Reading Comprehension on the SQuAD Dataset

FNU Budianto
Codalab username: budi71
`budi71@stanford.edu`
CS 224N Winter 2016-17, Assignment #4

Abstract

Reading comprehension is a challenging task for machine learning, since the system must be able to model complex interactions between the question and the context paragraph. For this task, I reimplemented the Bidirectional Attention Flow model and linked the two BiLSTMs in the contextual embed layer. A single model achieved 74.524% F1 and 64.261% EM on the test set. The ensemble of seven models achieved 77.465% F1 and 68.478% EM. The model achieved competitive rank on the class leaderboard, despite having 4% lower score than the original implementation.

1 Introduction

Although end-to-end deep neural network have gained popularity in the last few years and have been successful in several Natural Language Processing (NLP) tasks such as sequence labeling [1], the task of reading comprehension remains a challenging task for NLP researchers. This is because the system must be able to model complex interactions between the question and the context paragraph and must be able to do several complex skills such as coreference resolution, commonsense reasoning, causal relations, and spatiotemporal relations [2]. Another problem was the lack of dataset, but this has been solved by SQuAD [3], a reading comprehension dataset with more than 100,000 questions.

For this assignment, I have implemented three models: Very Basic Model (VBM), Simple Attention Model (Baseline), and Bidirectional Attention Flow (BiDAF) [4]. I started with the VBM model, which is a simple two layer BiLSTM. Then, I added the seq2seq attention mechanism [5] to it and called it the baseline model. After that, I reimplemented the BiDAF model which is a hierachical multi-stage architecture that achieved a very high rank on the SQuAD public leaderboard. Finally, I built an ensemble of seven BiDAF models with majority voting strategy and this ensemble achieved a competitive rank on the class leaderboard. I also provide a study on the effect of adding multiple components, performance analysis of the three models, and visualization of the attention vector and output probabilities for two example questions (one correct and one incorrect).

2 Related Work

There has been a lot of work on building deep learning systems for the SQuAD Dataset, as can be seen in the leaderboard (`https://rajpurkar.github.io/SQuAD-explorer/`) [6] [7]. The top models utilize some form of attention mechanism, which has been proven useful in improving accuracies in many NLP tasks. It was first introduced in Neural Machine Translation [5], where it generate a translation word based on the whole original sentence states, not just the last state. The idea is then applied to reading comprehension, allowing the model to select a subset of context paragraph and a subset of question that are most relevant. That way, the model can use the most relevant information to give a better answer.

One successful model for this task is the Dynamic Coattention Networks [8]. This model consists of document encoder, question encoder, coattention encoder, and dynamic pointer decoder. The coattention encoder is the attention mechanism for this model. It encodes the interaction between the encoded question and the encoded document. The dynamic pointing decoder uses highway maxout network to compute the probability of the start index and the end index. The process is repeated a few times, using the information of the previous prediction to improve the next prediction. This iterative process allows the model the escape from a local maxima.

Another successful model for reading comprehension is the Bidirectional Attention Flow model. This model introduces a bidirectional attention flow mechanism to obtain query-aware context and context-aware query without early summarization. This is the model that I have implemented that it will be discussed further in Section 4.

3 Dataset and Features

The dataset for this task is the Stanford Question Answering Dataset (SQuAD) [3], a reading comprehension dataset built by crowdworkers on a set of Wikipedia articles. The dataset contains more than 100,000 question-answer pairs on 500+ articles. The data is split into 80% training set, 10% development set, and 10% test set (hidden). 5% of the training set is taken for validation set, which is used in hyperparameter search. The histograms of the question, context, and answer length of the training set can be seen in Figure 1. Over 90% of the questions have short answers (less than 10 words). Using the histogram information, I set the maximum length of context to be 400, maximum length of the question to be 30, and maximum length of the answer to be 10. Setting the limit of context and question helps to decrease the training time, and setting the limit of answer helps to increase the F1 score.

The sentence is tokenized using NLTK toolkit (`nltk.org`) [9]. The resulting word indices (one-hot vectors) are then mapped into distributed word representation using Glove [10]. The Glove version used is the Common Crawl (840B tokens, 2.2M vocab, cased, 300d vectors).

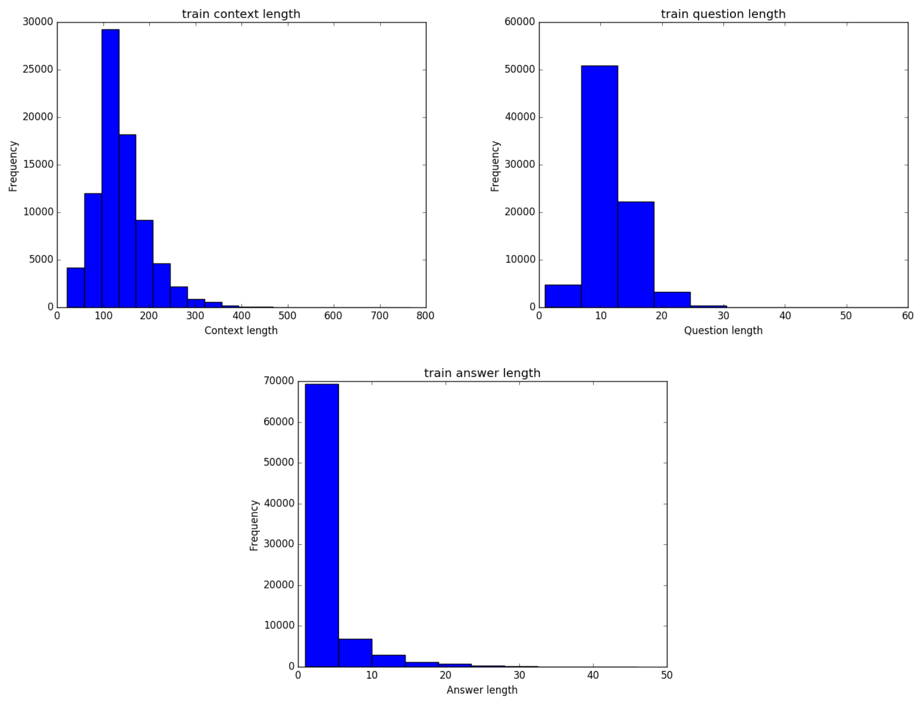


Figure 1: Histogram of context length, question length, and answer length in the training set.

4 Model Architecture

Figure 2 shows the general architecture of the model. First, there is the embedding layer, where the context and question are mapped into the distributed word representation. The output of this layer, the context embedding and the question embedding, are then fed into the answering layer, where it will output the probability of start index and probability of end index, for each of the context word.

Following from the BiDAF model, the embedding layer consists of word embedding layer and character embedding layer. I use the Glove840B300d Common Crawl for the word embedding layer. For the character embedding layer, I use Convolutional Neural Network (CNN) with max pooling [11]. The output from word embedding and character embedding are then concatenated and fed into highway networks [12]. Highway network is a network with gating mechanism to control which data can pass though and which data will be transformed non-linearly. The goal of using character embedding is to help the model when the word is not in the vocabularity, and the goal of highway network is to compute features at different level of granularity.

There are three types of answering layer implemented: Very Basic Model (VBM), Simple Attention Model (Baseline), and Bidirectional Attention Model (BiDAF). All three models use the same embedding layer. After manual hyperparameters searching, the chosen hyperparameters are learning rate = 0.5, optimizer = adadelta [13], learning rate decay = 0.999, dropout keep prob = 0.8, word embedding = fixed glove840B300, char embedding size = 50, highway network layer = 1, max question length = 30, max context length = 400, max answer length = 10, hidden size = 100 (bidaf), 200 (others). The loss function used is the sum of the softmax cross entropy of the start index and softmax cross entropy of the end index.

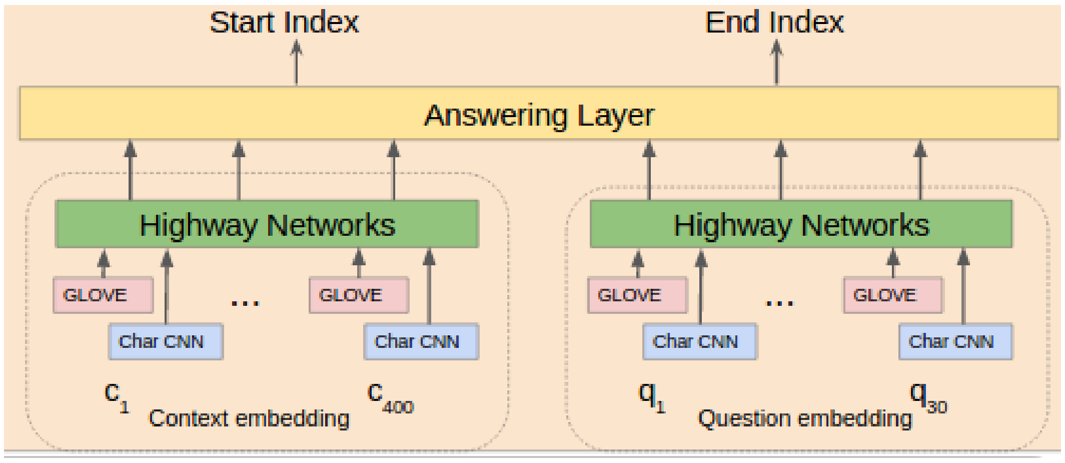


Figure 2: The general model infrastructure.

4.1 Very Basic Model (VBM)

This model runs a BiLSTM [14] over the question embeddings and then runs another BiLSTM over the context embeddings, using the final state of the question BiLSTM as the initial state of the context BiLSTM. It uses a recurrent neural network to process the question and context in order to capture contextual information from the sentence.

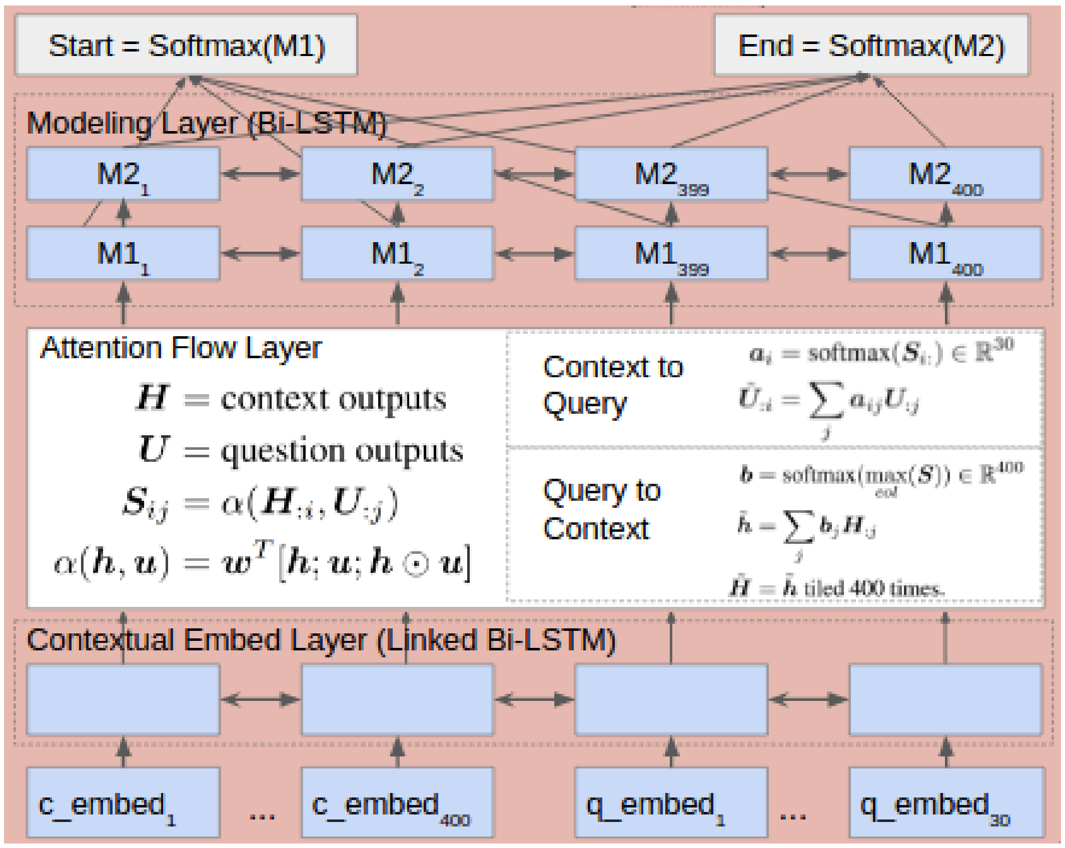
4.2 Simple Attention Model (Baseline)

Instead of using the last state of the question BiLSTM as the initial state of the context BiLSTM, I implement the attention mechanism similar to the one proposed by Bahdanau [5]. I also change the BiLSTM to BiGRU [15] because it is easier to implement the attention mechanism on top of a GRU cell and it has less parameters so the model can train faster. Here is the implemented attention mechanism:

$$\begin{aligned}
\text{c\_out}_i &= \text{GRU}(h_{i-1}, \text{c\_in}_i, b_i) \\
b_i &= \sum_{j=1}^{30} \alpha_{ij} \text{q\_out}_j \\
\alpha_{ij} &= \frac{\exp(e_{ij})}{\sum_{k=1}^{30} \exp(e_{ik})} \\
e_{ij} &= \text{score}(h_{i-1}, \text{q\_out}_j) \\
\text{score}(x, y) &= x^T W y
\end{aligned}$$

When running a GRU over the each word in the context, it selects the relevant subset of question by computing a weighted sum of all question outputs and uses the weighted sum as the second input to the GRU cell. The weight is computed using the above equation.

4.3 Bidirection Attention Flow (BiDAF)



Start = Softmax(M1)
End = Softmax(M2)
Modeling Layer (Bi-LSTM)
$M2_1$
$M2_2$
$M2_{399}$
$M2_{400}$
$M1_1$
$M1_2$
$M1_{399}$
$M1_{400}$
Attention Flow Layer
H = context outputs
U = question outputs
$S_{ij} = \alpha(H_{:i}, U_{:j})$
$\alpha(h, u) = w^T[h; u; h \odot u]$
Context to Query
$a_i = \text{softmax}(S_{i:}) \in \mathbb{R}^{30}$
$\tilde{U}_{:i} = \sum_j a_{ij} U_{:j}$
Query to Context
$b = \text{softmax}(\max_{col}(S)) \in \mathbb{R}^{400}$
$\tilde{h} = \sum_j b_j H_{:j}$
$\tilde{H} = \tilde{h}$ tiled 400 times.
Contextual Embed Layer (Linked Bi-LSTM)
c_embed_1
...
c_embed_{400}
q_embed_1
...
q_embed_{30}

Figure 3: The BiDAF model.

The BiDAF model is depicted in Figure 3. First, there is a contextual embed layer on top of the embedding layer in order to model contextual and temporal relations between the words. The output is $\boldsymbol{H}^{2d \times 400}$ for the context and $\boldsymbol{U}^{2d \times 30}$ for the question (d = hidden size). Next is the attention flow layer, where it first computes the cross-product of all question words and context words ($\boldsymbol{S} \in \mathbb{R}^{400 \times 30}$). The matrix $\boldsymbol{S}$ is then used to compute Context to Query attention and Query To Context attention. In Context To Query, it computes an attention vector of the question words for each of the context word. In the Query To Context, it computes one attention vector of the context words for all of the question words. The output from contextual embed layer and attention flow layer are

Table 1: Model performance on the test set vs state of the art.

Vs. State of The Art (Test Set)	F1 (%)	EM (%)
Human Performance	91.221	82.304
r-net (ensemble)	84.006	75.034
Orig. BiDAF (ensemble)	81.525	73.744
My BiDAF (ensemble, voting)	**77.465**	**68.478**
Orig. BiDAF (single model)	77.323	67.947
Dynamic Coattention Networks (single model)	75.896	66.233
My BiDAF	**74.524**	**64.261**
Match-LSTM with Bi-Ans-Ptr (Boundary)	73.743	64.744
Logistic Regression Baseline	51.0	40.4

combined to create $\boldsymbol{G} = \beta(\boldsymbol{H}, \tilde{\boldsymbol{U}}, \tilde{\boldsymbol{H}}) = [\boldsymbol{H}; \boldsymbol{U}, \boldsymbol{H} \odot \tilde{\boldsymbol{U}}; \boldsymbol{H} \odot \tilde{\boldsymbol{H}}]$. $\boldsymbol{G}$ is then fed into the modeling layer, where there are two layers of BiLSTM ($\boldsymbol{M}1$ and $\boldsymbol{M}2$). Next, the probability of start index over the context words is computed by this formula: $\boldsymbol{p}_1 = \text{softmax}(\boldsymbol{w}_1^T[\boldsymbol{G}; \boldsymbol{M}1])$. Similarly, the probability of end index is $\boldsymbol{p}_2 = \text{softmax}(\boldsymbol{w}_2^T[\boldsymbol{G}; \boldsymbol{M}2])$ Finally, the answer is chosen by finding the span (i, j) where $\boldsymbol{p}_1[i] * \boldsymbol{p}_2[j]$ is the maximum, $i \leq j$ and $j - i + 1 \leq$ max_answer_length.

My modification over the original BiDAF is I linked the two BiLSTM in the contextual embed layer (they are separated in the original). The reason is that the contextual embed layer of context should depend on the question.

5 Experiment Results and Discussion

The system is implemented using Tensorflow (`tensorflow.org`) and it follows the code structure of Assignment #3. I reused the ProgBar and minibatches utilities from Assignment #3. For the 1D convolution and highway networks implementation, I adapted the code from the BiDAF github (`https://github.com/allenai/bi-att-flow/blob/master/my/tensorflow/nn.py`) and changed the code to support dropout keep probability from a tensor and replace the old linear functions with the new tf.fully_connected functions. To combine the answer tokens into a sentence, I utilize the untokenize function from `https://github.com/commonsense/metanl/blob/master/metanl/token_utils.py` and added another step to remove whitespaces between double quotes and the phrase (e.g " foo bar " → "foo bar"). On Microsoft Azure, one epoch of VBM takes about 600 seconds, one epoch of baseline model takes about 900 seconds, and one epoch of BiDAF model takes about 1700 seconds. I have also created a simple demo and it is accessible at `http://squad.budianto.id`. The model used in the demo is the single BiDAF model.

F1 score and Exact Match (EM) score are used for evaluating the model. F1 score is the harmonic mean of precision and recall (the higher the better). For each question, precision is calculated as the number of correct words divided by the number of words in the predicted answer. Recall is calculated as the number of correct words divided by the number of words in the ground truth answer. The F1 score is computed per question and then averaged across all questions. EM score (the higher the better) is the number of questions that are answered in exact same words as the ground truth divided by the total number of questions. A single model of my BiDAF implementation achieved 74.524% F1 and 64.261% EM on the test set, and the ensemble of seven BiDAF models with majority voting strategy achieved 77.465% F1 and 68.478% EM. The comparison of these results against the current state of the art can be seen in Table 1. My implementation of BiDAF has 4% lower F1 score than the original BiDAF and this might be because they did larger hyperparameter search or there are other details that are not mentioned in the paper.

5.1 Analysis on the effect of different components in the model

I tried a lot of different component and parameter combinations and the result can be seen in Table 2. From the result, we can see that attention mechanism is the most important important, even with just a simple attention the F1 score improved by more than 30%. Modeling layer of the BiDAF is

Table 2: Experiment results on the dev set.

Own Model (Dev Set)	**F1 (%)**	**EM (%)**
My BiDAF (ensemble)	77.712	69.026
My BiDAF (with linked context embed layer)	74.952	65.563
Reimplementation of original BiDAF	74.04	63.916
BiDAF without char embedding	73.911	63.888
BiDAF with fixed Glove6B100d	72.904	62.242
BiDAF without modeling layer	69.863	58.6
Simple Attention Model (Baseline)	68.224	57.565
VBM	37.981	29.80
VBM without char embedding	37.425	29.205
VBM with fixed Glove6B100d and w/o char embedding	34.478	26.348

Table 3: My BiDAF (single model) performance on different question types in the dev set.

Question Type (Dev Set)	**F1 (%)**	**EM (%)**	**#questions**	**Avg. Answer length**
When	82.997	77.662	864	2.24
Who	76.068	69.935	1377	2.62
How	74.654	64.651	1389	2.5
What	73.377	63.181	6073	3.02
Where	73.025	62.008	508	2.89
Why	60.708	36.076	158	6.87

also important, without it, the F1 score decreased by 5%. This means that contextual interactions between context representations conditioned on the question is important. Using Glove Common Crawl 840B tokens instead of Glove Wikipedia 6B tokens improved the F1 score by 2% and that is because there are fewer words that are not in the vocabulary. One interesting thing that I found is that fixing the Glove vectors results in 1% higher F1 score than training them, and I guess that is because the Glove vectors are already well trained (on 840B tokens) and training them again will cause overfitting. Character embeddings improved F1 score by 1% which does not seem much and it is probably because there are not many words that are not in the vocabulary and the Glove vectors are already representative enough. I have also tried to increase the character embedding output dimension to 100 and the number of highway network layers to two, but that didn't seem to have any noticable impact on the F1 score. Finally, linking the two BiLSTM (use the question's final state as the context's initial state) in the BiDAF's contextual embed layer improved the F1 score by 1%.

In the answer generation, using proper untokenize function that removes whitespace between a word and the punctuation instead of simple token joining improved the F1 score by 0.5% and EM score by 1.5%. That is because there are quite a number of answers in the dev set that contain some punctuations, and simple joining will not give EM score of 1.0 for these questions. For the ensemble strategy, I have tried three different strategies: sum_preds (summing the raw probabilities), max_prob (pick the answer with the highest probability $\boldsymbol{p}_1[i] * \boldsymbol{p}_2[j]$, and max_vote (majority voting and fallback to max_prob for tie breaker). The highest increase in F1 (3%) is gotten by using max_vote. Max_prob only increased the F1 by 1% and sum_preds did not increase F1 at all. I am also limiting the answer length to a maximum of 10 words and that increased the F1 by another 1%. The limit helps because more than 90% of the questions have short answers. Using exponential moving average of parameters (LSTM weights, biases, etc.) instead of the raw parameters during evaluation improved F1 score by 2-4%.

5.2 Analysis on different question types and answer lengths

The model's performance on different question types can be seen in Table 3. The BiDAF model performs the best on when questions and performs the worst on why" question. That is because "when" questions have short answer length (2.24 on average), whereas "why" questions have higher answer length (6.87 on average). The model's performance on different golden answer length can be seen in Table 4. We can see that the EM score for the last two rows is zero and that is because

Table 4: My BiDAF (single model) performance on different golden answer length in the dev set.

Ground Truth Answer Length (Dev Set)	**F1 (%)**	**EM (%)**	**#questions**
1-5	76.770	69.089	9453
6-10	66.502	46.856	843
11-15	38.286	2.105	190
16-20	41.830	0	55
20+	30.855	0	29

I limit the answer length to a maximum of 10 words. I still get a few EM score on the 11-15 range and that is because the evaluation script will normalize the ground truth answer when comparing the answers.

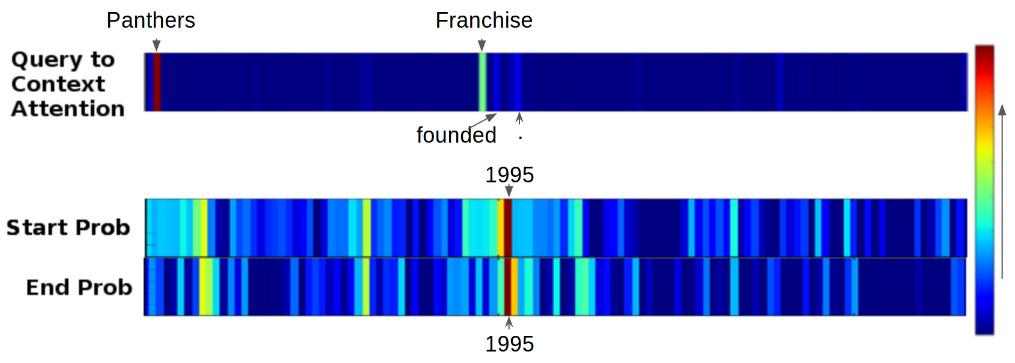


Figure 4: Visualization for question: ”What year was the Carolina Panthers franchise founded?”.

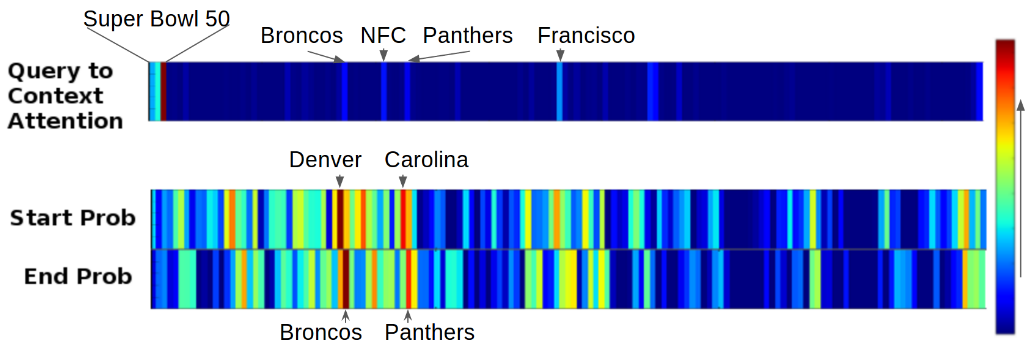


Figure 5: Visualization for question: ”Which NFL team represented the NFC at Super Bowl 50?” Actual = Carolina Panthers, predicted = Denver Broncos.

Figure 4 and 5 visualize the query to context attention and the output probabilities for two different questions. Figure 4 is for the question that the model answered correctly. We can see in the attention vector that it managed to emphasize the question words in the context (Panthers and Franchise) and the model is able to place high confidence of the answer (1995), indicated by only one red column in the probability vector. Figure 5 is for the question that the model answered incorrectly. We can see the ”Super Bowl 50” is emphasized in the attention vector, but somehow ”Francisco” is also emphasized, even though it is not in the question. The resulting output probabilities for this question contain multiple red and orange columns, causing the model to not have a high confidence on the answer.

6 Conclusion and Future Work

For this assignment, I have reimplemented Bidirectional Attention Flow model and made minor modification in the contextual embed layer by using the question's final state as the context's initial state. This reimplementation achieved a competitive result on the test set, but it is still behind the original paper's performance by about 4%. From the component analysis, attention mechanism is the most important component for this task, since using just a simple attention already increased the F1 score by more than 30%. The visualization shows that the model is able to pick relevant context words according to the question and use that information to make a confident prediction on the answer start and end index.

One possible future work is to replace BiDAF modeling layer with DCN Dynamic Pointer Decoder. The idea is that Dynamic Pointer Decoder is able to recover from local maxima, which should help improving the performance of BiDAF. There is also still a gap between my BiDAF implementation and the original BiDAF implementation and that can still be improved by doing further hyperparameter tuning and bug fixing. Lastly, the web demo can be improved by adding more features, such as attention vector visualization.

Acknowledgments

I would like to thank Chris Manning, Richard Socher, and all the TAs for a great class experience. I would also like to thank Microsoft for providing GPU VM on Azure to train the models.

References

[1] X. Ma and E. Hovy, "End-to-end sequence labeling via bi-directional lstm-cnns-crf," *arXiv preprint arXiv:1603.01354*, 2016.

[2] S. Sugawara and A. Aizawa, "An analysis of prerequisite skills for reading comprehension," *EMNLP 2016*, p. 1, 2016.

[3] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ Questions for Machine Comprehension of Text," *ArXiv e-prints*, Jun. 2016.

[4] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi, "Bidirectional Attention Flow for Machine Comprehension," *ArXiv e-prints*, Nov. 2016.

[5] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *ArXiv e-prints*, Sep. 2014.

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

[7] 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.

[8] C. Xiong, V. Zhong, and R. Socher, "Dynamic Coattention Networks For Question Answering," *ArXiv e-prints*, Nov. 2016.

[9] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ser. ETMTNLP '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 63–70. [Online]. Available: http://dx.doi.org/10.3115/1118108.1118117

[10] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[11] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[12] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," *arXiv preprint arXiv:1505.00387*, 2015.

[13] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[15] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *ArXiv e-prints*, Jun. 2014.