An Exploration of Question-Answering Modules

Margaret Guo
Department of Biomedical Informatics
Stanford University
mguo123@stanford.edu

Wen Torng
Department of Bioengineering
Stanford University
wtorng@stanford.edu

Abstract

In this project, we experimented with various deep learning-based models for question-answering. Areas we explored include: extraction of features such as exact match, part-of-speech (POS) word tokens, and character-based embeddings; attention modules including Bidirectional Attention Flow (BiDAF), Co-Attention, and self-attention; and different output modules that condition end predictions on start predictions. Our final model uses the BiDAF attention along with a LSTM-based start-conditioning output module, and achieved a test F1 score of 0.7061 and a EM score of 0.6020.

1 Introduction and Background

The task of machine compression (MC) is of particular importance within the national language processing domain. MC problems are a form of question-answering, in which a machine attempts to answer a question about a given contextual paragraph. Both the predicted and true (or labeled) answers are spans within the context. Evaluation of these systems is performed by evaluating the exact match (EM) and F1 scores on a separate test set. Recent advances in the neural-net based QA systems include character-level embeddings, various forms of feature engineering, different advanced attention mechanisms, and output modules that condition end predictions on start predictions.

Character-level embeddings provide insight into internal structure of words and can be used to learn representations of unknown words [1]. Additionally, prior models, such as the DrQA model[2], have obtained high performance using features such as exact match (a word-to-word matching between context and question), word-based features (including its stem word, entity type from Named Entity Recognition [2], and part of speech (POS) tags), and aligned question embedding.

Attention level improvements allow the model to capture more complex interactions between the context and question. Bi-directional Attention Flow (BiDAF) [1] allows for a two-way flow of attention between question to context (Q2C) and context to question (C2Q). The Dynamic Co-Attention model [3], is similar to BiDAF in that it has a bi-directional attention flow; however, there is an secondary attention layer that allows the C2Q to attend to the Q2C output layer. Finally, self-attention has been used as a supplementary attention layer to allow the context additionally to attend to itself [8].

For the output module, the most basic models use two independent fully connected layers to generate the start and end position scores. Because of the natural dependency of the end position on the start, recent research has been focusing on novel architectures that condition end predictions on start predictions [7].

In this paper, we use the SQuAD dataset [4] as the source of our context/question/answer examples. We then made various changes to the input, encoding, attention, and output models. We report our F1/EM scores of these experiments as well as perform error and attention level analysis on our best model to inform future work on this challenge.

2 Approach

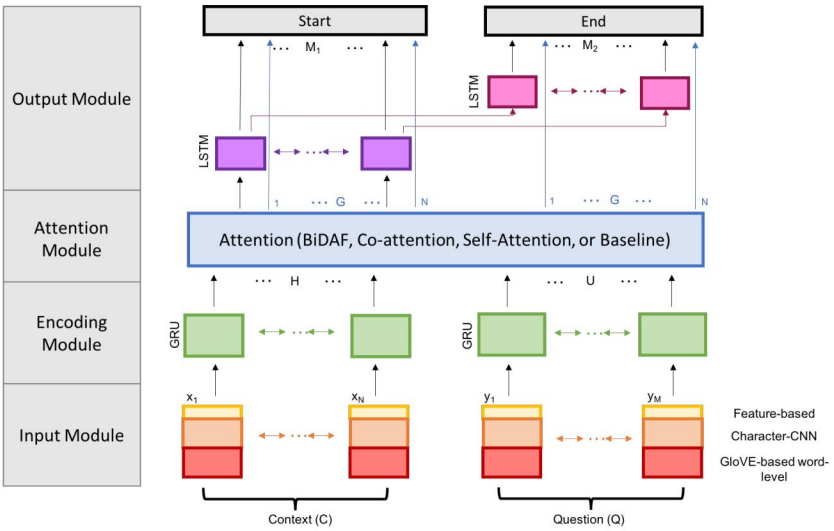

Figure 1: Schematic representation of our approach

Our model includes 4 modules that we outline and then describe in detail below:

1. **Input Module**: Converts word, character, and feature-based inputs into vector space
2. **Encoding Module**: Takes input features to learn temporal dependencies among input words
3. **Attention Module**: Integrates info from hidden states of the question and context words.
4. **Output Module**: Takes attention output to predict start and end positions within the context.

2.1 Input Module

The input module is split into word-level, character-level, and feature-based embeddings.

Word-level input module: Word-level embeddings are created by mapping individual word tokens into a lower dimensional space (d_w) based on pre-defined GloVE embeddings[5]. For a given number of words in an input L_S, the output is of size L_S x d_w.

Character-level input module: Character-level embeddings are created by mapping the characters of individual words into a lower dimensional space (d_c). These embeddings are created by initially extracting integer tokens that represent each character of every word. These character tokens are then mapped into a trainable embedding layer (d_c), such that a word of length L_W with character tokens ($c_1 \dots c_{L_W}$), has embeddings ($e_1 \dots e_{L_W}$), where e_i is a vector of size d_c for the ith character in the word. For a given input sentence of length L_S, the output matrix is of size L_S x d_c.

Once these embeddings have been created, we feed each character vector into a 1 dimensional convolutional neural network (CNN) to create a hidden representation of the characters ($h_1 \dots h_{L_W}$), such that the character representation of h_i for the ith character is computed from convolving character embeddings ($e_{\lfloor i-\frac{k}{2} \rfloor}, \dots e_i, \dots, e_{\lceil i+\frac{k}{2} \rceil}$ with a filter of size k to generate a vector of size f. The output of this layer is a matrix of size L_S x L_W x f. This result is max pooled over each individual word to create a L_S x f matrix for each input. Character level embeddings were then concatenated with the word-level embeddings to create a L_S x (d_w+f) size matrix.

Feature-based embeddings: We used exact match and part of speech tags as our additional features. For exact match, we created a Boolean vector that indicated whether a particular word in the question was found in the context, and vice versa. For part of speech tags, we generated the tags using the coreNLP software[6] for each word. Both feature vectors were concatenated to the above embeddings of size L_S x h, where h is the total hidden size of the embeddings. The length of the

input, L_S, is fixed for the context and question to N and M, respectively, based on analysis of input length distributions (see Section 3.1). We label the final embeddings for the context $x_1 \dots x_N$ and for the question $y_1 \dots y_M$.

2.2 Contextual Embedding Module

The contextual embedding module takes in the input features to learn temporal dependencies among the input words. We started with a single layer bi-directional GRU network as in the baseline model. We then experimented with adding additional GRU layers to this module. We concatenate the forward and backward hidden states to obtain the context hidden states and the question hidden states respectively. From this module, we obtain $H \in \mathbb{R}[N\text{x}2h]$ from the context word vectors $[x_1, x_2, \dots x_N]$, and $U \in \mathbb{R}[Mx2h]$from query word vectors $[y_1, y_2, \dots y_M]$.

2.3 Attention Module

In this project, we implemented different attention modules as follow:

Bidirectional Attention Flow: The BiDAF attention module uses two-way attentions, both from context-to-question (C2Q) and from question-to-context (Q2C). The context-to-question (C2Q) attention computes which question words are most relevant to each context word, which is conceptually similar to the baseline attention. The question-to-context (Q2C) attention signifies which context words have the closest similarity to ones of the query, which could essentially provide us information similar to the "exact match" input features. The final output of the BiDAF module is $G \in \mathbb{R}[N, 8h]$.

Co-Attention: The Co-Attention module also utilizes a two-way attention between the context and the question. It calculates an affinity matrix L where each row contains the affinity scores of each context word to the question words, and each column contains the affinity scores of each question word to the context words. The affinity scores are then used to calculate the α (C2Q) and β (Q2C) attention vectors. The Co-attention module then uses the computed C2Q attention softmax distribution to attend over the β attention vectors to get a second-layer attention S. To obtain the final attention output, we concatenate α attention vectors with the S attention vectors. We then fed the concatenated vectors through a single layer Bidirectional LSTM to obtain the final output G. The final output of the Co-Attention module is $G \in \mathbb{R}[N, 2h]$.

BiDAF with Inter-Context Self Attention: Although the Contextual Embedding Module should have encoded dependencies among words within the context, it is known that LSTM and GRU cells can still suffer from "memory loss" when the sentence is long. To enable information of early words to be accessible to the words far away in the sentence, in addition to BiDAF, we added a Self-Attention module to attend context hidden embedding to themselves. The equations of the Inter-Context Self Attention module are as follows:

$$C_{proj_1} = HW_1,\ C_{proj_2} = HW_2$$

$$Sim = C_{proj_1} C_{proj_2}^T$$

$$a_{self}^i = softmax(Sim_{i:})$$

From which we obtain the attention scores A, where $A_{i:}$ is a_{self}^i. We can then obtain the self-attention output:

$$Out_{self} = AH,$$

$$\text{where }, H \in \mathbb{R}[N\text{x}2h],\ W_1 \in \mathbb{R}[2h\text{x}2h],\ W_2 \in \mathbb{R}[2h\text{x}2h]$$

$$A \in \mathbb{R}[N\text{x}N],\ Out_{self} \in \mathbb{R}[N\text{x}2h]$$

Finally, we concatenated the Bidirectional attention flow output with the self-attention output and obtain the BiDAF with Inter-Context Self Attention $G \in R[N, 10h]$

2.4 Output Module

The output module integrates output from the attention layer to predict the start and end position of the answer. The baseline model predicts the start location and the end location independently, given the attention output. Since the position of end position should be dependent on the start, we experimented with two different strategies of conditioning end predictions on start predictions.

Predicting end positions conditioned on start attention : In this output module, we first calculate the start scores and probability from attention output G using a fully connected layer and down-projection layer as in the baseline output layer. This start probability distribution is then used to obtain a summary "attention vector" of the information important for making the start prediction. We then fed the attention output G into a GRU network, using this attention vector as the initial hidden state, to obtain a second layer hidden representation M_2. M_2 is then fed into a fully connected layer and down-projection layer to generate the end predictions. The equations are summarized as below:

$$Q_1 = Relu(GW_1 + b_1)$$

$$p_{start} = softmax(Q_1 w_{start})$$

$$\alpha_{vec} = p_{start} G$$

$$M_2 = SingleGRU(G, hidden\_size = 2h, initial_h = \alpha_{vec})$$

$$Q_2 = Relu(M_2 W_2 + b_2)$$

$$p_{end} = softmax(Q_2 w_{end})$$

Where $G \in \mathbb{R}[NxD]$, where D is the dimension of the attention output, $W_1 \in \mathbb{R}[Dx2h]$, $b_1 \in \mathbb{R}[2h]$, $w_{start} \in \mathbb{R}[2h\text{x}1]$, $W_2 \in \mathbb{R}[2hx2h]$, $b_2 \in \mathbb{R}[2h]$, $w_{end} \in \mathbb{R}[2h\text{x}1]$.

BiDAF Output Layer: The BiDAF paper uses two bidirectional LSTMs ($LSTM_{start}$ and $LSTM_{end}$, respectively) to compute the hidden representations for predicting the start and end positions. Although this module did not model the end prediction on the start probability explicitly as in the previous module, since it uses the output of $LSTM_{start}$ as the input to $LSTM_{end}$, this module implicitly conditions the end prediction on the start information. Following the Modeling and Output layer described in the BiDAF paper [1], we implemented the output model described by the following equations:

$$M_1 = BiLSTM_{start}(G, hidden\_size = h)$$

$$M_2 = BiLSTM_{end}(M_1, hidden\_size = h)$$

$$S_{start} = [G; M_1] w_{start},\ p_{start} = softmax(S_{start})$$

$$S_{end} = [G; M_2] w_{end},\ p_{end} = softmax(S_{end})$$

Where $G \in \mathbb{R}[NxD]$, where D is the dimension of the attention output $w_{start} \in \mathbb{R}[(D + 2h)\text{x}1]$, $w_{end} \in \mathbb{R}[(D + 2h)\text{x}1]$.

3 Results

3.1 Datasets/ Use of Existing Software

We used the Stanford Question Answering (SQuAD) dataset [4] with over 100,000+ question-answer pairs on 500+ Wikepedia articles to train this Machine Comprehension model. The baseline code repository was provided by CS224N Winter 2018 class. The Stanford coreNLP module [6] was used to generate token based features such as part of speech.

To determine the fixed context (N) and question length (M) to be inputted into our model, we determined the distribution of the number of words in each context and question example. We determined a cutoff of 400 for the context length and 30 for the question length to provide the most amount of information while saving memory. For the character-level embedding, we used a character per word cutoff of 15 for both the context and question (see Supplement for Figure 4).

3.2 Experiments

To determine the optimal machine comprehension model, we experimented with approaches to the four modules outlined above. For the input module, we experimented with adding character level and/or feature-based vectors to our baseline GLoVE based word embeddings (dimension 100). For the contextual module, we experimented with adding additional RNN layers, which is categorized under hyper parameter tuning. For the attention module, we experimented with various forms of attention including, BiDAF, Co-Attention, and self-attention compared to baseline. For the output module we experimented with various means of conditioning end prediction on the start prediction. Finally, we determined the best hyper parameters (dropout, learning rate) for our best model.

The following parameters were used as our defaults for the experiment: learning rate $= 0.001$, context length $= 400$, question length $= 30$, and dropout $= 0.15$. Batch size was chosen to be either 100 or 64 pending on how much memory the NV12 could handle. We trained all models until they reached a steady state dev F1 score. We report the dev F1 and EM scores for evaluation.

3.2.1 Experiments in Input Module

Table 1: Experiments in Input Module, *=includes BiDAF output module

Experiments	**F1**	**EM**
Baseline	0.4071	0.2934
Baseline + character CNN	0.4040	0.2939
Baseline + POS tags	0.3641	0.2614
Baseline + exact match	0.3961	0.2841
Baseline+ exact Match + char CNN	0.4003	0.2961
Baseline + exact match + POS tags	0.5792	0.4425
BiDAF*	**0.6493**	**0.4990**
BiDAF* + exact match	0.6527	0.5020
BiDAF* + character CNN	**0.6551**	**0.4990**
BiDAF* + POS tags	0.6390	0.4895
BiDAF* + exact match + POS tags	0.6327	0.4885

To determine the usefulness of various features, we added these additional feature layers (character CNN, exact match word-based tags, and part of speech (POS) word-based tags), to the baseline model in addition to word-level embeddings. We found very little difference between model performance when adding in additional embedding layers individually to baseline. Interestingly, the addition of the two feature vectors POS tags and exact match tags substantially increased performance (0.5792) over baseline (0.4071).

For character level input features, we initially experimented with various versions of encoding the character level information for each word, including using one-hot encoding of character tokens, eliminating the trainable character embedding layer, and creating a dense representation of the character tokens (d_c= 1). We also experimented with the number of CNN layers and the width of the CNN window. None of these changes yielded significant improvements to the model.

We also report similar results when we add these features to our best attention module (described in a later experiment). However, when we add both exact match and POS tags to the BiDAF model, there is no improvement noted. Because of the relatively nonexistent performance increase on the dev set we did not include these additional features in our final submitted model.

3.2.2 Attention Mechanisms

To benchmark the performances between different attention mechanisms, we replaced the basic attention module in the baseline model with Co-Attention, Bidirectional Attention Flow (BiDAF), and BiDAF with self-attention, and compare their performances to the baseline model. Because in the original BiDAF paper, a BiDAF output module (described in section 2.4) was used along with the BiDAF attention module, we also experimented the BiDAF attention and Co-Attention with the BiDAF output module. Finally, we examine if adding inter-context self attention, described in section 2.3, can help improve the performances.

The first three entries in Table 2 summarize the performances of basic attention, Co-Attention and BiDAF attention along with the baseline output module. As expected, both Co-Attention and BiDAF attention modules achieved better dev F1 and EM scores compared to the basic attention module. However, the Co-Attention module achieved significantly better results (dev F1: 0.6097, EM: 0.4671) compared to the BiDAF attention module (dev F1: 0.4682, EM: 0.3445) when using the basic output module.

Interestingly, when combining with the BiDAF output layer, the BiDAF attention + BiDAF output layer model achieved a significant improvement (an 0.18 increase in dev F1 score) over the BiDAF + basic output model. However, the same level of improvement was not observed for the Co-Attention + BiDAF output layer. Adding inter-context self-attention to the BiDAF attention + BiDAF output model also did not seem to provide much improvement. Among all the six experiments we performed with attention mechanisms, the best performing model is the BiDAF attention + BiDAF output layer model. This is the final model submitted to the test leaderboard. The final official test F1 and EM scores are 0.7061 and 0.6020, respectively.

Table 2: Attention Mechanisms

Experiments	**F1**	**EM**
Basic Attention + Baseline output layer	0.4071	0.2934
Co-Attention + Baseline output layer	0.6097	0.4671
BiDAF Attention + Baseline output layer	0.4682	0.3445
BiDAF + BiDAF output layer	**0.6493**	**0.4990**
Co-Attention + BiDAF output layer	0.6171	0.4599
BiDAF + Self attention + BiDAF output layer	0.6266	0.4775

3.2.3 Different outputs

In this experiment, we benchmarked performances of different output modules compared to the basic output module used in the baseline model. We first replaced the basic output module in the baseline model with the Start Attention Conditioning output module described in section 2.4, and compared its performance against the baseline model. In addition, as described in the previous section, we compared performances of the BiDAF attention module trained with different output modules.

As can be seen in Table 3, models using both Start Attention Conditioning output module and the BiDAF output module achieved better performances over their counterparts that use basic output module. The Baseline + Start Attention Conditioning model achieved slight improvement over the Baseline with basic output model while the BiDAF + BiDAF Output layer achieved significant improvement over the BiDAF + basic output model. This result suggests that conditioning end predictions on start prediction information is critical to decide the optimal answer boundary.

Table 3: Output Mechanisms

Experiments	**F1**	**EM**
Baseline + Baseline Output layer	0.4071	0.2934
Baseline + Start Attention Conditioning	**0.4326**	**0.3041**
BiDAF+ Baseline Output layer	0.4682	0.3445
BiDAF + BiDAF output layer	**0.6493**	**0.4990**

3.2.4 Hyperparameter tuning

After we obtained our best model, the BiDAF model with BiDAF output module, we tried tuning our hyperparameters, but we found the default values to be the best. We explored changing the number of layers in the RNN encoder and changing the value of dropout. Note we did not perform a thorough investigation on the effect of learning rate, because minor changes in learning rate (from 0.001 to 0.005) resulted in models that did not converge (increased loss).

For dropout we tried three different dropout values 0.25, 0.15 (default), and 0.05. We noted lower dropout was helpful for speeding up training times. However, no significant change in the final dev F1/EM scores (Table 4) was observed . For our submitted model, we used a dropout value of 0.15.

Table 4: Hyperparameter Tuning

Experiments	F1	EM
BiDAF + BiDAF output layer + 1 additional GRU contextual embed layer	0.6368	0.4911
bidaf + bidaf output layer + dropout 0.05	0.6403	0.4895
bidaf + bidaf output layer + dropout 0.15 (default)	**0.6493**	**0.4990**
bidaf + bidaf output layer + dropout 0.25	0.6376	0.4867

3.3 Analyses

3.3.1 Error analysis

Here we analyze the performance of our best model, especially noting where the model does poorly and how we could potentially improve this model in future iterations.

Example #1: Improvement over baseline
(excerpt of) Context: . . . fire and explosion hazards exist when concentrated oxidants and fuels are brought into close proximity ; an ignition event , such as heat or a spark , is needed to trigger combustion . oxygen is the **oxidant** , not the fuel , but nevertheless the source of most of the chemical energy released in combustion . . . .
Question: rather than the fuel , what is oxygen to a fire ?
TRUE ANSWER: oxidant
Predicted Baseline Answer: oxygen
Predicted BiDAF + BiDAF output layer Answer: oxidant
In this case, the basic attention module using in baseline was able to draw attention from the context to query (C2Q) direction over the various question states, however, the additional Q2C directional attention allows us to deduce that we are not looking to repeat the word ”oxygen” but are looking form a word analogy that describes the association between ”fire” and oxygen.” In basic attention, we fail to capture this relationship and thus fail to detect the correct word; while using BiDAF attention, we obtain the correct answer.

Example #2: Lack of additional information
Excerpt of context: the bulk of huguenot émigrés relocated to protestant european nations such as england , wales , scotland , denmark , sweden , switzerland , the dutch republic they also spread beyond europe to several of the english colonies of north america , and **quebec** , where they were accepted and allowed to worship freely .
Question : what area in modern-day canada received huguenot immigrants ?
True Answer: quebec
BiDAF Predicted Answer: england , wales , scotland , denmark , sweden , switzerland , the dutch republic
This example shows the problems that arise when using a single contextual paragraph to answer a question, since not all the concept is captured within a paragraph. In this instance, the machine would have had to learn the relationship between Quebec and Canada. Which would have needed to be reflected in the original GLoVE embedding vectors. Given that we were unable to detect a relationship between Canada and Quebec, we might need additional features, or contextual clues to be able to draw attention to the appropriate noun. Ways to address this issue include pretraining the weights of the contextual embedding layer to include information on all available contexts so that we can use a larger dataset to learn semantic meaning.

Example #3: Lack of understanding of syntax
Excerpt of Context: . . . water on the eastern side flowed toward the atlantic , while to the west water flowed toward the pacific across the amazonas basin . . .
Question: where did water to the west of the amazon drainage basin flow towards ?
True Answer: the pacific
BiDAF Predicted Answer: the pacific across the amazonas basin
In this case, while we did get the correct answer ”the pacific” within our predicted context span. We were unable to understand that the syntax of the answer span was not a true prepositional phrase. The ”across the amazonas basin” is not referring to the pacific, but rather is referring to the movement of water. This is a complex dependency relationship that is not captured by simple attention. A potential solution to address this is to create vector representations of predicted dependency parsing results and then feed this information as an additional feature for the context.

3.3.2 Attention Visualization

For the Co-Attention model, we visualize the heat-map of the C2Q attention matrix, which are the softmax distributions calculated from the affinity matrix L. Below, we show two examples and their respective C2Q attention visualization. Similar effects are seen in Q2C matrices (not shown).

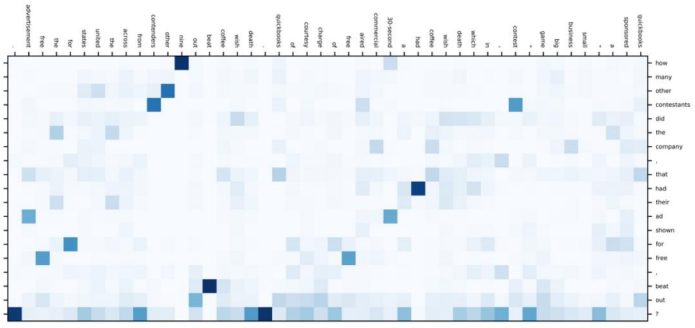

Figure 2: C2Q attention matrix for example 1.

Example 1 Question: "how many other contestants did the company, that had their ad shown for free, beat out?"

From the C2Q attention matrix shown in Figure 2 , we see that for the question word 'how', the co-attention mechanism focus strongly on the context word 'nine', and for the immediately neighboring question words 'many' 'contestants', the attention found matching words in the context neighboring to the word 'nine'. The model successfully uses the number information and the exact matching words near the number to answer the question correctly.

Example 2 Question: "what one word did the nfl commissioner use to describe what super bowl 50 was intended to be?"

From the C2Q attention matrix shown in Figure 3 , we see that for the question word 'what', the co-attention mechanism focus strongly on the context word 'spectacular'. The attention also assigned high score for the exact match words 'nfl', 'commissioner', 'super', 'bowl', The '?' is attending to the other punctuation marks. Interestingly, the context words " " " and " " " before and after answer "spectacular" both strongly attend to the question word "word". It likely captures the semantics that the word in between the quotation marks is emphasized and is more likely to be an answer. The model successfully uses this information to answer the question correctly.

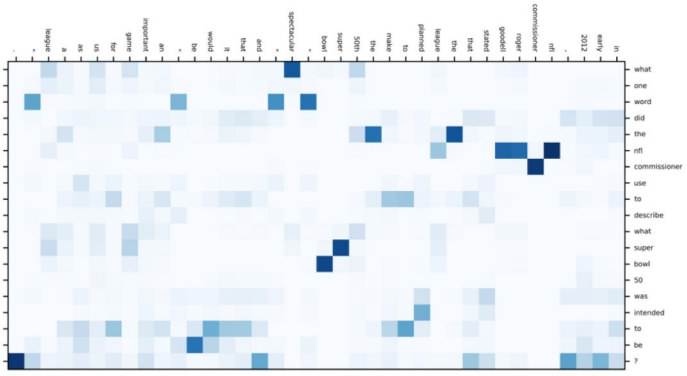

Figure 3: C2Q attention matrix for example 2.

4 Conclusions

In conclusion, we explored various means to improve models for question-answering. Our best model was similar to the BiDAF model with end conditioning on the start prediction. We noted that more complex attention mechanisms that allow for Q2C and C2Q direction attention markedly improved performance. For the output module, we noted that conditioning the end output prediction on the start prediction also improved performance. While we did achieve increases in performance using the word-based feature tokens including exact match and part-of-speech tags, these increases were not evident when combined with the BiDAF model. Future directions include exploring additional feature representations including dependency-based models and aligned question embedding features, and pre-training the contextual embedding layer using semi-supervised methods.

Acknowledgments

We want to thank the TAs for CS224N for all their advice for this project.

References

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

[2] Danqi Chen, Adam Fisch, Jason Weston, & Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint* arXiv:1704.00051, 2017.

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

[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, & Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

[5] Pennington, Jeffrey, Richard Socher, and Christopher Manning. "Glove: Global vectors for word representation." *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP).* 2014.

[6] Manning, Christopher, et al. "The Stanford CoreNLP natural language processing toolkit." *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations.* 2014.

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

[8] Natural Language Computing Group, Microsoft Research Asia. r-Net: Machine Reading Comprehension with Self-Matching.

Supplementary Info

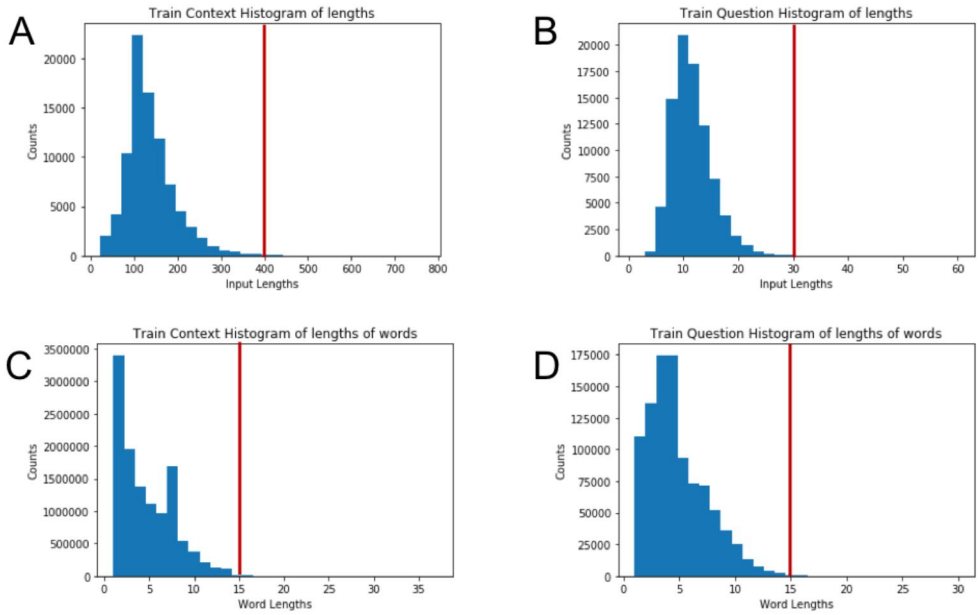

Figure 4: The frequency distribution of A) context input length within the training set, B) question input length within the training set, C) number of characters within each word of the context training set, D) number of characters within each word of the question training set. The red line indicates the max length input we used for the word and character level embeddings.

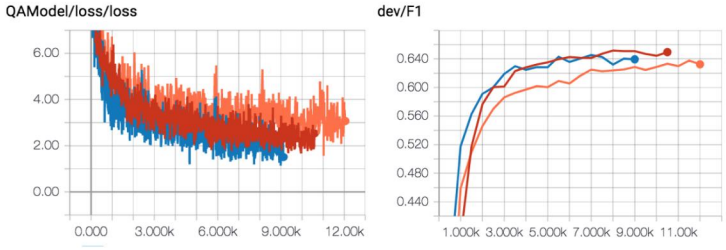

Figure 5: The training loss (left) and dev F1 score (right) vs. number of iterations for bidaf + bidaf output model with dropout of 0.05 (blue), 0.15 (red), 0.25 (orange)