A Bidirectional Attention-Based Approach to Machine Comprehension and Question Answering

Kevin Chen
Department of Computer Science
Stanford University
kvchen@stanford.edu

Abstract

In this paper, we replicate the results of the bidirectional attention-based approach to question answering first introduced by Seo et al. [1]. We attempt to improve upon this model using self-attention as proposed by Wang et al. [2] in the R-net architecture, but find that using a combination of the two attention types fails to produce an improvement in accuracy. The performance of our final model architecture achieves comparable results to the original papers on the Stanford Question Answering Dataset (SQuAD) [3], with an F1 score of 76.74 and an EM score of 67.86.

1 Introduction

Question Answering (QA) is a particularly well-known problem in the field of machine comprehension, in which a model attempts to locate the answer to a query within a passage of text. That is, given a context C (the passage of text) and a query Q about the context, we want to find the span of contiguous words in the context $(C_{start}...C_{end})$ that forms the correct answer A. For example:

> **Context paragraph**: Organized crime has long been associated with New York City, beginning with the Forty Thieves and the Roach Guards in **the Five Points** in the 1820s. The 20th century saw a rise in the Mafia, dominated by the Five Families, as well as in gangs, including the Black Spades. The Mafia presence has declined in the city in the 21st century.
>
> **Question**: The Forty Thieves and Roach Guards were two gangs that operated in what area of New York in the 1820s?
>
> **Answer**: the Five Points

This task still poses a challenge for many machine learning models, because the interaction between the context and the query can be very complex. As evidenced by the online leaderboards [3], this challenge has been a recent area of focus for many research groups and companies.

2 Background/Related Work

Previous approaches in this area have made heavy usage of recurrent neural networks (RNNs), which output a hidden state for each timestep in a sequence. This can be used to generate a vector representation that encodes information from each of the context and the question, similar to the idea behind word embeddings. The problem with this approach is that most models utilize only the final hidden state from the RNN, which may have already forgotten information from the beginning of the sequence by the time it reaches the end of the sequence. This makes long-term sentence dependencies difficult to deal with.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

To counteract this issue, recent approaches have centered around the idea of attention. The idea behind attention is simple; we want the decoder to focus on different parts of the input at each step of the decoding process. To accomplish this, all the hidden states from the RNN are combined to generate a new sequence, which is then fed into the decoder. By looking at a combination of all the hidden states and not just the final state, the attention model is able to more accurately determine which parts of the context and query are relevant.

In this paper, we investigate two different attention-based methods: bidirectional attention and self-attention. We look at ways to combine these two different methods and effects this has on the model accuracy. Our final model most closely resembles the BiDAF architecture and uses a bidirectional attention flow to create context and query representations that encode information about each other.

3 Approach

Our final approach is heavily based off previous work in the BiDAF architecture [1] with an additional self-attention layer [2]. These layers are documented in detail in their respective papers, so we only provide an abbreviated version of their implementations here.

3.1 Character Embedding Layer

We begin by using character embeddings trained on the GloVE dataset [4]. These embeddings are fixed-sized vectors that encode information about the most common ASCII characters. In this layer, we map each word in the input sequence into a character-level representation.

First, we preprocess each word by normalizing its Unicode values into ASCII equivalents as best as possible. For instance, the string "aä" is converted into "aa" prior to being fed into the character embedding layer. Next, we map each letter into its corresponding embedding. We take these embeddings $e_1, \ldots, e_L$ and pass them through a convolutional neural network (CNN) to produce a sequence of representations $h_1, \ldots, h_L$. Max-pooling is applied to the representations to produce a final character-level encoding.

3.2 Word Embedding Layer

We again use word vectors trained on the GloVE dataset [4]. We simply concatenate the character-level embedding to the word-level embedding to produce the final embedding for each token.

One area in which we deviate from the original paper is that we omit the highway network layers as described by Srivastava et al. [5]. We found that using this network tended to drastically decrease our model's performance, but this could also have been due to an incorrect implementation or poor choice of hyperparameters.

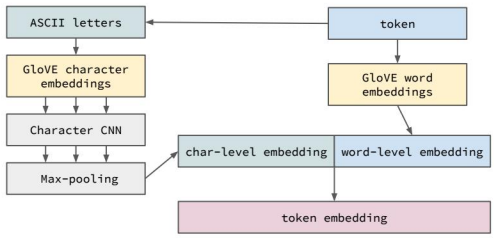

Figure 1: Token embeddings

3.3 Contextual Embedding Layer

This layer models the temporal interactions between nearby words to produce refined embeddings. This is accomplished using a 2-layer RNN with GRU cells in either direction. The output of this layer is another sequence of contextual embeddings. We share weights for this layer between the context and query networks, as we found that it helped to boost the accuracy of our model.

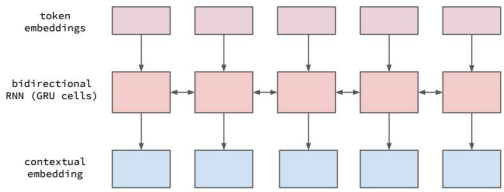

Figure 2: Contextual embedding

3.4 Bidirectional Attention Layer

This layer allows attention to flow both from the context to the question and from the question to the context. This ends up producing a series of query-aware context representations.

The first part of constructing this layer is to create a similarity matrix S, which encodes the pairwise similarity between the context and query words. We use a trainable weight w_{sim} such that for the context C and query Q, we have:

$$S_{ij} = w_{sim}^T [c_i; q_j; c_i \circ q_j] \tag{1}$$

We use S to compute the context-to-query (C2Q) attention, which determines which query words are most important to each context word. We also compute the query-to-context (Q2C) attention, which determines which context words are most similar to each query word. These two attention types are combined to form the bidirectional attention flow layer.

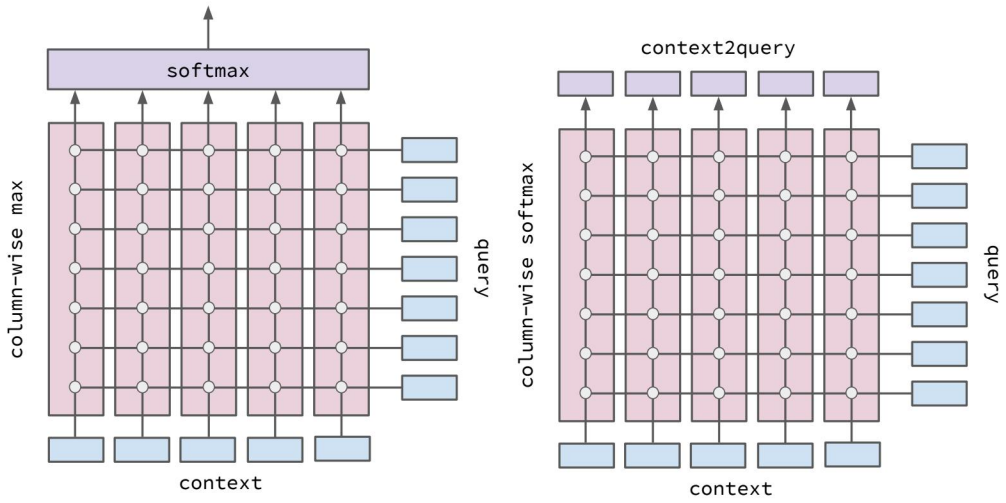

Figure 3: Bidirectional attention

3.5 Self-Attention Layer

Although it didn't end up in our final model, we also tried adding a self-attention layer immediately after the bidirectional attention layer. Self-attention allows us to aggregate information from the entire context in determining the answer.

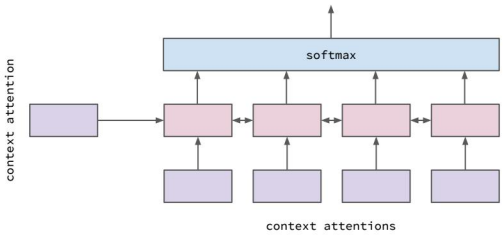

Figure 4: Self attention

3.6 Modeling Layer

Our modeling layer takes in the query-aware context representations G and, similar to the contextual embedding layer, outputs representations modeled around interactions with nearby words. These representations M are obtained by passing G through a two-layer bidirectional RNN.

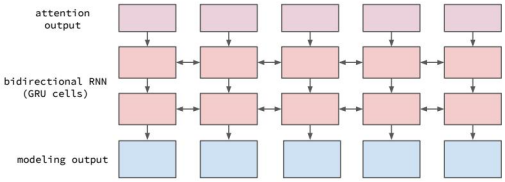

Figure 5: Modeling

3.7 Output Layer

We determine our start index by first taking a softmax over a fully connected combination of the attention output G and the modeling output M.

$$p^1 = softmax(w^T_{(p^1)}[G; M]) \tag{2}$$

We compute the output position by passing M through an RNN, then through a fully-connected softmax layer:

$$M^2 = RNN(M) \tag{3}$$

$$p^2 = softmax(w^T_{(p^2)}[G; M^2]) \tag{4}$$

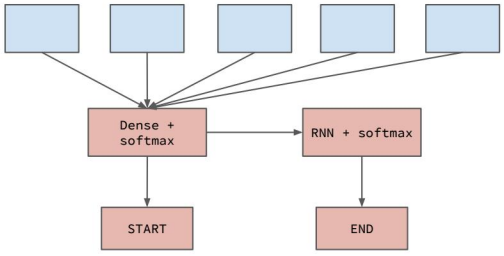

Figure 6: Output

We subject the final indices to the constraint that the end index is within some distance after the start index. That is:

$$x_{start} < x_{end} \leq x_{start} + max\_span \tag{5}$$

4 Dataset and Experimentation

We train our model using the Stanford Question Answering Dataset (SQuAD) [3], which consists of over 100,000 question/answer pairs on over 500 Wikipedia articles. To preprocess this data, we first use the NLTK library [6] to split each sentence up into word tokens. Each word token is further broken down to character tokens by first converting the individual Unicode characters to their ASCII equivalents (or omitting them if they fall outside a useable range). Each of these word and character tokens is then converted into an embedding using pretrained GloVE vectors [4] for use in our model.

To determine suitable hyperparameters for our model, we first looked at the word token lengths for each of the context and the question (Figure 7). We also look at how many characters are in each word for both the context and question (Figure 8). Finally, we examined the answers in our training dataset (Figure 9). We expect our model's output distributions to roughly match these answer distributions.

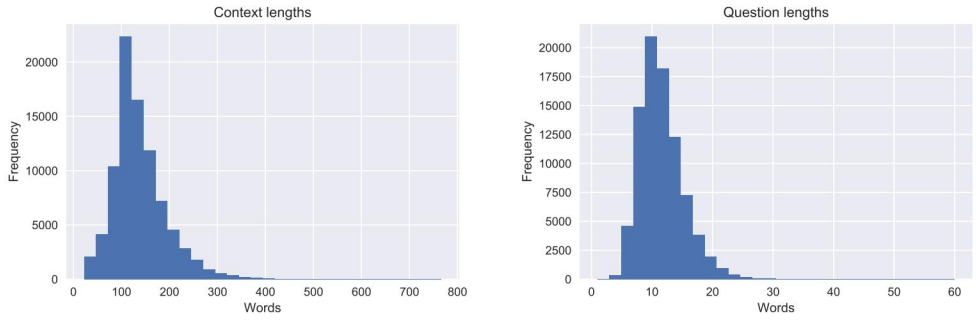

Figure 7: Context and question length histograms

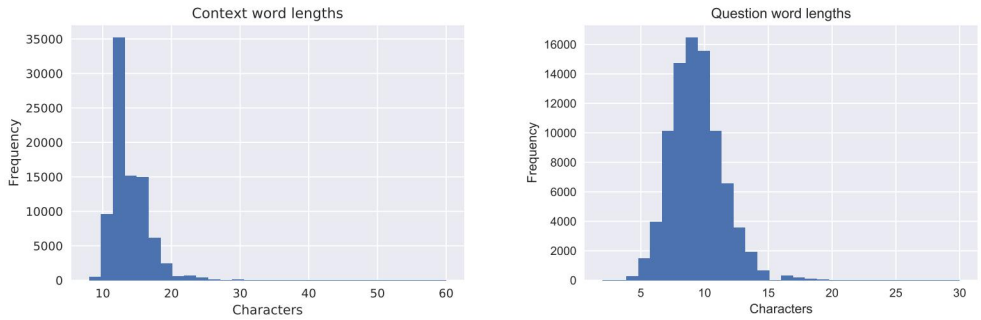

Figure 8: Context and question word length histograms

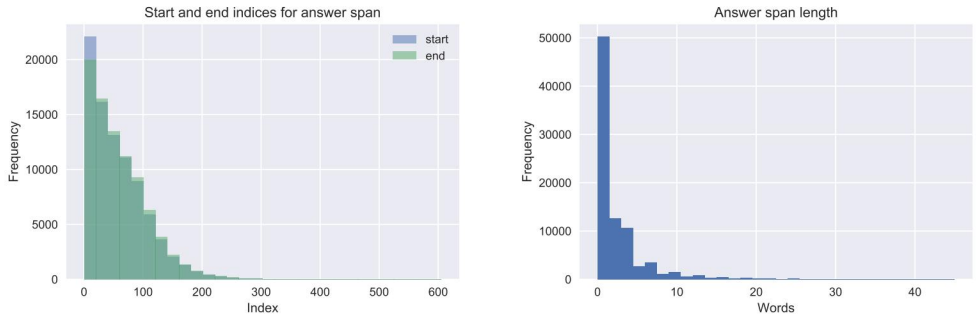

Figure 9: Context and question word length histograms

We performed repeated trials on our final model with different hyperparameters and end up with the final set listed in Table 4. These hyperparameters were generated not only based on the final accuracy of the model, but also the difference between the train and dev scores, as well as the training time for each batch.

Table 1: Hyperparameters

Hyperparameter	Value
Word embedding size	100
Character embedding size	300
Batch size	100
Adam learning rate	0.001
Maximum gradient norm	5.0
Dropout probability	0.2
Maximum context length	350
Maximum question length	25
Maximum answer span	15

4.1 Ensembling

To maximize the performance of our architecture, we used an ensemble of 7 models all trained using the same hyperparameters mentioned above. We use a slightly different strategy from the usual consensus voting model, since the selected answer spans can often vary wildly. Instead, each model outputs a (prediction, certainty) pair for each (context, question) pair in the test set. We take a weighted sum of each prediction by its certainty, and select the prediction with the highest score. This ensures that even if all the answers are different, we still select the one with the highest probability of being correct.

5 Results and Analysis

5.1 Performance

Our model performed fairly well on the final test set, coming in 24th place for F1 and 17th for EM. However, we noticed that our results significantly trailed those of the original papers with a 5 point difference under the original BiDAF paper (Table 2).

Table 2: Results on the SQuAD test set

	Single Model		Ensemble	
Model	F1	EM	F1	EM
BiDAF	77.3	68.0	**81.1**	**73.3**
R-Net	**77.5**	**68.4**	79.7	72.1
Our model	74.7	64.3	76.7	65.9
Baseline	42.9	34.3		

We also looked at how much each feature layer contributed to the overall performance of the model. We took each layer and replaced it with its baseline counterpart (for example, replacing the modeling layer with a fully connected layer instead). These values are computed at training time and use different accuracy rules that results in lower F1 and EM scores from those in other tables. However, the relative impact of each layer is still fairly evident in (Table 3).

Table 3: Layer contributions

Layer	F1 (delta)	EM (delta)
Without char CNN	51.5 (-1.8)	66.8 (-1.8)
Without bidirectional attention	52.6 (-0.7)	67.9 (-0.7)
Without modeling	49.6 (-3.7)	65.0 (-3.6)
Without RNN output	53.1 (-0.2)	68.3 (-0.3)
With self-attention	44.9 (-8.4)	60.2 (-8.4)
Final model	53.3 (+0.0)	68.6 (+0.0)

Of particular note is the contribution of self-attention, which ended up severly impacting our performance. This is likely because self-attention simply doesn't work with the rest of the BiDAF architecture, whether it comes right after the bidirectional layer or otherwise. We failed to investigate how the original R-net architecture uses this particular layer, so plugging it straight into our model was a poor choice without fully understanding the ramifications of this decision.

We broke our performance down into six categories based on question keyword that appeared in that question. We notice that we do best on questions that identify things (what, who), and poorly on questions that require more reasoning (how, why) (Table 4). We attribute this to our attention model; object identification often has shorter answers, so the model is more likely to attend to these important phrases and come up with the right answer. On the other hand, questions that involve reasoning often have longer answers, so our model is less likely to attend to the correct and complete part of the context that contains the answer.

Table 4: Performance on different question types (dev)

Question keyword	Total	Correct	Fraction correct
what	1337	943	0.705310
how	309	195	0.631068
when	173	115	0.664740
who	147	104	0.707483
where	77	51	0.662338
why	7	4	0.571429

5.2 Qualitative Error Analysis

We found that many of our errors fell into the same few categories. The first is simply a failure to attend to the correct part of the context. In these cases, the model selects an important phrase, but oftentimes it misses the overall context in the sentence required ot answer the question. This formed the majority of our errors.

Context paragraph: The previous record was 244 yards by the Baltimore Ravens in Super Bowl XXXV. Only seven other teams had ever gained less than 200 yards in a Super Bowl, and all of them had lost. The Broncos' seven sacks tied a Super Bowl record set by the Chicago Bears in Super Bowl XX.

Question: What team had the lowest downs and yards ever in the Super Bowl as of Super Bowl 50?

Our Answer: Baltimore Ravens

Correct Answer: the Broncos

A second less-common source of answers was ambiguous answers, where there are two potential correct answers the model picks the incorrect one. In the example below, humans would know to pick the first (and likelier) answer; however, this requires prior knowledge outside of the context.

Context paragraph: However, some civil disobedients have nonetheless found it hard to resist responding to investigators' questions, sometimes due to a lack of understanding of the legal ramifications, or due to a fear of seeming rude.

Question: Why have civil disobedients found it hard to reset responding to investigator's questions?

Our Answer: fear of seeming rude

Correct Answer: lack of understanding of the legal ramifications

The third is simply picking incorrect answer boundaries. This impacts our EM score greatly, but has less of an impact on our F1 score. This could likely be solved through a more sophisticated process for determining the final answer span.

Context paragraph: In 1978 a disco version of the theme was released in the UK, Denmark and Australia by the group Mankind, which reached number 24 in the UK charts.

Question: What place did the theme reach in Europe?

Our Answer: 24

Correct Answer: number 24

5.3 Overfitting

RNNs are often prone to overfitting, and despite adding dropout to all of our layers and between individual RNN cells we found that our model was still consistently overfitting with a 15 point difference between our performance on the training and dev sets (Figure 10). We intended to implement variational dropout as proposed by Gal and Ghahramani [7], but ended up not finishing it in time for the final model.

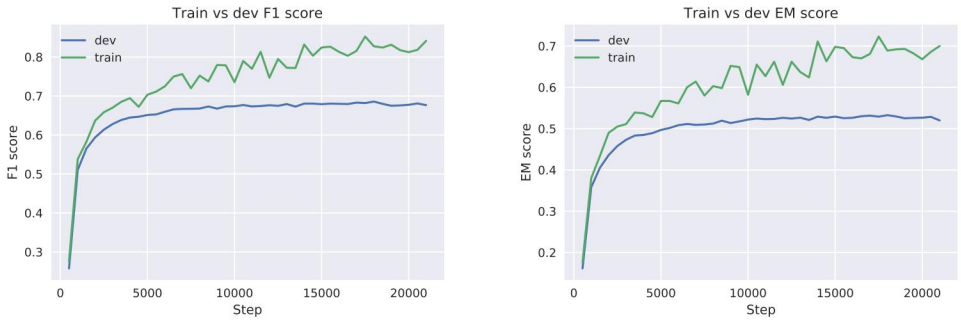

Figure 10: Context and question word length histograms

6 Conclusion and Future Work

We were partially successful in reimplementing the BiDAF architecture and in testing an additional self-attention layer. Unfortunately, we were unable to achieve the results listed in the original papers. This could have been due to a suboptimal choice in hyperparameters or implementation details, as these details were elided in the original papers. It is also possible that parts of our implementation were simply incorrect, as machine learning models tend to be difficult to debug in practice.

We would also like to make better use of the training set, as we only use one of the potential answers for each question. The training dataset actually includes multiple human-sourced answers for each question, so we could've easily greatly increased the size of our training dataset.

6.1 Acknowledgements

I would like to thank the entire CS224n staff for their hard work and dedication to the course. I would also like to thank Microsoft for providing Azure credits so that I was able to train my models in a timely manner.

References

[1] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016. URL `http://arxiv.org/abs/1611.01603`.

[2] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 189–198, 2017. doi: 10.18653/v1/P17-1018. URL `https://doi.org/10.18653/v1/P17-1018`.

[3] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. URL `http://arxiv.org/abs/1606.05250`.

[4] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[5] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL `http://arxiv.org/abs/1505.00387`.

[6] Steven Bird. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

[7] Y. Gal and Z. Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. *ArXiv e-prints*, December 2015.