Implementation and Improvement of Match-LSTM in Question-Answering System

Yundong Zhang[1] and Haomin Peng[2]

[1]CS224N: Natural Language Processing with Deep Learning
[2]CodaLab Username: Ben

Abstract

In this paper, we tackle the popular machine comprehension task derived from Stanford Question Answering Dataset (SQuAD), which consists of more than 100 thousand questions whose answers are segments of text from Wikipedia articles. We implemented the Machine comprehension system based on the model of[2], and made two extensions on top of that: 1. incorporate the word-level similarity in calculating attention; 2. use bi-attention (question-to-context and context-to-question) mechanism to infuse the information flow to the decoder. In our experiment, these two extensions improve the F1 score by 3% and EM by 2.3%. Our model ranks among top 10 in the test leaderboard by the end of March 20th, 2017.

1 Introduction

With the increasing demands of high-performance automatic text processing system, natural language reading comprehension has become an interesting topic which has great potential to be applied to a variety of tasks from different fields. To better boost the development of technology meanwhile providing a standard platform for evaluating the effectiveness of all the proposed methods came up from academia and industry, Stanford researcher had launched a dataset—the Stanford Questions Answering Dataset (SQuAD)[1] in 2016, which consists of a large set of passage, question and answer tuple created by first extracting massive paragraphs from Wikipedia articles then designing questions according to that context and labeling regions of the original paragraph as standard answer. The significant difference between SQuAD and other benchmark questioning and answering dataset is that all the questions and their answer span in context are created by human annotation, making the dataset more of a golden standard to examine how well a natural language processing (NLP) could work to tackle this task of machine comprehension. To challenge the performance on SQuAD, researchers are encouraged to design a NLP model which could predict the answer span in the original passage given questions generated from this paragraph.

By far, multiple well-designed models have been proposed and submitted to the SQuAD Leaderboard [2, 3, 4, 5], most of them being able to attain more than 70% F1 score and 60% exact match score. Unsurprisingly, the mainstream models rely on applying neural network frameworks, which typically consists of an encoder and a decoder. The basic structure of encoder is usually a bi-LSTM[8] or CNN[11] applied on both of the context and question, so that it could capture information of the word sequence while not neglecting the information provided by a reverse word sequence for the context and question. Mnih et al. raised this framework on computer vision [9] and Hermann et al. introduced this mechanism to machine comprehension [10]. It has the advantage of easily training since that all the parameters could be shared among each step. However, this framework on itself is still not sufficient for a good machine comprehension system, because it simply concatenate the hidden vectors from the context and question without considering the fact that not all the regions in context is of even importance to the question.

To simulate the unevenness in attention as human reading, researchers have introduced attention mechanism into the framework prototype which works by adding another layer that bear the information of how much each word in passage is related to each word in question, namely, attention weight, and feed this message into decoder as well. This type of structure, named sequence-to-sequence model, has gained huge success in multiple machine comprehension task (e.g. Machine translation[12], text generation[13], image labeling[14] and etc). Recently, A Match-LSTM (Wang & Jiang2016) model implemented this for SQuAD by feeding the traditional word to word attention weight into a one directional LSTM layer after doing certain transformations. This model aims to capture the sequential message in the attention weight and reported to have achieved an exact match score of 67.9% and an F1 score of 77.0% on the SQuAD black box test. A Bi-directional Attention Flow (BIDAF) (Seo at el , 2017)[5] further explored use of attention mechanism in machine comprehension by combining both the attention weight from passage to question and from question to passage. Other ways of fusing the representations of the question and the context paragraph have been addressed in the literature, include the Dynamic Coattention Network (DCN)[4]. However, considering the inequality in length of passage and question (passages are always much more longer than questions), just repeating the same process reversely may not reflect that dissymmetry and add to the amount of parameters at the same time. Excited by the tasks and datasets and inspired by those well-performed system, we built a similar model with the Match-LSTM paper and incorporated multiple components from other system, in order to boost the performance and evaluate the effectiveness of these small elements.

In this report, we implement the Match-LSTM model, and make two extensions to further power up the model performance: first we compute the cosine similarity between words in passage and question, to increase the performance of attention mechanism; second, besides computing the question-to-context attention, we also calculate the context-to-question attention to increase the information flow to Answer Pointer Layer.

2 Model

Our question-answering system consist of three hierarchical layers:

- (1). Features Extraction Layer: maps each word to its embeddings and extract the word similarity, contextual information as well as temporal interactions between words
- (2). Bi-attention Layer: computes the question-to-context attention and the context-to-question attention and outputs the weighted information
- (3). Answer Pointer Layer: utilizes the weighted information to indicate the possibility of each word in passage being start point or end point of the answer.

The overview of our model is presented as in fig. 1, we will show the details of each layer after that.

(1). Features Extraction Layer. Features Extraction Layer is responsible for transforming questions and passages into something that can capture contextual information and relation, or in a simple words, mapping the words to meanings. There are mainly two steps in our model: first maps words to a high-dimensional vector space; secondly we calculate the cosine similarity between words in passage and question; finally we apply a bi-directional LSTM[8] to both the question and passage separately to obtain contextual representations.

For the mapping of words to embeddings, we use the popular pre-trained word vectors, GloVe[6], which is able to capture word meanings as well as similarities and has been proved success in various tasks. Formally, we are given a set of (question, passage, answer) tuples. For each tuple, let $\mathbf{p} = (p_1, p_2, ..., p_m)$ denote a passage of m words, then the glove mappings returns matrix $\mathbf{P} \in \mathbb{R}^{o \times m}$ for the context, where o is the dimension of embedding vectors. Similarly, we have matrix $\mathbf{Q} \in \mathbb{R}^{o \times n}$ for the question, where n is the corresponding length.

To compute the cosine similarity, for each word i in passage and word j of question, we simply calculate $r_{i,j} = \frac{\vec{P}_i^T \vec{Q}_j}{|\vec{P}_i||\vec{Q}_j|}$, where $\vec{P}_i$ is the i^{th} column of $\mathbf{P}$ and $\vec{Q}_j$ is the j^{th} column of $\mathbf{Q}$. We then get a relevancy matrix $\mathbf{R}$ of shape $m \times n$. The reason for using the cosine similarity is to provide textual similarity to the following attention layers. Intuitively, more attention should be paid

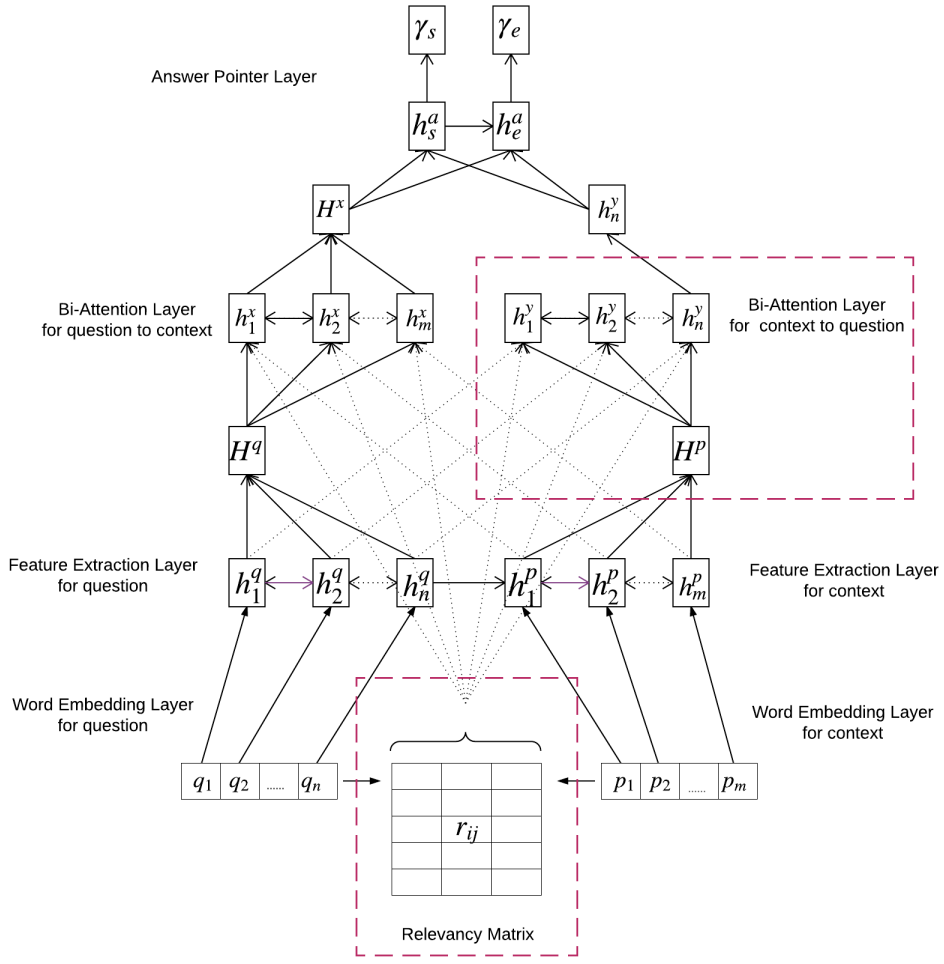


Figure 1: Architecture of our Bi-attention Match-LSTM model (the red boxes highlight our main improvement to Match-LSTM model)

if two words in passage and question are similar in the vector space. Unlike [3], we did not use the relevancy matrix to modify the context embedding matrix, as in our experiment this harms the model performance. The usage of relevancy matrix will be discussed soon.

For the last step, we obtain the hidden representation of each word in the passage and question, denoted as $H_p \in \mathbb{R}^{2d \times m}$ and $H_q \in \mathbb{R}^{2d \times n}$, where d is the hidden size. Note that $2d$ comes from the concatenation of the forward and backward LSTM. It is worthy to mention that these two steps are served as features extraction of the texts, and we can see that they are also performed in almost every successful models. Additionally, it is optional to use the last hidden state of question as the initial state of encoding paragraph, in our experiment this has almost no effect to the performance.

(2). Bi-attention Layer. The Attention Layer functions as a highlighter between question and passage. We extend the idea of the match-LSTM model[2] to a bi-directional attention, where we compute the attention scores of both question-to-context and context-to-question in a similar way. For question-to-context: the question is treated as a premise to the context. We simply go through the passage word by word, and for each word i we compute:

$$\begin{aligned}\overrightarrow{\mathbf{X}}_i &= \tanh\left(\mathbf{W}^{q_x}\mathbf{H}^q + (\mathbf{W}^{p_x}\mathbf{h}_i^p + \mathbf{W}^{r_x}\overrightarrow{\mathbf{h}}_{i-1}^x + \mathbf{b}^p) \otimes \mathbf{e}_n\right) \\ \overrightarrow{\alpha}_i &= \text{softmax}\left(\mathbf{w}_x^T\overrightarrow{\mathbf{X}}_i + k_x\mathbf{R}_{i,:} + b_x \otimes \mathbf{e}_n\right),\end{aligned} \tag{1}$$

where $\mathbf{e}_t$ denotes repeating the vector or scalar for t times, $\mathbf{W}^{q_x}, \mathbf{W}^{p_x}, \mathbf{W}^{r_x} \in \mathbb{R}^{2d \times 2d}$, $\mathbf{b}^p, \mathbf{w}_x \in \mathbb{R}^{2d}$ and $k_x, b_x \in \mathbb{R}$ are learnable parameters, $\overrightarrow{h}^x_{i-1} \in \mathbb{R}^{2d}$ is the hidden state of the match-LSTM layer, which is obtained by:

$$\overrightarrow{\mathbf{h}}^x_i = \overrightarrow{LSTM}\left(\overrightarrow{\mathbf{z}}_i, \overrightarrow{\mathbf{h}}^x_{i-1}\right), \tag{2}$$

and the vector $\overrightarrow{\mathbf{z}}_i$ is the concatenation of the hidden representation of current words and the weighted hidden representation of the question:

$$\overrightarrow{\mathbf{z}}_i = \begin{bmatrix} \mathbf{h}^p_i \\ \mathbf{H}^q \overrightarrow{\alpha}^T_i \end{bmatrix} \tag{3}$$

Intuitively, $\overrightarrow{\alpha}_i \in \mathbb{R}^n$ captures the relevancy between the i^{th} token of paragraph and the j^{th} token of the question, and it is used to 'filter' the hidden question representation, which is then fed into an LSTM together with the current passage word representation. The use of LSTM is to let it automatically store or discard the concatenated information, which is a perfect fit of LSTM. After going through the whole passage, we obtain $\overrightarrow{\mathbf{H}}^x \in \mathbb{R}^{2d \times m}$ representing the collections of $\overrightarrow{\mathbf{h}}^x_i$ for $i \in [1, m]$. We then repeat this process with the reverse direction of the paragraph and attain $\overleftarrow{\mathbf{H}}^x$. Finally, we arrive at a hidden attended representation of paragraph:

$$\mathbf{H}^x = \begin{bmatrix} \overrightarrow{\mathbf{H}}^x \\ \overleftarrow{\mathbf{H}}^x \end{bmatrix} \tag{4}$$

The extension we made is to use the similar mechanism but compute the context-to-question attention. Formally, we do:

$$\begin{aligned} \overrightarrow{\mathbf{Y}}_j &= \tanh\left(\mathbf{W}^{p_y}\mathbf{H}^p + (\mathbf{W}^{q_y}\mathbf{h}^q_j + \mathbf{W}^{r_y}\overrightarrow{\mathbf{h}}^y_{j-1} + \mathbf{b}^q) \otimes \mathbf{e}_m\right) \\ \overrightarrow{\beta}_j &= \text{softmax}\left(\mathbf{w}^T_y \overrightarrow{\mathbf{Y}}_j + k_y \mathbf{R}_{:,j} + b_y \otimes \mathbf{e}_m\right) \end{aligned} \tag{5}$$

where $\mathbf{W}^{q_y}, \mathbf{W}^{p_y}, \mathbf{W}^{r_y} \in \mathbb{R}^{2d \times 2d}$, $\mathbf{b}^q$, $\mathbf{w}_y \in \mathbb{R}^{2d}$ and $k_y, b_y \in \mathbb{R}$ are again learnable parameters, $\overrightarrow{h}^y_{j-1} \in \mathbb{R}^{2d}$ is the hidden state of the question-to-context match-LSTM layer. Similarly we also have:

$$\begin{aligned} \overrightarrow{\mathbf{h}}^y_j &= \overrightarrow{LSTM}\left(\overrightarrow{\mathbf{v}}_j, \overrightarrow{\mathbf{h}}^y_{j-1}\right) \\ \overrightarrow{\mathbf{v}}_j &= \begin{bmatrix} \mathbf{h}^q_j \\ \mathbf{H}^p \overrightarrow{\beta}^T_j \end{bmatrix} \end{aligned} \tag{6}$$

Instead of concatenate each hidden state as in question-to-context query, we only keep the last hidden state $\overrightarrow{\mathbf{h}}^y_n$ and $\overleftarrow{\mathbf{h}}^y_n$. The reason for doing this is compared to passage, question has a much smaller length, thus in theory the last hidden state is enough to capture all the attention information. Moreover, this allows us to incorporate the question-to-context information much easier in the following layers. Similarly, we concatenate the hidden representation of both direction and obtain $\mathbf{h}^y_n$.

(3). Answer Pointer Layer. We use the idea inspired by the Pointer Net model[7] to predict the start and end indices of the answer. The extension we made is besides feeding the attended representation of passage $\mathbf{H}^x$, we also input the attended representation of question $\mathbf{h}^y_n$ to the network. Formally, we have:

$$\begin{aligned} \mathbf{G}_s &= \tanh\left(\mathbf{W}^x\mathbf{H}^x + (\mathbf{W}^a\mathbf{h}^a_s + \mathbf{W}^y\mathbf{h}^y_n + \mathbf{b}^a) \otimes \mathbf{e}_m\right) \\ \gamma_s &= \text{softmax}\left(\mathbf{v}^T\mathbf{G}_s + c \otimes \mathbf{e}_m\right), \end{aligned} \tag{7}$$

where $\mathbf{W}^x, \mathbf{W}^y \in \mathbb{R}^{4d \times 2d}$, $\mathbf{b}^a, \mathbf{v} \in \mathbb{R}^{2d}$ and $c \in \mathbb{R}$ are learnable parameters, $\mathbf{h}^a_s$ is the hidden state of Answer Pointer Layer.

Intuitively, the neural net look at the attended representation of passage and question, output a probability distribution $\gamma_s \in \mathbb{R}^m$ of each word to be the start index. We then feed this information to an LSTM to obtain:

$$\mathbf{h}^a_e = LSTM\left(\mathbf{H}^x\gamma^T_s, \mathbf{h}^a_s\right), \tag{8}$$

Table 1: Model Results. "C" indicates fusing the cosine similarity to Attention Layer, and "B" indicates fusing bi-attention to Answer Pointer Layer

Model (single)	**hidden size** d	**training dropout** r	**F1**	**EM**
Logistic Regression[1]	-	-	51.0	40.0
Match-LSTM (re-implement)	75	0.6	60.708	47.323
Match-LSTM + C	75	0.6	61.332	47.698
Match-LSTM + C	100	0.6	62.105	48.103
Match-LSTM + C + B	100	0.6	62.771	48.609
Match-LSTM + C + B	100	0.5	**63.712**	**50.210**
Match-LSTM + C + B (test set)	100	0.5	**64.684**	**50.561**

Then we use the same mechanism again and obtain γ_e, with which we make prediction.

In our model, we minimize the cross-entropy loss to train the network, where:

$$CE_{loss} = -\sum_{i}^{samples} \mathbf{a}_s^i \log\left(\gamma_s^i\right) + \mathbf{a}_e^i \log\left(\gamma_e^i\right), \tag{9}$$

and $\mathbf{a}_s^i, \mathbf{a}_e^i$ is the true one-hot label of the i^{th} answer, corresponding to the start and end index.

3 Experiments and Discussion

3.1 Results

Our result is listed in table 1. The original implementation of Match-LSTM Answer-Pointer model achieves an F1 score of 71.2 and EM of 61.1. However, following their implementation, we are not able to reproduce the result as well as they did. Part of the reason might be we did not tune the model well enough, due to the limit of time and resources as this is a course project.

3.2 Data Preprocessing

We use the Stanford Question Answering Dataset (SQuAD) to conduct our experiments. There are in total 23,215 passages and 107,785 questions (there could be multiple question relating to the same passage). The data has been split into a training set, a validation set, a development set and a hidden test set. Based on our course requirements, we will use dev set as evaluation and validation set as hyper-parameters tuning set. We first examine the length distribution of our data in fig. 2:

As shown in fig.2, we notice that there are some passages whose length are extremely long and deviate from typical length. Allocating the size of input vectors according to the maximum size of passage would definitely harm the training speed (as well as the memory usage). Therefore we cut down paragraph length to 256. Similarly, we limit the question length to be 20. We found that this procedure only affect 2.5% of the whole dataset, while it drastically increases the training speed and frees memory usage a lot.

To initialize the word vector embeddings, we used the GloVe word embeddings of dimensionality $o = 300$ and vocabulary size of 2.2M that have been pre-trained on Common Crawl. We did not train the word embeddings, since our dataset is not very large. In our early experiment, we also found that training word embeddings are bad for generalization (since in the dev set and test set, we have unseen words).

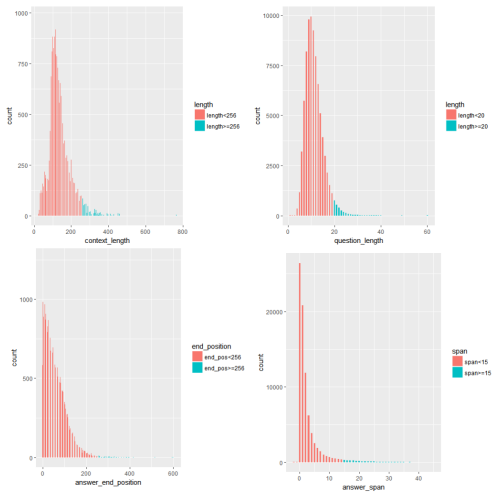


Figure 2: Length distribution for context,question,answer and the distribution of answer ending position in passage

3.3 Experiment Details

In our implementation, we use Adamax Optimizer instead of Adam[15]. According to the paper, Adamax is good for sparse parameters update (e.g. embeddings) and less susceptible to gradient noise. Although due to time constraints we did not find whether the choice of Adamax or Adam has direct relation to the evaluation score, our observation is that Adamax converge faster (5-7 epoch) than Adam (9-11 epoch). In both cases, learning rate is quite important. With a learning rate of 0.01, the model can not even converge. We find that a learning rate of either 0.001 or 0.002 is good. The learning curve is plotted below:

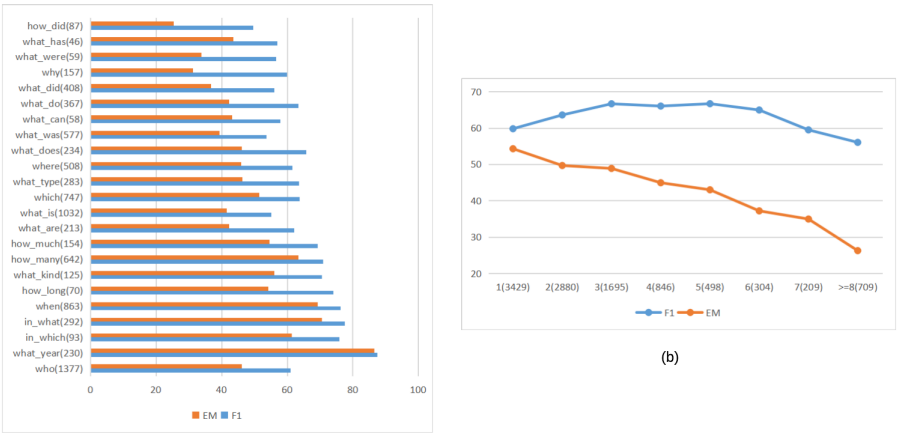
We use dropout rate of 0.4. We have tried dropout rate of 0.2 and that does reduce the generalization of the model–EM of dev set reduce from 48 to 46.

3.4 Result Analysis

To better understand the behavior of our model when tackling different tasks, we did further analysis on prediction results of dev set.

Figure 2(b) shows the relation between performance, including F1 score and exact match score ,and answer length (the number in bracket indicates how many answer of this length is in dev set). It is obvious from the figure that both of the scores drop as the answer length increase. For example, the F1 score of questions with more than 8 words drops to below 60% and the EM score is only around 30%, while the F1 score for shorter answer questions could be nearly 70% and EM score 55%. Another phenomenon is that the EM drops much quicker than F1. This shows that finding the approximate answer region is much easier than finding exact answer span for our model. However, the F1 score is relatively stable, this shows that our model could maintain its strength in identifying the approximate answer span even when the span is long.

Figure 2(a) shows how our model words on different question types. The question types are classified by how the questions are asked, specifically, the question words. According to the performance on the dev set, our models work best on what yearquestions, with when, in what, in whichand how many questions follow as second place. This may because answer to these time or location related questions are usually short and have fixed patterns. Moreover, the in in in what and in which could be a significant word to help locate the answer span. Our model works worst on how did questions. This could be explained since that answer to these type of questions usually dont have a set pattern like a list of nouns, and the answer pointer layer of our model dont scan every word of passage to get



(a) (b)

Figure 3: (a) performance for various answer types; (b) performance for answers of different length

a more accurate estimate on answer span, therefore it is not surprising that the model is not working good in identifying exact span, even though the F1 score seems plausible.

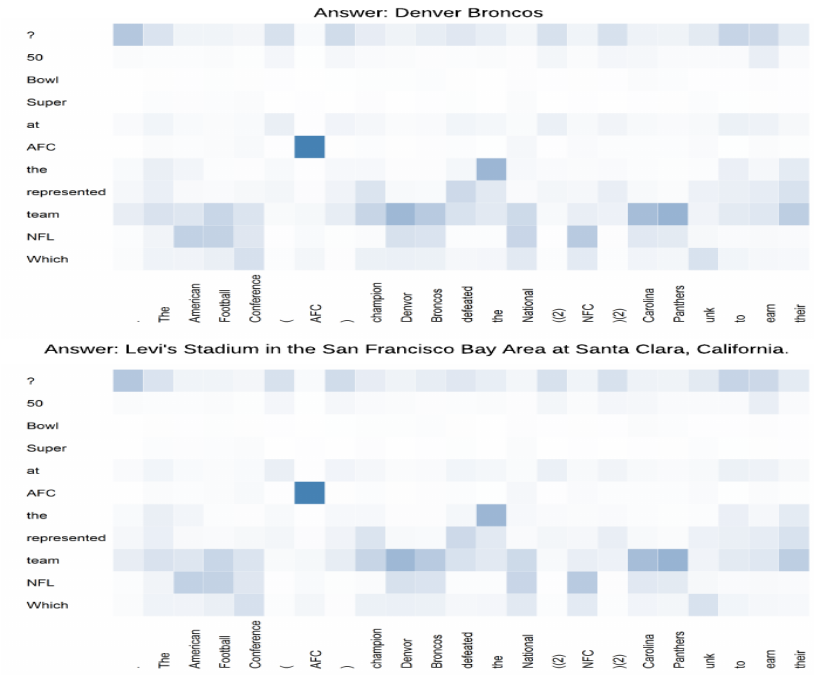


Figure 4: Visualization of the attention weights for two questions

Figure 4 is the heatmap of attention weights between questions and some regions of context. The larger the weight, the darker the color is. It can be observed that some words in context are more related with words in questions which have similar semantic meaning. For example, the word "Denver Bronco" and "Carolina Panthers" are both name of sports team, therefore among all the question

words they have largest attention weights on words “team”. “Levi”, “San Francisco”, “California”, “Bay Area”, and “Santa Clara” are locations, and we can see that they get more attention with “where” and “place” in question. This show our bi attention layer is helpful for the model to locate potential answer span.

4 Model discussion and Future Work

Having walked through the model of Match-LSTM and proposed some improvements, we believe that the remaining major limitation of this system is the decoder. Although Answer Pointer net has proved its success in[2] and[7], we think the decoding procedure is not as intelligent as what human do. In general, when human dealing with this task, we first find several most likely places as candidate answer span, then we go through those parts again and again to refine our answers. However, in Match-LSTM and hence our implementation, the model heavily relies on a one-time success of the Pointer Net, not to mention that once it makes error in predicting the start index, the error will propagate to the end index. We believe that a more reasonable decoder should work in an iterative way: it has some form of evaluating the prediction and can generate feedback to re-predict. A good example of this function is the Highway Maxout Network, where the model is able to pool across multiple variations and iterate through the paragraph many times until converge.

Hence, in the future we would like to add this feature to the Match-LSTM system, so as to refine the prediction of Answer Pointer Net. One idea we would like to try is to have an additional ”Evaluation Net”, which consists of a sigmoid unit that can generate the confidence of the prediction by looking at the hidden representation of the predicted answer span and compared that with the question, then use the confidence score to mask out low-confidence context and make prediction again. However, due to time constraint and resources limitation, we are unable to finish trying our hypothesis.

5 Contribution

Both team members contributed equally to model building, coding, model training and paper writing.

References

[1] Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text.*arXiv preprint arXiv:1606.05250.*

[2] Wang, S., & Jiang, J. (2016). Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905.*

[3] Wang, Z., Mi, H., Hamza, W., & Florian, R. (2016). Multi-Perspective Context Matching for Machine Comprehension. *arXiv preprint arXiv:1612.04211.*

[4] Xiong, C., Zhong, V., & Socher, R. (2016). Dynamic Coattention Networks For Question Answering. *arXiv preprint arXiv:1611.01604.*

[5] Seo, M., Kembhavi, A., Farhadi, A., & Hajishirzi, H. (2016). Bidirectional Attention Flow for Machine Comprehension.*arXiv preprint arXiv:1611.01603.*

[6] Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global Vectors for Word Representation. In*EMNLP* (Vol. 14, pp. 1532-1543).

[7] Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. In *Advances in Neural Information Processing Systems* (pp. 2692-2700).

[8] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.

[9] Mnih, V., Heess, N., & Graves, A. (2014). Recurrent models of visual attention. In *Advances in neural information processing systems* (pp. 2204-2212).

[10] Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., & Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems* (pp. 1693-1701).

[11] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

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

[13] Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 3156-3164).

[14] Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., & Darrell, T. (2015). Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 2625-2634).

[15] Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.