Question Answering on SQuAD

Chenjie Yang
Department of Computer Science
Stanford University
yangcj@stanford.edu

Haque Ishfaq
Department of Statistics
Stanford University
hmishfaq@stanford.edu

Abstract

In this project, we exploit several deep learning architectures in Question Answering field, based on the newly released Stanford Question Answering dataset (SQuAD)[7]. We introduce a multi-stage process that encodes context paragraphs at different levels of granularity, uses co-attention mechanism to fuse representations of questions and context paragraphs, and finally decodes the co-attention vectors to get the answers. Our best model gets 62.23% F1 score and 48.72% EM score on the test set.

1 Introduction

Question Answering (QA) has become a popular field in natural language processing over the past few years. The limited size of previous QA datasets ([8] etc.) prevents the researchers from training data-intensive models like deep neural networks.

The recently released Stanford Question Answering dataset (SQuAD)[7] addresses the weakness of the previous datasets. It is orders of magnitude larger than all previous hand-annotated datasets, and is challenging: all the answers can be arbitrary spans within context paragraphs rather than a limited set of multiple choices.

The SQuAD dataset gives us a chance to develop more expressive and realistic models. These models typically consist of three parts: encoding layer, attention layer and decoding layer. One of the key progress in this area has been the use of attention mechanism[14][9], which extracts the most relevant part within a context paragraph to answer the question. Other works that attempt to empower the encoding or decoding process have also been published. For example, match-LSTM[13] model explicitly aggregates the matching of the attention-weighted premise to each token of the hypothesis when encoding textual information. RASOR[4] efficiently builds fixed length representations of all spans in the context paragraph with a recurrent network and produce answers based on scores of these spans.

In this project, we explore several neural network architectures for the SQuAD task. We begin at a baseline model that has basic encoding, attention and decoding layer. Then we try to add advanced components to each layer to boost the performance step by step. Our best model gets 62.23% F1 score and 48.72% EM score on the test set.

The rest of this paper is structured as follows: section 2 briefly describes the SQuAD dataset, section 3 discusses the architecture of our model in details, section 4 talks about the experiment settings and some special considerations in choosing hyper-parameters, section 5 shows the experiment results and result analysis.

2 Dataset

SQuAD is a new reading comprehension dataset, consisting of 100000+ question-answer pairs on 500+ articles. The answer to each question is always a span in the context paragraph. The model is trained to produce answers that can match one of the human written answers. SQuAD is significantly larger than previous reading comprehension datasets, and is also much more challenging because the answers do not come from a small set of candidate answers and they have variable lengths. The public dataset has two parts: 81386 question-answer pairs for training, and 4284 question-answer pairs for validation. Also, the SQuAD website contains a leaderboard for researchers to evaluate their models on a hidden development set and a hidden test set.

3 Architecture

In this section, we introduce our end-to-end neural architecture for question answering in details. An overview of our model can be seen in Figure 1. Basically, the model includes three parts: **encoding layer**, **attention layer** and **decoding layer**.

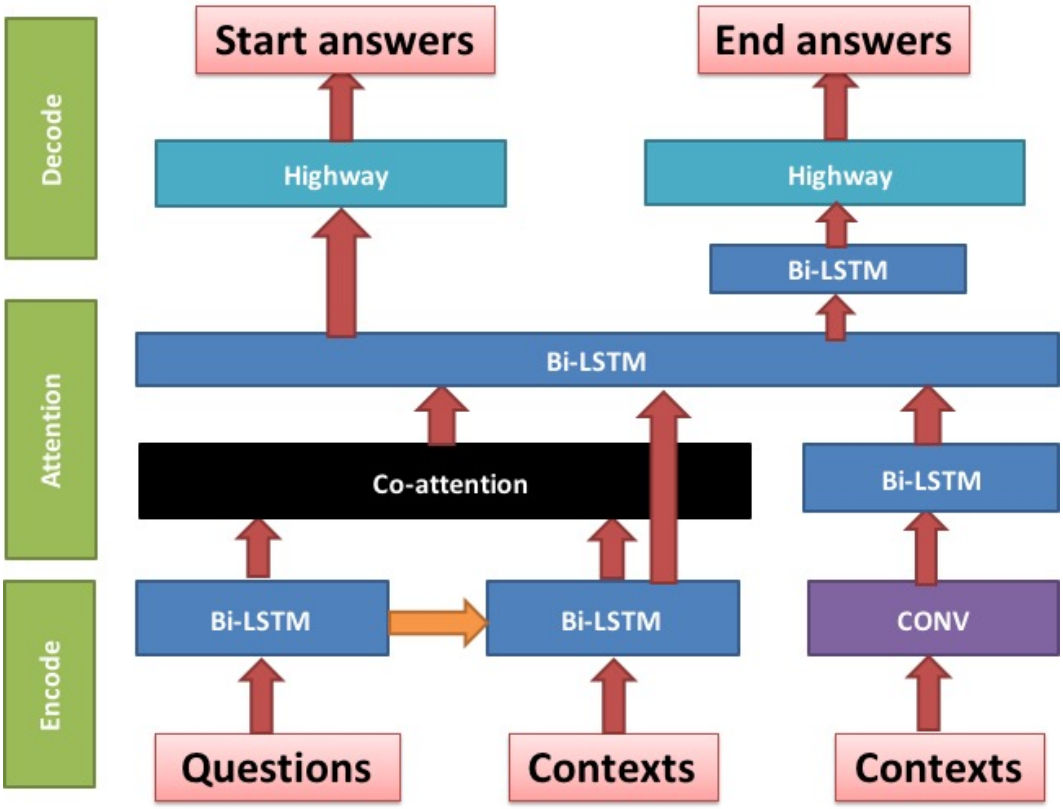

Figure 1: Architecture Overview

3.1 Encoding Layer

Word Embedding Layer maps each word to a d-dimensional vector space using a pretrained word embedding model. Here we use GloVe [6] to initialize the model. Words not found in Glove will be mapped to random vectors. The word embeddings are not updated during training.

Convolution Layer applies convolution filters[2] with different filter widths on the contexts to capture coarse grain semantics, and possibly phrase representations.

Contextual Embedding Layer feeds embeddings of questions, contexts, convolutionized contexts to a shared Bi-LSTM to encode the sequence or temporal information of these sentences. To enable conditional encoding of context paragraphs (same paragraph will be encoded differently if attached to different questions), we use the final states of the question encoding as the initial states of context/convolutionized context encoding.

3.2 Attention Layer

We use the co-attention mechanism proposed in [14] that attends to the questions and contexts simultaneously. Let $Q = \{q_1, q_2, \cdots, q_m\} \in R^{n \times 2h}$ denote the encoding matrix of questions, where h is the hidden size of cells in the encoding Bi-LSTM. $C = \{c_1, c_2, \cdots, c_m\} \in R^{m \times 2h}$ denote the encoding matrix of contexts. Then the affinity scores corresponding to all pairs of question words and context words are: $L = CQ^T \in R^{m \times n}$. Then we use softmax to normalize it.

$$A^C = softmax(L) \in R^{m \times n}, A^Q = softmax(L^T) \in R^{n \times m} \tag{1}$$

Next we compute the summaries of the context in light of each word in the question, and similarly compute the summaries of the question in light of each word in the context.

$$C^Q = A^Q C \in R^{n \times 2h}, Q^C = A^C Q \in R^{m \times 2h} \tag{2}$$

Then we map the question encoding back into space of context embedding, and concatenate it with Q^C to get a co-attention vector.

$$C_o = [Q^C; A^C C^Q] \in R^{m \times 4h} \tag{3}$$

Now we have C_o, a co-dependent representation that encodes both context-to-question attention information and question-to-context attention information. We concatenate it with the encoding matrix of contexts $C \in R^{m \times 2h}$ and the encoding matrix of convolutionized contexts $C_C \in R^{m \times 2h}$ to form the final co-attention vector $Att = [C_o, C, C_c] \in R^{m \times 8h}$. Finally, we feed the co-attention vector Att to another Bi-LSTM for the fusion of temporal information. The output of this Bi-LSTM will be fed into the decoder layer to predict answers.

This attention mechanism is simple but powerful. We also explored other promising attention mechanisms like match LSTM in [13], and bi-directional attention flow in [9]. However, after we implemented them, we found that: match LSTM was slow to train; bi-directional attention flow would consume a lot of memory for calculating the attention vector and tend to cause out of memory error even if we use small $batch\_size$ (bi-directional attention method needs to calculate a matrix whose dimension is $batch\_size \times question\_length \times context\_length \times hidden\_size$, which is huge). So finally we chose to use the simple co-attention mechanism described above.

3.3 Decoder Layer

Since in SQuAD dataset, the answer to each question is always a span in the context paragraph. A natural way for producing answer span is by predicting the start and end points of the span [12]. This is referred to as a *boundary* model. Another kind of model is the *sequence* model, where the answer is represented by a sequence of integers indicating the positions of the selected words in the context paragraph. Here we decide to choose boundary model since: 1.Several previous works (like [13]) have shown that the performance of the boundary model is comparable to the sequence model (even better in most cases). 2.Decoder using a boundary model is typically much smaller than that using a sequence model, which means the training process could be faster.

At first we only used a baseline decoder: we feed the co-attention vector Att into a fully-connected layer followed by a non-linear relu layer to further extract features. The features will then be put into the softmax layer to predict the answers.

After getting baseline results, we tried to use more advanced models. Motivated by [11] and [13], we implemented the Answer Pointer layer as our decoder. However, the performance was not good.

Then we replaced the fully-connected layers in decoder with highway network layers[10], since highway network layers allow unimpeded information to flow across and maintain gradients of parameters. Also, when predicting end points, we add another Bi-LSTM layer to further extract the temporal information of co-attention vector. We observed a performance boost after using this decoder.

3.4 Predicting answers

The model predicts the start and end points of answers separately. It's possible that the predicted end index of answer is larger the predicted start index. To avoid this and to make the predicting process more robust, we incorporate a search mechanism when producing answers: the answer span (k, l) where $k \leq l$ with the maximum value of $p_k^1 p_l^2$ is chosen. This can be solved in linear time with dynamic programming.

4 Experiments

4.1 Data Preprocessing

We first tokenize all the passages, questions and answers using NLTK. Then we use word embeddings from GloVe[6] to map words into embedding vectors. To decrease the out of vocabulary (OOV) error, we use the Common Crawl 840B 300d GloVe vectors. Words not found in GloVe are initialized randomly. The word embeddings are not not trained during training. Since the GPU memory in Azure is limited, and some very long questions/contexts will cause OOM (out of memory) error if we use complex models, we limit the max question length to be 40 and the max context length to be 500. Longer question/context are truncated. Among 81386 training pairs, only 2 questions and 25 context paragraphs need to be truncated.

4.2 Model Details

For the best model we have (as shown in Figure1): The hidden state size h of all Bi-LSTM models is 256. The convolution layer uses 2,3,4,5 filter widths, each with 75 filters. We use the Adam[3] optimizer, with a mini-batch size of 50 and an initial learning rate of 0.002. The learning rate has a decay rate of 0.5 every epoch.

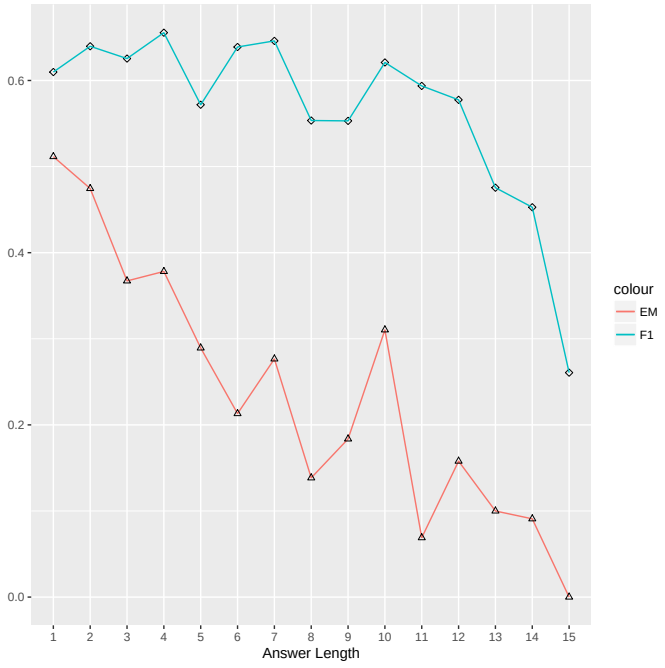

Figure 2: Variations of CCHNS performance on ground truth answer length (up to 15).

In practice, we found that our model starts to overfit quickly after 2-3 epoches. We applied dropout to all LSTM layers and highway network layers to prevent overfitting. However, we found that it's hard to fine-tune the dropout rate. A higher dropout rate will seriously impact the model's performance.

Table 1: Model Performance

Models	Val F1	Val EM	Dev F1	Dev EM
Baseline2	36.41%	23.6%	/	/
Co-attention	62.16%	45.6%	55.84%	43.55%
Co-attention+conv	65.38%	46.4%	59.59%	46.09%
Co-attention+conv+pointer	61.05%	44.1%	/	/
Co-attention+conv+highway	66.73%	47.6%	61.17%	46.95%
Co-attention+conv+highway+search	**68.63%**	**49.6%**	**61.72%** (**62.23% test**)	**47.92%** (**48.72% test**)

Finally, we chose to use a dropout rate of 0.1. Also, we apply an early stopping mechanism: the model will stop training if the overall validation loss starts to increase.

5 Results

5.1 Performance

We explored several models, as described in 3. Table1 shows the performance of these models. We submitted the best model to the test set leaderboard and it achieves **62.23%** F1 and **48.72%** EM scores. Our best model was combination of encoder using co-attention and decoder using convolutional neural network, highway network and search mechanism. For brevity, we denote our best model as CCHNS (Co-attention and Convolutional Highway Network with Search).

5.2 Error Analysis

To better understand our best system CCHNS, we performed various qualitative analysis for our best model.

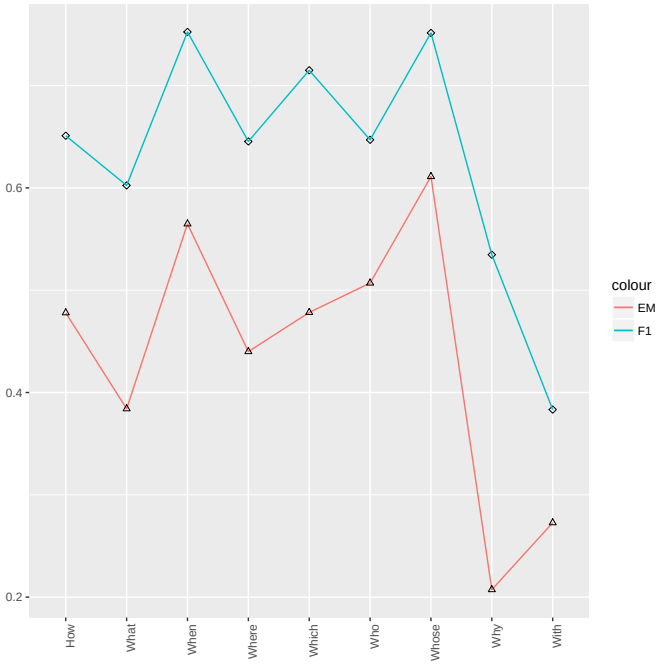

Figure 3: Performance comparisons for different question head word.

Performance across length To understand the system's performance while predicting answers of different lengths, we show the F1 scores and the exact match (EM) for answers up to 15 tokens in Figure 2. As in the case for most NLP applications such as neural machine translation [5], we expect the model performance to worsen as the answer length increases. However, from Figure 2, we see that there is no notable degradation in F1 score for answers of up to length 12. But after that we see a consistent fall in accuracy for both EM. This is intuitively expected since it becomes more challenging to compute the correct answer span as the number of words increases. From the graph, we see that as the lengths of the answers increases, both EM and F1 scores drops. We also observe that the accuracy for EM and F1 drop in different speed and the gap between F1 and EM widens as the answer length increases.

Performance across question head We also examine our model's performance across common question head words. In Figure 3, we note that the F1 score for questions starting with common question words such as "how","what", "when" etc are pretty high. But we also observe that our model performs realy poorly for questions starting with words "why" and "with". Also we see a dramatic difference between F1 and EM for "why" questions. This indicates that for this type of questions, it is easier to locate the core of the answer but it is much harder to identify the exact span of the answer. It is interesting to note that other published state-of-the-art models such as [14], [9], [15] also perform poorly on questions starting with "why" indicating the inherent difficulty in achieving high accuracy for this question type.

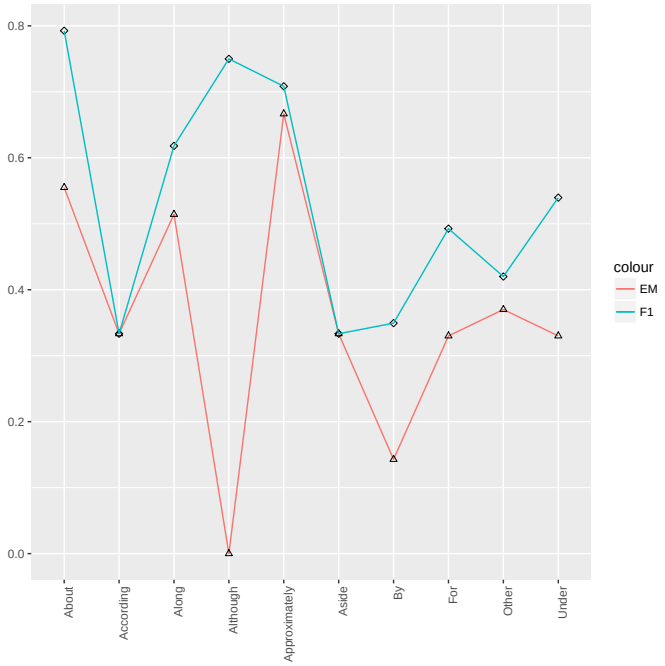

Figure 4: Performance comparisons for non-interrogative question starts.

Performance across questions with non-interrogative head While it is common for existing literature to analyze performance across questions starting with different interrogative words, we could not find any literature that examined the performance of their system for questions starting with non-interrogative word. These questions are usually known as non-interrogative questions [1] and they are usually more complex and rhetorical in nature. We analyzed our model's performance for such questions and show result for several of them in Figure 4. Even though our model performance varies widely across different start words here, we can still observe some interesting phenomena. For example, from Figure 4, we see that our model performs really well for questions starting with "about" and "approximately". Our most interesting observation is the dramatic difference between F1 score and EM for questions starting with "although". From our day-to-day usage in English, we can easily imagine that questions starting with "although" will be structurally complex sentence.

From our model's performance on this type of question, we see that it is exceedingly difficult to get the exact answer in this case, but our model does a really good job in getting the gist of the answer. We envision that future works focusing on improving performance for non-interrogative questions will lead to systems capable of understanding intricate meaning of natural language.

6 Future Work

In the future, we will try to find mechanisms to handle answers with long spans, and handle answers start with non-interrogative heads.

7 Acknowledgments

We sincerely thank the instructors and TAs of the CS224n course to help us learn NLP and make progress in this project! We also thank Microsoft Azure for providing access to their GPU system.

References

[1] Peter Bull. On identifying questions, replies, and non-replies in political interviews. *Journal of language and social psychology*, 13(2):115–131, 1994.

[2] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[3] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] Kenton Lee, Tom Kwiatkowski, Ankur Parikh, and Dipanjan Das. Learning recurrent span representations for extractive question answering. *arXiv preprint arXiv:1611.01436*, 2016.

[5] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[6] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[7] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[8] Matthew Richardson, Christopher JC Burges, and Erin Renshaw. Mctest: A challenge dataset for the open-domain machine comprehension of text. In *EMNLP*, volume 3, page 4, 2013.

[9] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[10] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[11] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[12] Shuohang Wang and Jing Jiang. Learning natural language inference with lstm. *arXiv preprint arXiv:1512.08849*, 2015.

[13] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*, 2016.

[14] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*, 2016.

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