Coattention-Based Neural Network for Question Answering

Jim Andress **Cristian Zanoci**

CodaLab Username: jandress

Abstract

Machine comprehension (MC) and question answering (QA) are related NLP tasks which have seen increased interest with the recent release of the Stanford Question Answering Dataset (SQuAD). In this paper, we explore a neural architecture for the QA task which is based largely on the coattention encoder proposed by Xiong et al. [1]. After encoding the data with a context-aware representation of the question and question-aware representation of the context, we decode using simple linear decoders to construct the final probability distributions for the answer location within the context paragraph. Our best single model achieves an F1 score of 70.0% on the withheld test dataset, while an ensemble achieves 74.1% F1.

1 Introduction

Question Answering is a long-established problem in natural language processing. In this task, a computer system must respond to queries presented in natural language based on a context paragraph that contains the answer to the query. Question Answering systems have recently been gaining popularity and are improving quickly due to developments in recurrent neural network architectures and concepts such as the attention mechanism. Trained QA systems can be applied to a variety of problems from information retrieval in personal assistants like Siri or Cortana to even game shows, as was recently demonstrated by IBM's Watson system.

One factor which has recently driven much of the interest in QA research was the 2016 release of the SQuAD dataset [2]. The SQuAD dataset consists of 107,785 (question, context, answer) triples, with about 10,000 examples withheld from the public as a test set. The context paragraphs are drawn from 536 Wikipedia articles, while the questions and answers were generated by crowdworkers. The data was divided into a training set of 87,599 triples and a development set of 10,570 triple. We additionally use 5% of the training set for validation purposes, including hyperparameter tuning.

The original SQuAD paper proposed a simple sliding window baseline and slightly more complex logistic regression model which achieved 20% and 51% F1 scores, respectively. These scores pale in comparison to the average human performance of 86.8% F1 and recent results using deep learning techniques. Wang et al. proposed the Match-LSTM with Answer Pointer architecture [3], which adapted techniques originally introduced for the Natural Language Inference task. A series of stacked LSTM-RNNs provides an encoding for the question and context, while another RNN produces probability distributions for the next word in the answer. Wang et al. explored both a sequence model (selecting words the words in the answer one at a time) and a boundary model (selecting only the first and last word in the answer), and they found the boundary formulation of the QA task to produce significantly higher results.

Xiong et al. created the Dynamic Coattention model [1], which exploits the common deep learning technique of attention while encoding the context and question. In particular, this architecture begins

by computing an affinity matrix between each pair of words in the context and question, which is then used to weight the continuous representations of the two documents. In order to decode, their system alternatingly optimizes its estimates of the start and end location of the answer, allowing it to better handle local minima in the search space. An ensemble of these models is capable of achieving an 80.4% F1 score.

Finally, Seo et al. developed the Bi-Directional Attention Flow (BiDAF) model [4]. This model has a similar encoder to the Dynamic Coattention model in that they first compute an affinity matrix which is used to compute both context-to-query and query-to-context attention. However, the decoder and output layers for BiDAF are much simpler, merely passing the final representation of the context paragraph to an RNN and using a dot product to convert each of the resulting vectors to a single score. An ensemble of these models achieves an F1 score of 81.5%, the third highest score on the official SQuAD leaderboard at this time. As will be seen in the following section, we drew upon these last two papers in creating our model for the QA task.

2 Methods

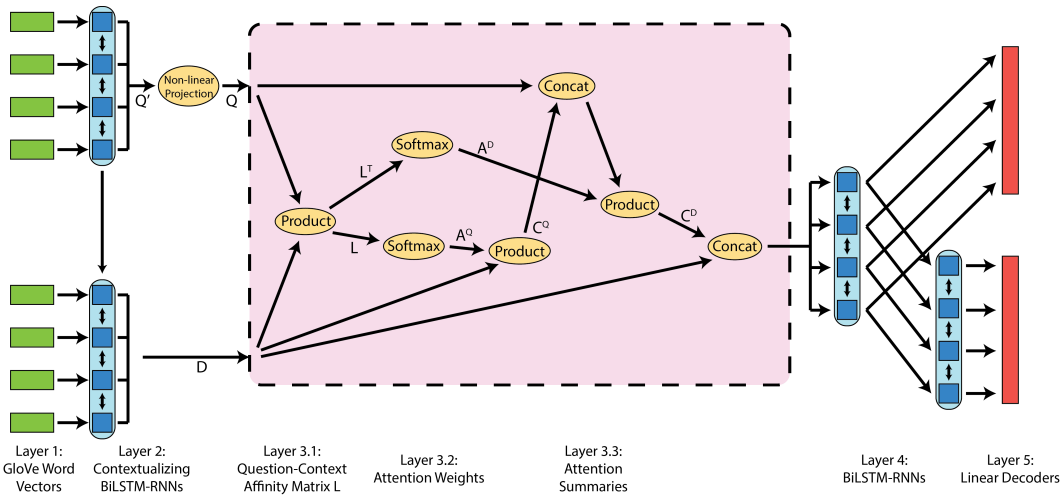


Figure 1: The architecture of final neural net model. The coattention encoder is boxed in pink.

The architecture of our deep learning model is diagrammed in Figure 1. The following sections will describe in detail the operation of each stage or layer of the pipeline.

2.1 GloVe Word Vector Embeddings

In order to convert from a sparse representation to a high-dimensional vector space we use word embeddings. In particular, in this paper we use pre-trained GloVe vectors to represent the words in the context and question.

2.2 Contextualizing Bidirectional RNN

Let $[q^{(1)}, q^{(2)}, \ldots, q^{(w_q)}]$ be the sequence of word vectors corresponding to the question, and similarly define $[d^{(1)}, d^{(2)}, \ldots, d^{(w_d)}]$ to be the word vector representation of the context document. Our first goal is to contextualize these word vectors by feeding them into a bidirectional recurrent neural network. We chose to use LSTM cells in this RNN, allowing the network to remember past features of the input as it encodes each word.

For notation, we will write $A = \overleftrightarrow{LSTM}(B; f, b)$ to indicate that the matrix A is comprised of the output states of the BiLSTM-RNN where each column of B is an input, f is the starting hidden state for the forward LSTM, and b is the starting hidden state for the backward LSTM. Using this

notation, the next step of our deep learning model is to compute

$$Q' = \overleftrightarrow{LSTM}([q^{(1)}, \ldots, q^{(w_q)}]; 0, 0) \in \mathbb{R}^{h \times w_q} \tag{1}$$

$$D = \overleftrightarrow{LSTM}([d^{(1)}, \ldots, d^{(w_d)}]; Q'[:, 0], Q'[:, w_q]) \in \mathbb{R}^{h \times w_d} \tag{2}$$

Note in particular that the two matrices are fed through the same LSTM in order to share representation power, and the final hidden states in the question encoding are fed in as the initial hidden states when encoding the context. This produces an initial question-aware encoding of the context and mimics the common reading comprehension test strategy of "reading the questions before the passage."

Finally, we make two small modifications to the context and question representations. First, we add to each a length h sentinel vector $d^{(\emptyset)}$ or $q^{(\emptyset)}$ so that in future stages the words are able to give attention to something besides the true set of words. This increases the dimensionality of Q' and D to $\mathbb{R}^{h \times (w_q+1)}$ and $\mathbb{R}^{h \times (w_d+1)}$, respectively. Finally, we wish to allow the question and document to lie in different projective spaces. Therefore, we add a non-linearity to the question encoding, giving us

$$Q = \tanh\left(W^{(Q)} Q' + b^{(Q)}\right) \in \mathbb{R}^{h \times (w_q+1)} \tag{3}$$

2.3 Coattention Encoder

Given the document matrix $D \in \mathbb{R}^{h \times (w_d+1)}$ and question matrix $Q \in \mathbb{R}^{h \times (w_q+1)}$, we next use the coattention pipeline in order to produce a final encoding of the context paragraph. The first step is to compute the affinity matrix

$$L = D^T W^{(L)} Q \in \mathbb{R}^{(w_d+1) \times (w_q+1)} \tag{4}$$

where $W^{(L)} \in \mathbb{R}^{h \times h}$ is a trainable weight matrix. Within L, entry (i, j) represents the similarity between word i in the context paragraph and word j in the question.

Next, L is normalized in order to produce attention weights for both the question and document representations. Namely, we compute

$$A^Q = softmax(L) \in \mathbb{R}^{(w_d+1) \times (w_q+1)} \tag{5}$$

$$A^D = softmax(L^T) \in \mathbb{R}^{(w_q+1) \times (w_d+1)} \tag{6}$$

where $softmax(\cdot)$ normalizes each row of the input independently. Essentially, the ith row of A^Q is a length $w_q + 1$ vector of weights describing how relevant each word in the question is to the ith word in the document. Similarly, the jth row of A^D is a length $w_d + 1$ vector of weights describing how relevant each word in the document is to the jth word in the question.

Then, using the attention weights, we produce the attention summaries for first the question, then the context paragraph. We start by computing

$$C^Q = DA^Q \in \mathbb{R}^{h \times (w_q+1)} \tag{7}$$

That is, for each word in the question we compute a weighted average over all the word vectors in the document, with the weights given by the row in A^Q corresponding to that word.

We next take the original encoding of the question, Q, and concatenate it with the attention summarized encoding of the question, A^Q, which is then used to compute the attention summary for each word in the context document

$$C^D = \left[Q; C^Q\right] A^D \in \mathbb{R}^{2h \times (w_d+1)} \tag{8}$$

where $[\cdot; \cdot]$ indicates that two matrices are stacked vertically. Just as was done with the question, the final step is to fuse the original encoding of the context paragraph D and the attention summarized encoding of the paragraph C^D, meaning that the final output of our coattention encoder is the matrix $[D; C^D]$.

2.4 Output Layers

The matrices D and C^D capture the representation of the context words conditioned on the question. However, in the previous step, the vectors for each context word are produced independently of each other. In order to capture the connection between the question-aware vectors of the context words, we apply a bidirectional LSTM

$$U = \overleftrightarrow{LSTM}([D; C^D]; 0, 0) \in \mathbb{R}^{3h \times (w_d+1)} \tag{9}$$

This LSTM is supposed to temporally merge information from different parts of the coattention context [1]. Since we no longer have a need for the sentinel vector $d^{(\emptyset)}$, we remove it by dropping the last column of U, so that now $U \in \mathbb{R}^{3h \times w_d}$

Next, we try to predict the start and end points of an answer span. Since the answer is a contiguous sub-array of the context tokens, by correctly predicting its start and end points we are guaranteed to retrieve the full answer. First, we use a linear projection to convert the vectors for every context word to a score which indicates how likely it is that the answer starts with that word. Then we apply a softmax to obtain the probability distribution of the start index i_s over the entire context

$$p(i_s) = softmax(W^{(s)}U) \tag{10}$$

where $W^{(s)} \in \mathbb{R}^{1 \times 3h}$ is a weight vector. We want to capture the fact that predicting the start and end indices should be treated differently, and therefore use another BiLSTM to project U onto a different hyperspace before transforming it into the probability distribution of the end index i_e

$$U' = \overleftrightarrow{LSTM}(U; 0, 0) \in \mathbb{R}^{3h \times w_d} \tag{11}$$

$$p(i_e) = softmax(W^{(e)}U') \tag{12}$$

where $W^{(e)}$ has the same dimension as $W^{(s)}$. Finally, we output

$$(i_s^*, i_e^*) = \underset{1 \leq i_s \leq i_e \leq w_d}{\operatorname{argmax}} \; p(i_s)p(i_e) \tag{13}$$

as our prediction for the answer span.

3 Exploration

3.1 Model

While building our model, we made several design decisions. First was the decision to feed the question representation in as the starting hidden state when computing the context representation, as described in Section 2.2. It intuitively made sense to us that an encoding of the context conditioned on the question would help our model perform better, and this was backed up by a roughly 0.5% score increase when we made this change.

The next feature unique to our implementation is the inclusion of the $W^{(L)}$ term in the affinity computation of Equation 4. In lecture and one of his papers [5], Prof. Manning discussed several scoring functions when computing attention weights for machine translation. These scoring functions included

$$score(h_t^T, \overline{h}_s) = \begin{cases} h_t^T \overline{h}_s & dot \\ h_t^T W_a \overline{h}_s & general \\ v_a^T \tanh(W_a[h_t; \overline{h}_s]) & concat \end{cases} \tag{14}$$

If we hadn't included the $W^{(L)}$ term (just as in the original), each of the affinity score computations would have been of the first form. However, as was recommended by Prof. Manning, we instead

chose the second bilinear form, which allows for more interactions between the terms in the two word vectors.

We also explored how to best decode after the coattention encoder. At first, our model did not include the second LSTM in the decode layers, meaning that we did not compute U' and instead found the probability distribution for the ending word as

$$p(i_e) = softmax(W^{(e)}U) \tag{15}$$

This model achieved an F1 score of 68.8% and EM of 59.1% on the test set.

Another decoder option that we implemented was the answer pointer layer, as described in [3]. However, this actually resulted in a decrease in performance, scoring only 64.8% and 55.4% on F1 and EM metrics when run on the dev set. This motivated us to go back to our original decoder and improve it with the addition of the LSTM layer before predicting the end index.

3.2 Experimental Setup

We start by tokenizing the contexts, questions, and answers using the Natural Language Toolkit (NLTK) tokenizer. We then proceed by embedding each word using the GloVe word vectors pre-trained on the 840B Common Crawl corpus [6]. We found that switching from the default 100-dimensional GloVe vectors to the larger 300-dimensional representation improved the performance across all models by roughly 20%. Embeddings are initialized randomly for words that are out-of-vocabulary, as well as for the two sentinel vectors in Q' and D.

As evidenced by Figure 2, our initial data analysis showed that more than 97% of the paragraphs in the training set have length at most 200. Therefore, to improve the training speed of our model, we truncate the longer paragraphs to 200 words (throwing out any examples whose answer occurs after word 200) and pad the shorter contexts accordingly.

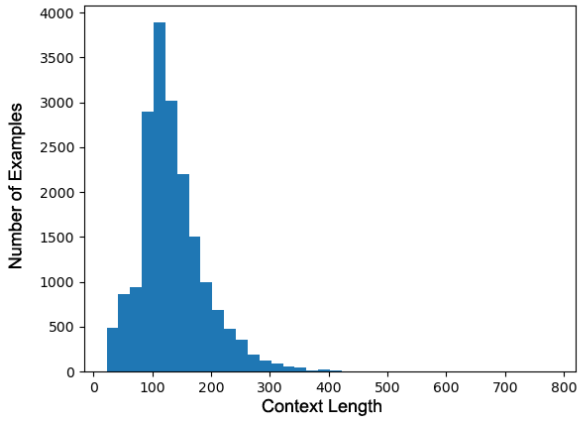


Figure 2: A histogram of the lengths of the context paragraphs in the training data.

To train our model, we use minibatch gradient descent with a batch size of 40 and an initial learning rate of 0.001. We have experimented with learning rates of 0.1 and 0.01, but found that our models did not perform nearly as well with those parameters. In order to improve the learning capability of our model in later epochs, we used an exponentially decaying learning rate with a decay rate of 0.88, making our final learning rate expression for a specific minibatch

$$\ell = 0.001(0.88)^{m/M} \tag{16}$$

where m is the total number of minibatches that have been processed during training and M is the number of minibatches in one epoch.

To avoid the "exploding gradients problem," we employ gradient clipping when the sum of all gradient norms exceeds 100. For our optimizer we use ADAM and train for 10 epochs. In most of the

cases, the best model is produced after epoch 6 or 7, with only minor changes occurring beyond that point.

The BiLSTM hidden layer size h was set to 400, i.e. the forward and backward hidden states are of size 200 each. All the weights and biases were initialized using Xavier initialization. In order to avoid overfitting, we apply a dropout of 0.2 to the outputs of all LSTMs. Our experiments showed that using a smaller dropout rate, such as 0.15, resulted in sub-optimal performance in which the model began overfitting after only around four epochs.

A heuristic similar to the one for context lengths is used to reduce the length of our selected answers. As can be seen in Figure 3, the answers usually span only a few words. Therefore, in order to prevent our models from outputting unrealistically large answers, we limit our prediction to no more than 15 words. This observation improved the performance of our models by about 1%.

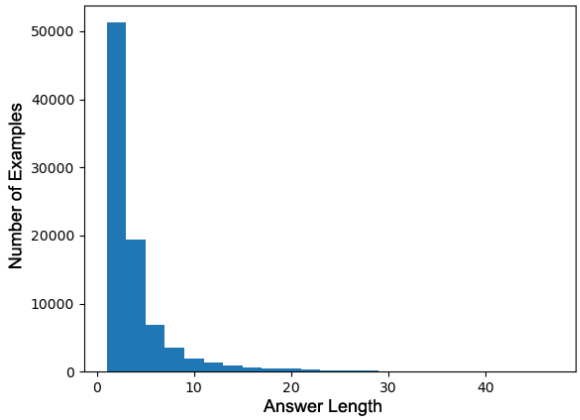


Figure 3: A histogram of the lengths of the answers in the training data.

It is worth mentioning that thanks to a smart implementation that uses a minimal number of expensive matrix manipulations, such as reshapes, our model only takes about 30 minutes per epoch to train, even though it has 10,900,000 parameters. Moreover, we use a curriculum learning approach [7], which means that we sort the contexts by length in every 20 batches to ensure that longer contexts are grouped together in a batch and shorter paragraphs don't get unnecessarily long padding. This speed-up allowed us to explore various hyperparameters, as well as train an ensemble for our model.

4 Results

The SQuAD task is evaluated using the F1 and Exact Match (EM) metrics. The F1 metric measures how well the predicted answer overlaps with the ground truth answer, while the EM metric simply measures whether or not the two answers were exactly the same. Our final F1 and EM values are shown in Table 1. Notice that we achieve the best result when predicting the answer using an ensemble of four models, where the probabilities for start and end indices are simply averaged across models.

Figure 4 shows how our model performed on several different types of questions. As is fairly standard for these types of deep learning QA systems, our model performs very well on simple "When" questions and has much lower performance on the more complicated "Why" questions.

Figure 5 shows how our model performs based on the answer length and context length. Notice that the performance decreases as the answer length increases. This shows that our model has more difficulties when trying to find longer answers. On the other hand, the F1 and EM scores of our model are more or less constant for paragraphs of length up to 350 words. Note that the behavior on longer contexts is not really representative since our data only has a few samples of that length. One

Model	Dev F1	Dev EM	Test F1	Test EM
Single output layer, no $W^{(L)}$ matrix	68.6	57.3	68.8	59.1
Final Model (single)	70.3	60.0	70.0	59.7
Final Model (ensemble of 4)	74.0	63.9	74.1	64.1
State of the Art (from Official SQuAD Leaderboard)	-	-	84.0	76.9
Human Performance [2]	90.5	80.3	86.8	77.0

Table 1: A table showing the F1 and EM scores of our various models (first three rows) as well as the current state of the art and human performance.

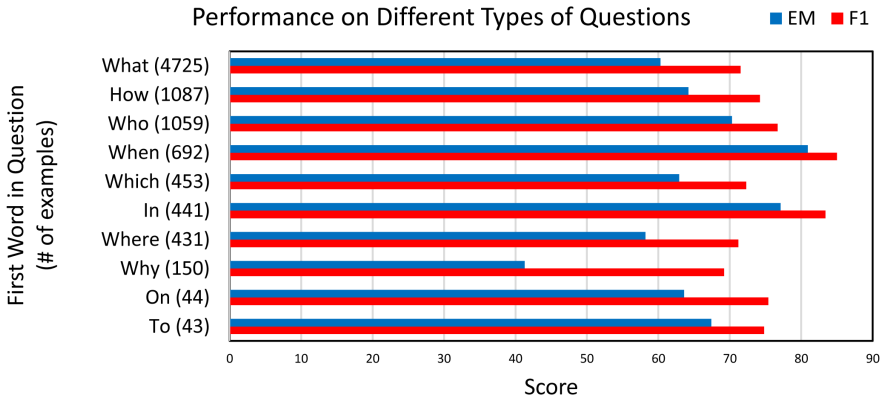
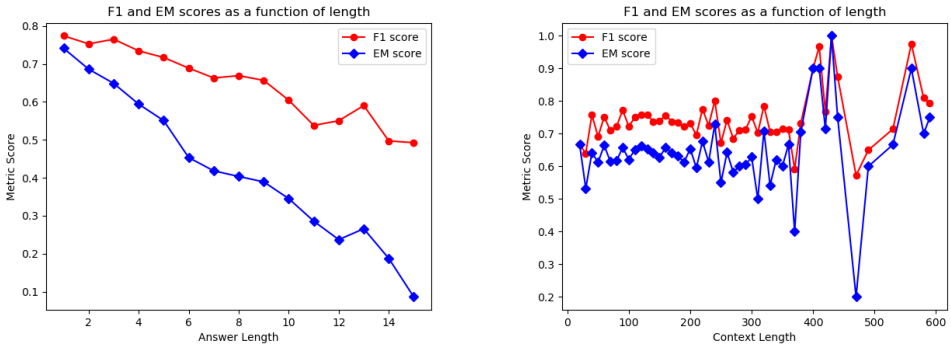


Figure 4: The break down of our model's performance based on the first word in the question.

might expect the performance to decrease when given a longer paragraph, since it would be harder to identify the correct answer. However, the plot shows that our model is able to ignore most of the irrelevant parts in the context and focus its attention on the smaller section containing the answer.

Figure 6 gives an example of our system in action. Given the context sentence "The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California." and the question "What venue did Super Bowl 50 take place in?" the resulting probabilities for the locations of the start and end of the answer are shown in the figure. In this case the system is very confident about its answer, but notice that it does give some weight to "San Francisco Bay Area" as a possible answer, suggesting that the model can still be led astray by related words (here, other locations in the same sentence).



(**a**) (**b**)

Figure 5: Performance of our model as a function of (a) answer length (b) context length.

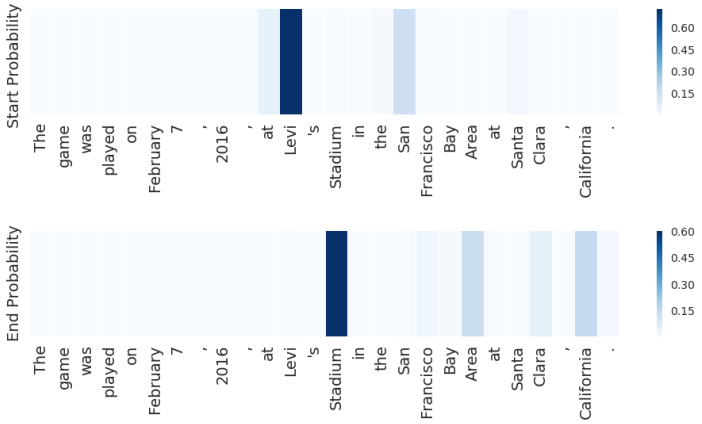


Figure 6: The probability distribution computed by our model for the location of the start and end word of the answer in the context when given the question "What venue did Super Bowl 50 take place in?".

5 Conclusions

We were able to build a successful model capable of answering questions presented to it in natural language. A primary take away from our project is the fact that even with a relatively simplistic decoder, we were still able to achieve high performance. However, it is likely that if we were to implement a decoder with more representation power, our results would improve even further. As of right now, we achieve a F1 score of 74.1 and an EM score of 64.1, which is on par with some of the entries on the official SQuAD leaderboard.

Another possible extension of this model could be to use more word features. Yu et al. [8] found that by adding POS and NE features to the standard GloVe vectors could result in an increase of 2% in performance.

References

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

[2] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

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

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

[5] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[6] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation.," in *EMNLP*, vol. 14, pp. 1532–1543, 2014.

[7] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, ACM, 2009.

[8] Y. Yu, W. Zhang, K. Hasan, M. Yu, B. Xiang, and B. Zhou, "End-to-end answer chunk extraction and ranking for reading comprehension," *arXiv preprint arXiv:1610.09996*, 2016.