CS224N: Question-Answering Utilizing Bidirectional Attention Flow

Wesley C. Olmsted
Department of Computer Science
wolmsted@stanford.edu

Trevor S. Danielson
Department of Computer Science
trevord@stanford.edu

Abstract

We investigated a question answering model using bidirectional attention flow. We evaluated different features from various papers in our model. We were able to score a respectable F1 Score of 71.0 and EM score of 59.9.

1 Introduction

Question answering requires a great deal of machine natural language understanding. In our problem, the machine learns to find an answer in a context paragraph given a query. We used the Stanford Question Answering Dataset (SQuAD) to evaluate various techniques from other papers. These techniques included using advanced forms of attention, optimizing hyperparameters, and feature construction. There has been a great amount of research done using the SQuAD dataset, and we chose to use techniques and insights from this previous research.

2 Problem Definition

Our goal is to answer the question given based on the context given. The way we do this is by building a classifier that will classify a "start" and "end" index within the context, where the answer is contained. So given a context with T tokens and a question with J tokens, our model will classify a start index, x, and end index, y, where $0 \leq x \leq y \leq T$.

3 Related Work

The SQuAD dataset has been out for around two years and there have been many publications investigating different models. One model by the Joint Laboratory of HIT and iFLYTEK Research was able to achieve near human F1 and EM scores [3]. We decided to focus on two papers, namely the BiDAF and DrQA paper. Both of these models achieved high F1 and EM scores so we implemented various parts of both.

4 Features and Data

Our training set contained 86,326 training examples and our dev set contained 10,391. These samples came in the following format:

> **Question:** Which NFL team represented the AFC at Super Bowl 50?
> **Context:** Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers...

Answer Start: 29
Answer End: 30

Our data contains a context string, a query string, and the start and end index of the answer within the context. Besides the corresponding embeddings to the tokens within these strings, we added one extra features: a bit that is set to on if a context token appears in the question. For example the context token ”Super” will have a 1 concatenated to the word embedding because ”Super” also occurs in the question.

In addition, we analyzed our training data in order to better understand the possible effects of changing the specified context and question lengths during the embedding layer. Figure 1 shows a histogram of answers location within the specified context. We decided to lower the context length to 550 tokens in order to cut down on memory overhead associated with longer contexts. Analyzing figure 1 shows that a about 3,500 more training examples would be prematurely trimmed of their answers during embedding compared to the baselines 600 token length, which is about 4 percent of the total number of training examples. Based on Figure 2, which shows the question lengths, we decided to keep the question length to 30 tokens. All questions in our dataset were shorter than 30 tokens making it optimal to keep it at this point.

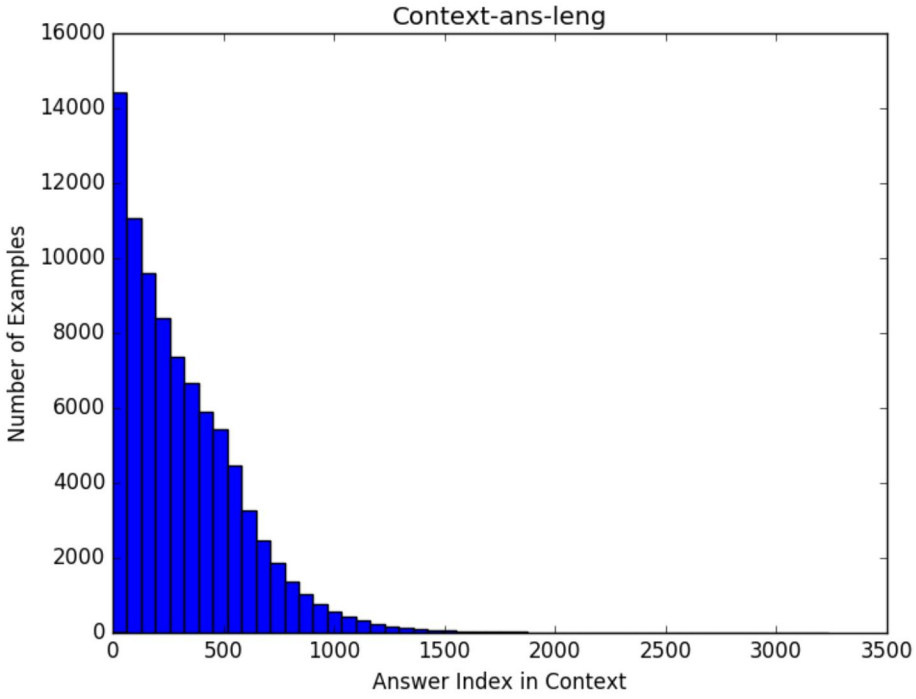

Figure 1: Histogram of Answer Location Within Context

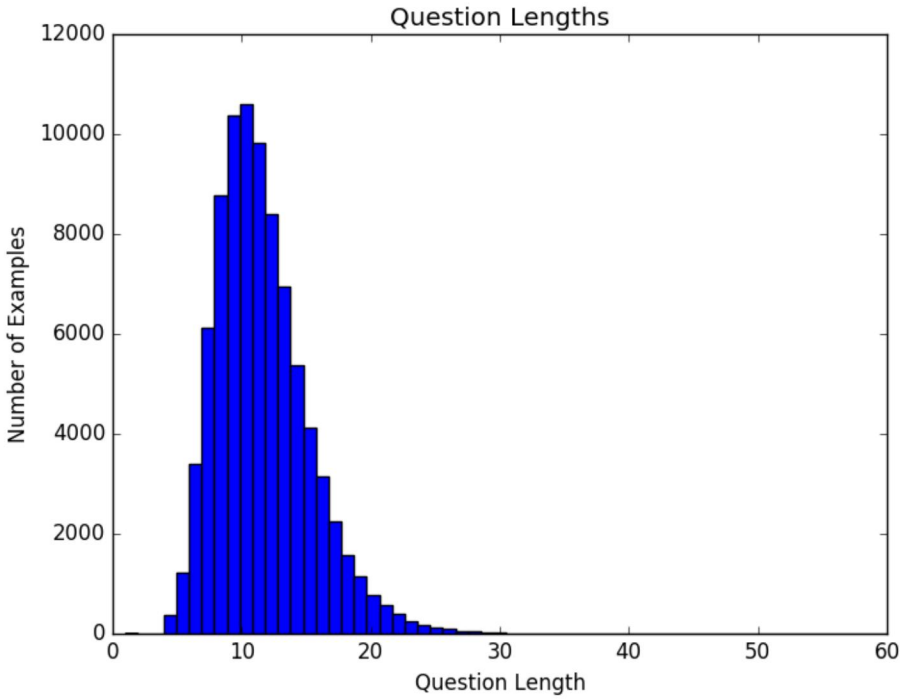

Figure 2: Histogram of Answer Location Within Context

5 Model

5.1 Baseline Model

The baseline model we started with used a simple attention layer that found the similarity between each context word and the question sequence. This was done for us by calculating the dot product between the two. Then we use the softmax function on this vector and calculate the weighted sum for every one of the question hidden states. We then append this vector to our original context. From there, the concatenated matrix is passed through a fully connected layer, and then is finally passed twice through two separate fully connected layers and a softmax layer, one for the start and one for the end.

5.2 BiDAF Model

We mostly followed the guidelines of the model implemented in the BiDAF paper [1]. The one part we chose not to implement was the final LSTM layer that conditions the end index on the start index.

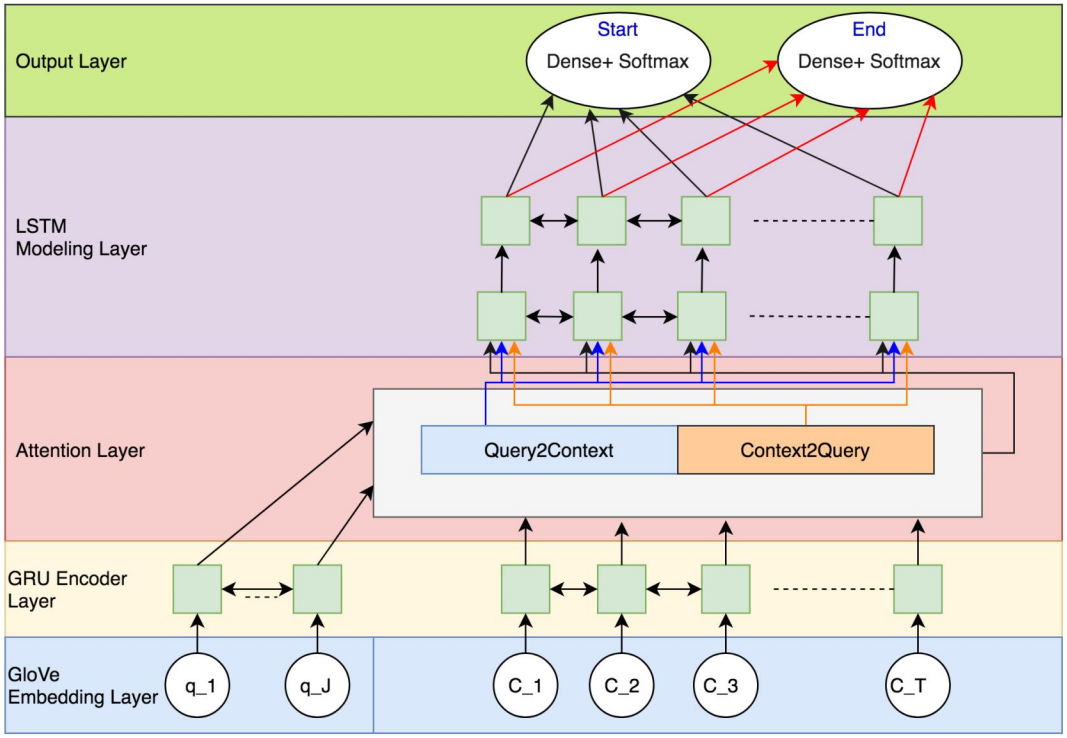

Figure 3: Bidirectional Attention Flow Modeling Architecture

5.2.1 The Encoder Layer

In this layer, we encode our questions and contexts. We pass both the question embeddings and context embeddings through a bidirectional GRU. We decided to keep GRU cells instead of LSTM cells because of their higher efficiency. Once we pass the embeddings through the GRU, we have question sequence length, M, question hidden states and context sequence length, N, context hidden states. These are also double the hidden state size due to concatenating the outputs of both flows of the bidirectional GRU. This layer is shown in yellow in Figure 3. We then pass in these concatenations into our Attention layer.

5.2.2 The Attention Layer

The first thing we implemented was the attention layer. Instead of our basic dot-product attention, we used a more advanced attention that takes into account query to context attention and context to query attention. This is calculated by first generating a similarity matrix, $\mathbf{S}$ where each term $\mathbf{S}_{ij}$ is the similarity between hidden context state $\mathbf{c}_i$ and hidden question state $\mathbf{q}_j$. $\mathbf{S}_{ij} = \mathbf{w}_{sim}[\mathbf{c}_i; \mathbf{q}_j; \mathbf{c}_i \circ \mathbf{q}_j]$. The matrix multiplication of the weights vector, $\mathbf{w}_{sim}$ and $[\mathbf{c}_i; \mathbf{q}_j; \mathbf{c}_i \circ \mathbf{q}_j]$ requires storing multiple large matrices in memory so we decided to implement this by separating the weights. We separated $[\mathbf{c}_i; \mathbf{q}_j; \mathbf{c}_i \circ \mathbf{q}_j]$ into 3 matrices instead of concatenating them together. We then had corresponding weights for each of these three. We could then calculate the products of the three pairs and then sum them to get the actual $\mathbf{S}$. Once we have $\mathbf{S}$, we calculate the context to query attention (C2Q). We calculate the softmax on each row i of $\mathbf{S}$ and then create a weighted sum. For the query to context attention (Q2C), we take the max over each row and pass that through a softmax. From this softmax, we create another weighted sum. After this calculation is done, the output of our attention layer is $\mathbf{output}_i = [\mathbf{c}_i; \mathbf{a}_i; \mathbf{c}_i \circ \mathbf{a}_i; \mathbf{c}_i \circ \mathbf{c}']$ where $\mathbf{c}$ is our context hidden states from the contextual layer, $\mathbf{a}$ is our C2Q attention and $\mathbf{c}'$ is our Q2C attention. This layer is seen in red in Figure 3.

5.2.3 The Modeling Layer

We take the output of the attention layer and pass that into a two layer bidirectional LSTM with hidden size 128. The outputs of the context sequence size are then concatenated for both directions. We decided on two layers because that is what the BiDAF paper uses [1]. Extending the amount of layers also had less of an improvement for the extra time that was needed to train. From the modeling layer, the outputs are passed independently to a dense and softmax layer to get start and end prediction. This layer is seen in purple in Figure 3.

5.2.4 Hyperparameters

We used 200 dimensional word vectors instead of 100 dimensional ones, which was made possible by reducing our context length. For our learning rate, we used a value of 0.001 because we noticed that the values an order of magnitude larger would lead to premature convergence and values an order of magnitude smaller would lead to very long training times. We also began our first experiments with a dropout value of 0.2, which we applied to every layer except the final ones. We eventually raised the dropout to 0.22 because at later epochs, our train and dev set f1 scores would begin to increase to about a 15 point difference. We experimented with various optimizers including Adagrad, Adam, and Adadelta. We found the balance between shorter computation time along with higher dev f1 scores with the Adam optimizer. We achieved our best dev scores at 150000 iterations.

6 Results

Our implementation saw a significant increase to both F1 and EM metrics, about 27 percent and 25 percent respectively, as seen in table 1. Comparing the results to our reference models [1][2], our model scored 6-7% below on both EM and F1. Comparing to the DrQA model, which was our reference for the additional feature, we were unable to obtain the same level of performance with our architecture.

Table 1: Model F1 and EM Dev Set Scores

Model	F1	EM
Baseline	44.0	34.7
BiDAF (Our implementation)	71.0	59.9
BiDAF [1] (reference)	77.3	67.7
DrQA [2] (reference)	78.8	69.5

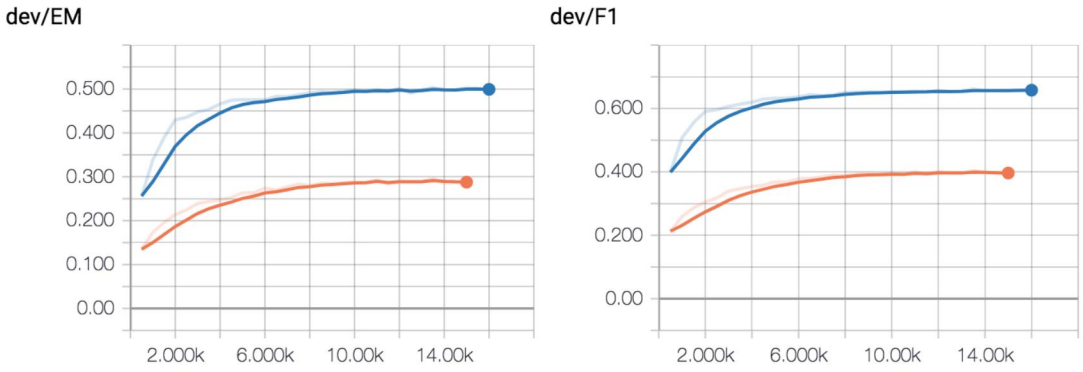

Figure 4: Baseline Comparison to our BiDaf Model

7 Analysis

The main decision we had to make when deciding our best model was whether to use the extra feature (a bit set to on if a token from the context is in the question). We noticed the training loss

dropped rapidly when we initially added the question-match feature, but it quickly started to level off, and did so faster than our BiDAF with standard features model. This behavior is shown in Figure 4.

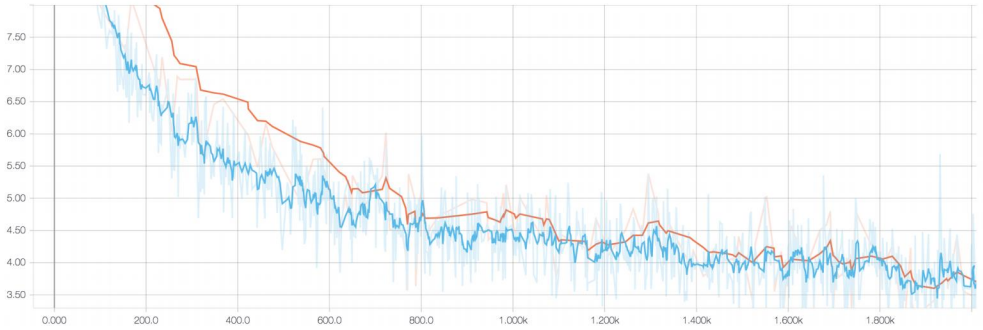

Figure 5: Train Loss With and Without Question-Match Feature

For qualitative analysis, we noticed some examples that our model was not very good at identifying. Most notably, our model struggled when commas separated answers. In a set of 10 randomly evaluated examples, we found these errors:

Question: where is the apache point observatory located ?
True Answer: sunspot , new mexico
Predicted Answer: new mexico
F1 Score Answer: 0.800
EM Score: False

Question: what type of studies explored student motivation ?
True Answer: controlled , experimental studies
Predicted Answer: experimental
F1 Score Answer: 0.500
EM Score: False

Question: how are green chloroplasts ' plastoglobuli arranged ?
True Answer: singularly , attached directly to their parent thylakoid
Predicted Answer: attached directly to their parent thylakoid
F1 Score Answer: 0.923
EM Score: False

As shown by these examples, errors related to separating commas have a large, adverse effect on the EM score. The F1 score is usually not impacted too much unless the answer is short.

8 Conclusion and Future Improvements

Overall, we were able to achieve a decently high F1 and EM score. Our score was slightly below the two papers we referenced, BiDAF and DrQA. We believe we could achieve a slightly higher score if we worked on this more and implemented some of the below improvements.

As discussed previously, the inclusion of the additional feature, the model seemed to function significantly better initially, before leveling off to about the same as the original BiDaf model. One improvement that may prevent this eventual leveling, would be to change the optimizer used. We used Adam for our models, however the use of Adamax could have proven beneficial as it was employed in the DrQA model [2].

Also noteworthy, was the difficulty the model had in predicting exact matches (See examples in Features and Data Section) which included a comma somewhere within the correct answer. We

hypothesize that increasing the the GloVe vectors' dimension from 200 to 300 would allow for our model to locate and learn the intricacies inherent in the use of commas. According to the original GloVe paper, 300-dimensional word embeddings work the best for finding relationships between tokens [4]. We believe that this same behavior would apply well with commas. The relationship of commas separating two adjectives would be represented in these 300 dimensions most likely.

Another potential area for improvement with regards to this issue would be to implement a Character Level Convolutional Neural Network (CNN). Improving the ability to answer these types of examples would have the most significant impact on our EM score, as the current model still seems to perform at a rate similar or slightly higher than the overall model score of 71.0 with regards to F1 score. However, there would still see a slight improvement to F1 as well, but not on the same level as EM increase.

Given the nature of hyper parameter tuning, in which the effects of changes are usually only visible at later iterations, this causes it to be extremely costly both in time and money. With even more time to experiment, we would be able to optimize our model's hyper parameters in order to get the most out of out architecture.

In terms of training, we could have set which embeddings we chose to train instead of training on all embeddings. In the DrQA paper, they only train the embeddings of the 1000 most common words in the questions [2]. We estimate that some sparse changes to our word embeddings may have slightly skewed how they function in question answering.

Acknowledgements

Many thanks to the CS224N teaching staff for a wonderful quarter. We both learned a lot in this course, and will continue to apply this knowledge in the future.

References

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

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

[3] Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-overattention neural networks for reading comprehension. arXiv preprint arXiv:1607.04423, 2016.

[4] R. Jeffrey Pennington and C. Manning. Glove: Global vectors for word representation. 2014.