Question Answering with Bi-directional Attention Flow and Self-Attention

Olivier Pham
Management Science & Engineering
Stanford University
mdopham@stanford.edu

Yuguan Xing
Computer Science
Stanford University
yuguanx@stanford.edu

Abstract

For this project, we implement a Bi-directional Attention Flow layer stacked with a Self-Attention layer coupled with improvements on the output layer to answer the The Stanford Question Answering Dataset(SQuAD). Our model obtained an F1 score of 74.2 and an EM score of 63.1.

1 Introduction

Automatically answering questions is currently one of the main topic of research in Natural Language Processing. This is all the more essential as we have seen the rise in 2017 of smart assistants with Google Assistant, Amazon Alexa and Siri which all rely on a powerful NLP model to interact with users.
This is a difficult task though, as being able to answer a question requires understanding not only the question but also the original text from which the answer comes from.

In this paper, we first present the problem we are trying to solve along with the dataset we worked on. In the second part, we introduce our model and describe the different implementation details. Finally, in the last part, we analyze our results and the different sources of errors. We then conclude with a summary of the interesting results and suggestions for future research.

Dataset & Metrics

The Stanford Question Answering Dataset (SQuAD)[1][1] is a reading comprehension dataset compiled by a group of Stanford University computer scientists. It consists of more than 100,000 pairs of questions and answers on a set of 536 Wikipedia articles. Each of the answer to every question is a segment of text from the corresponding reading passage.
The metrics used to evaluate the quality of a model are the Exact Match (EM), which is the percentage of exactly correct answer, and the F1 score which is the harmonic mean of precision and recall. A baseline model consisting of a simple neural network achieves an F1 score of 40% which significantly lower than the human performance of 91%. Hence this leaves a lot of room for improvement and make this dataset a standard evaluation of NLP models' performance.
For this project, the data is randomly partitioned into a training set of approximately 86300 question-answers (89%) and a dev set of approximately 10400 question-answers (11%). Around another 10 000 question-answers are kept secret to test the model on the leaderboard.

2 Model

Our approach comes from experimentations based on the suggestions in the project handout. We started by implementing each of the attention model and then added different improvements in order

[1]https://rajpurkar.github.io/SQuAD-explorer/

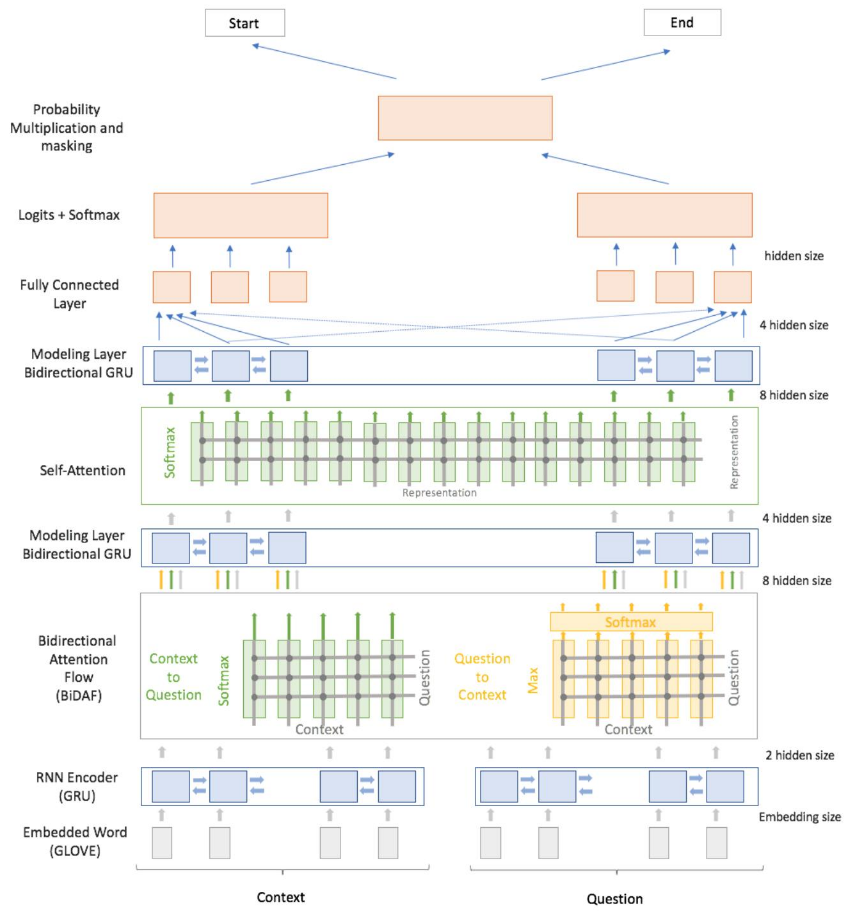


Figure 1: Structure of our model (graph inspired by the original BiDAF paper [2])

to find the best trade-off between having a good score and having a reasonable model size to avoid Out Of Memory (OOM) errors.
The structure of our model is presented in figure 1.

2.1 Embedding Layer

We used pretrained GloVe [3] embeddings to map each word to its corresponding vector. After having tried 4 different embedding size (50, 100, 200 and 300), we decided to use the embedding size of 200 given that it lead to much better results than embedding size of 50 or 100, and similar results to 300.
For the word encoding, we used a GRU cell instead of an LSTM cell given that we found that LSTM led to similar results using the same hyperparameters, but had more parameters, hence limiting the available memory and model size.

Finally, we decided against retraining our word embeddings or training characters embedding given the relatively small size of the training set (GloVe was trained on all Wikipedia articles) even though it would have helped learning new words. Moreover, the table comparison in the BiDAF paper [2] showed only a 2% improvements on the F1 score with char embeddings, hence, we chose to experiment other improvements instead.

2.2 Attention Layer

2.2.1 Bi-directional Attention Flow [2]

We first implemented the bidirectional attention flow layer which consists of adding an attention from the questions to the context but also from the context to the questions. The first step is to compute the similarity matrix $S = w[C, Q, C \circ Q]$. In order to efficiently compute it with Tensorflow, we split the calculation in three parts and used $w(C \circ Q) = (w \circ C)Q$. Therefore $S = w_1 C + w_2 Q + (w_3 \circ C)Q$.
Once the Similarity matrix is computed, we get the Context-to-Question (C2Q) Attention by calculating $\alpha^i = softmax(S_{i,:})$, and then $a_i = \sum_{j=1}^{M} \alpha_j^i q_j$. The Question-to-Context is obtained by calculating $\beta = softmax(max_j S_{i,j})$ and $c' = \sum_{i=1}^{N} \beta_i c_i$. Finally, the output $[c_i, a_i, c_i \circ a_i, ci \circ c]$. We used the masked softmax which ignores padded entries to compute the softmax in the two cases above.

We found that adding a modeling layer after the bi-directional attention flow consisting of a bidirectional RNN made of GRU cells enabled a significant score improvement of almost an additional 10% on a local dev F1 score. It is reasonable as it contextualize the output of the bi-directional attention flow and summarize it.

2.2.2 Co-Attention [4]

We also implemented co-attention which is similar to BiDAF as is contains a Question to Context attention, but also a Context to Question attention. However, there is an additional attention computation layer where we attend representations which are outputs of the previous attention layer.
We found however that BiDAF performed better both in term of final dev F1 and EM score, but also in term of computational resource. As a result we decided not to go any further with co-attention.

2.2.3 Self-Attention [5]

The self-attention layer is proposed as the key component in the R-Net [5] model as a solution to the general problem in recurrent neural networks that they cannot effectively aggregate evidences useful for question answering, given very long passages. The self-attention layer retains two trainable matrices that are used to compute weighted sums between context representation at any given location with those at other locations in the passage, which is then passed to a softmax layer to compute an attention distribution among all locations in the context passage. The model computes a weighted average of context representations at all locations using the attention distribution, and generates as output a blended representation of the context, again using bidirectional GRU units with concatenations of the original representations and the weighted averages as inputs. Intuitively, this mechanism is learning the relationships between different parts of the passage, and for any given location in the context, gather information from other parts of the passage that would be useful for answering given questions.

In our model, we added this layer after the bi-directional attention flow layer.

2.3 Output Layer

2.3.1 End Position Pointer

We implemented a simplified version of the end position pointer presented in R-Net paper [5]. While still having the distribution of the end pointer depend on the distribution of the start pointer, we simply concatenated the start position distribution to the blended representations (output of the second modeling layer) and fed it through a bidirectional GRU. We then take the softmax of the output as our end pointer distribution.
Using the same hyperparameters as the previous model led to an Out-Of-memory error. We therefore had to reduce the hidden size from 120 to 100. After training it, we found that this smaller model with pointer achieved exactly the same result as the previous model without pointer but with hidden size 120. Hence, we decided not to use the end position pointer in our final model.

2.3.2 Inference Rule

Since the inference mechanism implemented in the baseline model outputs start position and end position separately, it is possible that the inferred end position is before the start position. In such cases, the model would output an empty string. We analyzed the model's outputs on `tiny-dev.json`, and demonstrate the result in table 1.

end_pos - $start\_pos$	counts
< 0	20
0	292
1	257
2	131
3	54
4	19
≥ 5	37

Table 1: Distribution of the difference between inferred end position and start position on tiny-dev.json

From the table, we can see that the model outputted empty strings on about 2.5% of the data points, which has a negative influence on the accuracy of predictions. By only taking positions after the start position as candidates for end positions during inference, the F1 score on the development set improved by 1.

Furthermore, as mentioned in the DrQA model by Chen et al.[6], it is usually helpful to set an upper bound to the answer length, since most answers are rather short. Figure 2 demonstrates the distribution of lengths of true answers as included in `tiny-dev.json`.

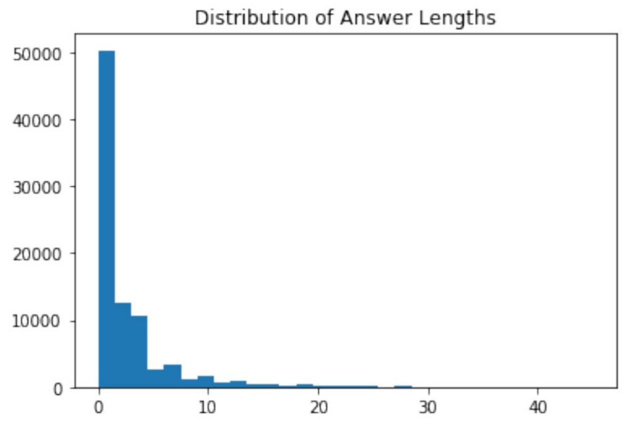


Figure 2

From the figure, we can make the observation that most answers have lengths that are less than 20 tokens. The authors of [1] set the restriction that the end position must be within 15 tokens from the start position. However, setting this restriction, using the cutoff values 15 or 20, caused a decrease in our model's F1 on the development set. Therefore, in the version that we submitted, we did not enforce this restriction.

2.4 Training Details

We trained our model on the training dataset as introduced in section 1. On an NVIDA Tesla M60 GPU, it takes 2.5 seconds on average to run an iteration on a batch of 100 data points. The model achieved highest development F1 after 9,500 iterations.

2.4.1 Hyperparameter Tuning

Some crucial hyperparameters in this problem are maximal context length allowed (`context_len`), maximal question length allowed (`question_len`), length of first hidden representations generated by the RNN encoders (`hidden_size`), size of trained word vectors (`embedding_size`), and dropout rate used in all RNNs. During our experiments, we observed that `context_len` was a major bottleneck in terms of memory resources. However, provided that the true answer span is within cutoff range, increasing its value does not contribute to a significant boost in model performance. Therefore, it would be reasonable to make it as small as possible, while ensuring the coverage of answer spans. In order to find an optimal value for `context_len`, we plotted the distribution of lengths of context paragraphs and that of start and end locations of true answers. The results are demonstrated in Figures 3, 4 and 5.

Figure 3

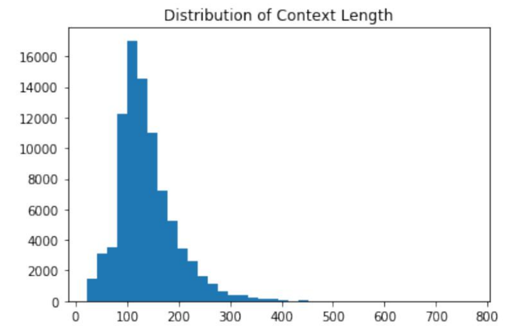


Figure 4

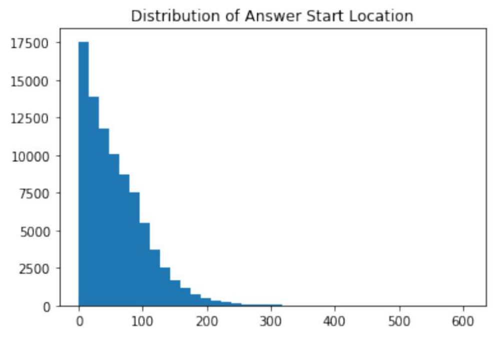
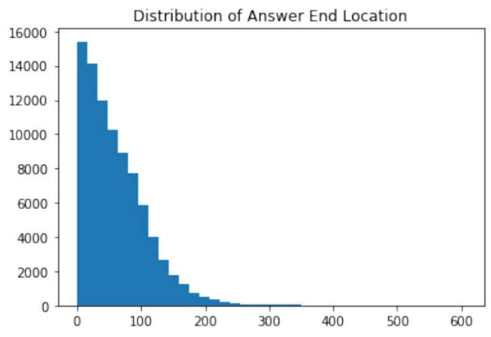


Figure 5



From the above, we noticed that it is safe to set `context_len` to 350 without severely damaging the performance. This contributed to a major memory save, which enabled us to maintain a relatively large `hidden_size`, which we set to 120. We performed experiments with dropout rate 0.1, 0.2, 0.3, 0.4 and 0.5, and concluded that 0.2 was the optimal value. By increasing the `embedding_size` from 100 to 200 we gained a major boost in performance, but 200 to 300 did not demonstrate the same effect. Therefore, we used 200 in our submitted model.

3 Results and discussion

3.1 Results

We achieved a F1 score of 74.2% on the dev set and 74.5% on the test set, and an Exact Match score of 63.1% on the dev set and 64% on the test set.
The results for each of our experiments are summarized in the table 2 below where SA means Self-Attention, BiGRU corresponds to the bidirectional GRU modeling layer and Infer to the inference rule described previously.

Model	Local Dev EM/F1	Dev Leaderboard EM/F1	Test Leaderboard EM/F1
Baseline	29/40	34.6/43.8	
Only BiDAF	45/56		
BiDAF + Bidirectionnal GRU	51/65		
Only Self-Attention (SA)	46/59.6	54/65.1	
BiDAF + biGRU + SA	52/67.3	62.2/73	
BiDAF + biGRU + SA + Infer	52.8/68.1	63.1/74.2	64/74.5
BiDAF + biGRU + SA + Infer + Pointer	52.5/68.1		
BiDAF (reference, single model)		67.7/77.3	68.0/77.3
Self-Attention (reference, single model)			68.4/77.5
Co-Attention (reference, single model)			66.2/75.9

Table 2: Results of our different approaches on the dev set and test set

We can see that the main improvement comes from the attention and modeling layer which made it possible to obtain an F1 dev score above 70%. The final improvements did not significantly improve the score.
Our final model competitively scored only 3 points below the original BiDAF implementation.

3.2 Error analysis

Examples

We first analyzed random misclassified samples generated to find where errors come from for our final model.

In most cases, there is an overlap between the correct answer and the predicted answer - meaning that the F1 score is above 0 - but the position of the start or end pointer where just incorrect by one position or two. For example:

1. - Context: ”in second place in total viewership behind long-dominant cbs”
   - Question: ”in 2013-14 , nbc finished behind what network in the ratings ?”
   - Correct Answer: ”cbs”
   - Predicted Answer: ”long-dominant cbs”
2. - Context: ”the first recorded settlement in what is now newcastle was pons aelius , a roman fort and bridge across the river tyne .”
   - Question: ”what river was there originally a bridge across in roman times ?”
   - Correct Answer: ”tyne”
   - Predicted Answer: ”river tyne”

For the two cases above, we can see that the error comes from not knowing the type of the word ”long-dominant” and ”river”: if the model knew that those words are respectively adjectives and nouns it would probably not have included them in the answer. It would be interesting to implement the token features (Part-of-Speech tag, the Named Entity type and the Normalized Term Frequency) suggested in the DrQA [6] paper to improve on this.

Another error comes from incorrectly annotated answers, where the predicted answer is perfectly acceptable as well:

- Context: ”the ussr ’s invasion of afghanistan was only one sign of insecurity in the region”
- Question: ”which country ’s invasion show the insecurity of the middle east ?”
- Correct Answer: ”ussr ’s invasion”
- Predicted Answer: ”ussr”

The ”invasion” should not be part of the answer given that the question is about a country. This suggests that the SQuAD dataset is not perfect.

Finally, another source of error comes from having answers that are too long. For example:

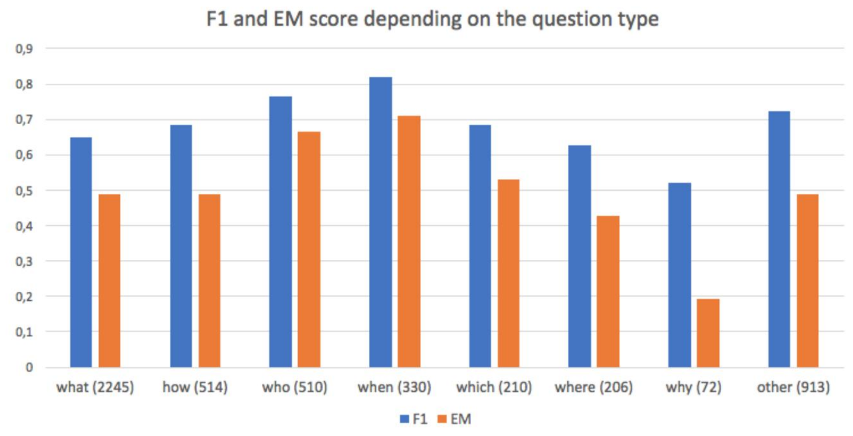


Figure 6: F1 and EM score for different question types, the number in parenthesis is the number of occurrences for the question type. The overall F1 score on this dataset is 0.68 and the overall EM is 52.

- Context: ”currently there is no indication as to whether the new deal includes the additional video on demand and high definition content which had previously been offered by skyb”
- Question: ”does the new deal include video on demand and high definition ?”
- Correct Answer: ”no”
- Predicted Answer: ”no indication as to whether the new deal includes the additional video on demand and high definition content”

This problem comes from the lack of interaction between start position and end position. However, we still obtained the same error with our basic pointer model even though we suspect it might be experiencing vanishing gradient. It would be interesting to implement a more robust pointer or to use dynamic programming to introduce a smarter relation between the start position and the end position.
This example raises two other questions. First, the answer here is clearly a yes/no, so the answer should not be that long: the behavior of the model should depend on the question type. Then, it also raises the question of whether our model performs better or worse on long answers compared to short answers.

Comparison of different question types

We separated the F1 and EM score calculations depending on the type of the question and ran it over 5000 examples in dev set. We obtained the graph in figure 6.

It is interesting to notice how **who** and **when** questions perform extremely well compared to the other types of questions. Indeed, **who** and **when** questions are usually straightforward and require simply a one or two words answer corresponding to the name or the date. On the other hand, **why** questions are extremely difficult to answer to especially since they require justifications and therefore longer answer. This can be seen with the very low EM score. To verify that assumption, we will compare the F1 and EM score to the length of the answer in the next section.

Comparison of various answer length

In figure 7, we plot the F1 score and EM score as a function of the correct answer length. Like in the previous section, this is computed using 5000 examples from the dev set.

The scores are decreasing as a function of the answer length: indeed the longer it is, the harder it is to correctly capture its beginning and end.

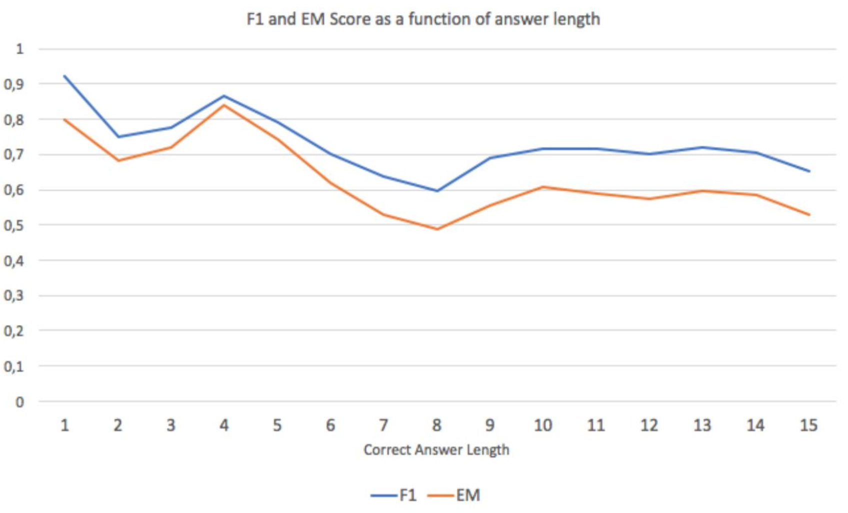


Figure 7: F1 and EM score as a function of the correct answer length

4 Conclusion and future work

In this project, we implemented a bidirectional attention flow layer and a self-attention layer coupled with improvements on the output layer which achieves competitive results on the leaderboard, only 3 points below the original BiDAF implementation.
The analysis of the results shows that, while the attention region is almost always correct, most errors comes from the lack of information about the words and their part-of-speeches. Hence future work could include implementing handcrafted features for the words, or using other word embeddings such as ELMo [7].

References

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

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

[3] Jeffrey Pennington, Richard Socher, Christopher D. Manning. GloVe: Global Vectors for Word Representation, 2014

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

[5] Natural Language Computing Group, Microsoft Research Asia. R-Net: Machine Reading Comprehension with Self-Matching Networks, 2017

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

[7] Deep contextualized word representations Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer. NAACL 2018