined by

$$\boldsymbol{z}_i = tanh(\boldsymbol{W}_{Qz}^{1\top}\tilde{\boldsymbol{S}}_{Qi} + \boldsymbol{W}_{Qz}^{2\top}\boldsymbol{Q}_i + \boldsymbol{b}_{Qz}) \tag{15}$$

$$\boldsymbol{f}_i = \sigma(\boldsymbol{W}_{Qf}^{1\top}\tilde{\boldsymbol{S}}_{Qi} + \boldsymbol{W}_{Qf}^{2\top}\boldsymbol{Q}_i + \boldsymbol{b}_{Qf}) \tag{16}$$

$$\tilde{\boldsymbol{Q}}_i = \boldsymbol{f}_i \circ \boldsymbol{Q}_i + (1 - \boldsymbol{f}_i) \circ \boldsymbol{z}_i \tag{17}$$

6. Ruminating Reader without BiLSTM modeling in context ruminate layer. Formally, we have $\tilde{\boldsymbol{S}}_C = \boldsymbol{S}_C$ in query ruminate layer and all other components remains the same.

7. Ruminating Reader without query input at $\boldsymbol{z}$, $\boldsymbol{f}$ in query ruminate layer. While all other components remain the same as in Ruminating Reader, the gate in query ruminate layer is defined by

$$\boldsymbol{z}_i = tanh(\boldsymbol{W}_{Qz}^{1\top}\boldsymbol{S}_{Qi} + \boldsymbol{b}_{Qz}) \tag{18}$$

$$\boldsymbol{f}_i = \sigma(\boldsymbol{W}_{Qf}^{1\top}\boldsymbol{S}_{Qi} + \boldsymbol{b}_{Qf}) \tag{19}$$

$$\tilde{\boldsymbol{Q}}_i = \boldsymbol{f}_i \circ \boldsymbol{Q}_i + (1 - \boldsymbol{f}_i) \circ \boldsymbol{z}_i \tag{20}$$

8. Ruminating Reader without context input at $\boldsymbol{z}$, $\boldsymbol{f}$ in context ruminate layer. While all other components remain the same as in Ruminating Reader, the gate in context ruminate layer is defined by

$$\boldsymbol{z}_i = tanh(\boldsymbol{W}_{Cz}^{1\top}\tilde{\boldsymbol{S}}_{Ci} + \boldsymbol{b}_{Cz}) \tag{21}$$

$$\boldsymbol{f}_i = \sigma(\boldsymbol{W}_{f}^{1\top}\tilde{\boldsymbol{S}}_{Ci} + \boldsymbol{b}_{Cf}) \tag{22}$$

$$\tilde{\boldsymbol{C}}_i = \boldsymbol{f}_i \circ \boldsymbol{C}_i + (1 - \boldsymbol{f}_i) \circ \boldsymbol{z}_i \tag{23}$$

9. Ruminating Reader without query input in query ruminate layer. In this version, we discard query encoding input $\boldsymbol{Q}$ in the gate of query ruminate layer. Formally, the gate in Query Ruminate layer is

$$\boldsymbol{z}_i = tanh(\boldsymbol{W}_{Qz}^{1\top}\boldsymbol{S}_{Qi} + \boldsymbol{b}_{Qz}) \tag{24}$$

$$\boldsymbol{f}_i = \sigma(\boldsymbol{W}_{Qf}^{1\top}\boldsymbol{S}_{Qi} + \boldsymbol{b}_{Qf}) \tag{25}$$

$$\tilde{\boldsymbol{Q}}_i = (1 - \boldsymbol{f}_i) \circ \boldsymbol{z}_i \tag{26}$$

10. Ruminating Reader without context encoding input in context ruminate layer. We ablate the context encoding input $\boldsymbol{C}$ in the gate of context ruminate layer. Therefore, the gate in context ruminate layer is

$$\boldsymbol{z}_i = tanh(\boldsymbol{W}_{Cz}^{1\top}\tilde{\boldsymbol{S}}_{Ci} + \boldsymbol{b}_{Cz}) \tag{27}$$

$$\boldsymbol{f}_i = \sigma(\boldsymbol{W}_{f}^{1\top}\tilde{\boldsymbol{S}}_{Ci} + \boldsymbol{b}_{Cf}) \tag{28}$$

$$\tilde{\boldsymbol{C}}_i = (1 - \boldsymbol{f}_i) \circ \boldsymbol{z}_i \tag{29}$$

11. Ruminating Reader without summarization information input in query ruminate layer. In case that the summarization do not help the encoding, while on the other hand, the gate contributes to the learning, we design the experiment that allows to eliminate the influence of summarization. We discard the summarization input in query ruminate layer.

Formally, the gate in query ruminate layer is defined as

$$\boldsymbol{z}_i = tanh(\boldsymbol{W}_{Qz}^{2\top}\boldsymbol{Q}_i + \boldsymbol{b}_{Qz}) \tag{30}$$

$$\boldsymbol{f}_i = \sigma(\boldsymbol{W}_{Qf}^{2\top}\boldsymbol{Q}_i + \boldsymbol{b}_{Qf}) \tag{31}$$

$$\tilde{\boldsymbol{Q}}_i = \boldsymbol{f}_i \circ \boldsymbol{Q}_i + (1 - \boldsymbol{f}_i) \circ \boldsymbol{z}_i \tag{32}$$

12. Ruminating Reader without summarization information input in context ruminate layer. The summarization information is not included in $\boldsymbol{z}_i, \boldsymbol{f}_i$

$$\boldsymbol{z}_i = tanh(\boldsymbol{W}_{Cz}^{2\top}\boldsymbol{C}_i + \boldsymbol{b}_{Cz}) \tag{33}$$

$$\boldsymbol{f}_i = \sigma(\boldsymbol{W}_{Cf}^{2\top}\boldsymbol{C}_i + \boldsymbol{b}_{Cf}) \tag{34}$$

$$\tilde{\boldsymbol{C}}_i = \boldsymbol{f}_i \circ \boldsymbol{C}_i + (1 - \boldsymbol{f}_i) \circ \boldsymbol{z}_i \tag{35}$$