Answering Questions with CharCNN and Bi-directional Attention Flow

Cindy Jiang
cindyj@stanford.org

Connie Xiao
coxiao@stanford.org

Abstract

The Question Answering (QA) task posed by the SQuAD dataset involves answering a question by selecting a segment of the context passage. There are many models developed to perform this task, but only recently have some models been able to match the 86.8% F1 score achieved through human performance. In this paper, we implement components of the recent BiDAF model, which includes adding a character embedding layer, attention flow layer, modeling layer, and output layer on top of the baseline model.

1. Introduction

We implemented the model proposed by Seo *et al.* (2017) [7]. We added the following layers on top of the baseline implementation: a character-level convolutional neural network layer (CharCNN), Bidirectional Attention Flow (BiDAF) layer, modeling layer, and output layer. In addition to this implementation, we experimented with hyperparameters and tuned our model on the training and dev set while comparing performance to the provided baseline.

2. Dataset

The dataset used is the Stanford Question Answering Dataset (SQuAD). It is a reading comprehension dataset consisting of 100,000+ questions corresponding to context paragraphs, where the answer to the question is a segment of the context paragraph. The context paragraphs are from Wikipedia articles and the questions and answers were crowd-sourced using Amazon Mechanical Turk [4].

3. Model

The model we implemented is based off of the Bi-directional Attention Flow model introduced in Seo *et al.* (2017) [7]. The multiple layers of the model are visualized in the diagram Figure 1, taken from the reference paper.

The baseline model implementation contained a word embedding layer, an RNN encoder layer, a basic attention layer, an output layer and the training layer.

3.1. Character Embedding Layer

We trained a convolutional neural network using one layer of convolution and max pooling to obtain character embeddings on the words from our input questions and contexts [7]. This is done by taking the characters in each word $c_1, ..., c_L$ for a word with L characters and representing those characters with trainable character embeddings $e_1, ..., e_L$. Those embeddings go though a CNN that computes hidden states based off a window of character embeddings $[e_{i-k}, ..., e_i, ..., e_{i+k}]$ for position i and window size k. Each of the $h_1, ..., h_L$ hidden states go through max pooling, which results in the character encoding of a word. Using a simple CNN for natural language processing tasks have been effective in sentence classification with little hyperparameter tuning [2]. Further studies have also demonstrated how character-level convolutional networks can achieve state-of-the-art results in text classification [11] and other NLP tasks like text generation [3] and part of speech tagging [6].

3.2. Word Embedding Layer

We used pretrained GloVe word embeddings from the baseline. These embeddings captured the meaning and context of words within a vector representation.

3.3. RNN Encoder Layer

The input to this layer is the concatenation of word embeddings with their corresponding character embeddings from the previous two layers. The input vectors are fed into a bi-directional GRU, which outputs forward hidden states and backward hidden states for both context and query words. This captures the contextual relationships between the context words, query words and the nearby words within the input passages.

3.4. Attention Flow Layer

The hidden states generated from the RNN encoder layer are used to calculate attention. Attention is an effective mechanism to improve performance, allowing query words to focus on certain context words more and vice versa. The baseline implemented a basic dot-product attention layer with context-to-query (C2Q) attention, which gives weights

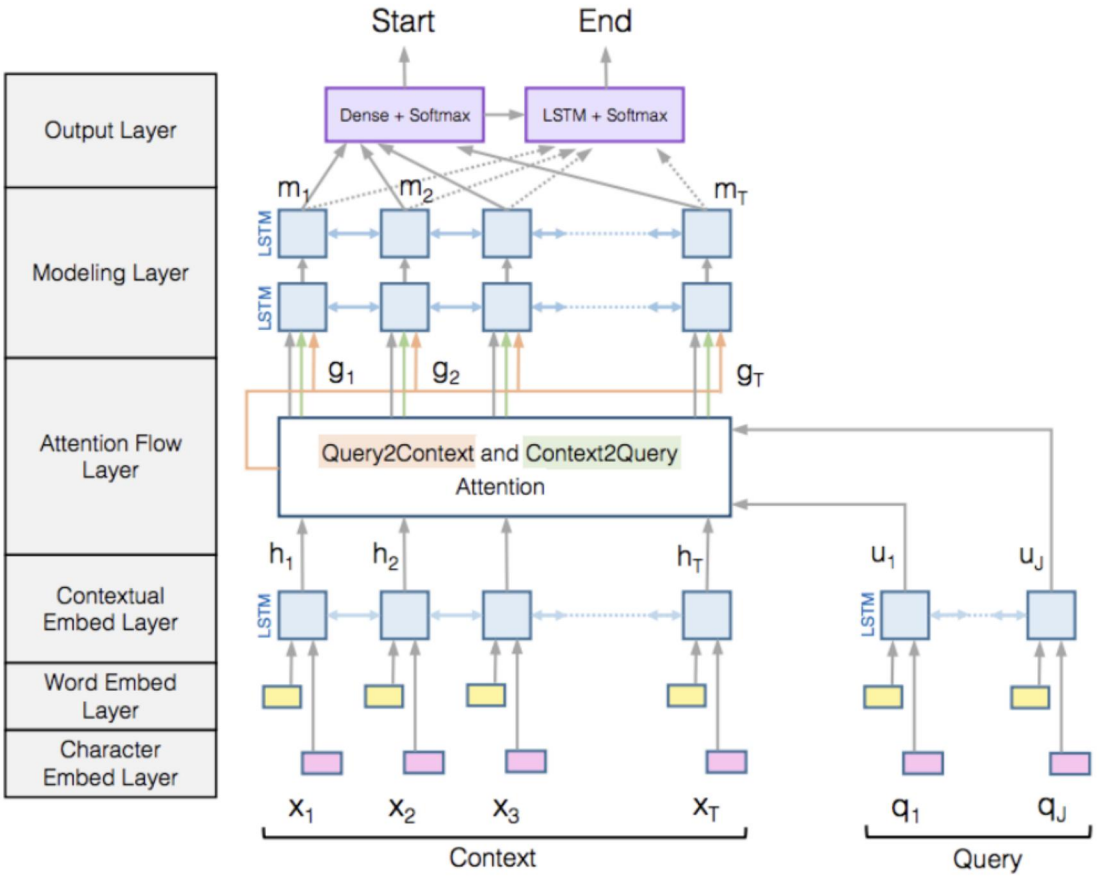

Figure 1. Bi-directional Attention Flow model

to how relevant query words are to context words. In addition, bi-directional attention flow considers query-to-context (Q2C) attention, which gives weights to how relevant context words are to query words. The implementation is based off of Seo *et al.*'s BiDAF model [7]. At every time step, we compute these two directions of attention with the similarity matrix $\boldsymbol{S} \in \mathbb{R}^{N \times M}$ found by taking the product of every question and context word by pairing and multiplying the whole matrix by a weight vector $\boldsymbol{w}_{sim} \in \mathbb{R}^{6d}$. The output is not just fixed vector representations of the query and context; rather, the output is the query-aware representations of context words. This reduces the informational bottleneck of trying to encode the full query and context into a fixed representation. The output of this layer flows into the modeling layer as an input.

3.5. Modeling Layer

The modeling layer receives the query-aware representations of context words from the modeling layer and outputs a matrix $\boldsymbol{M} \in \mathbb{R}^{2d \times T}$. The modeling layer consists of two stacked layers of bi-directional RNNs. We used bi-directional GRUs rather than the bi-directional LSTMs used by Seo et al. (2017). The output matrix $\boldsymbol{M}$ represents the context of each word with respect to the whole context passage and query. The output to this layer flows into the next layer as an input.

3.6. Output Layer

The output layer receives the contextual representations of words $\boldsymbol{M}$ with respect to the context passage and query and outputs the probability distributions for both the start and the end indices. First, we find the softmax probability distribution of the start index over the context passage. Then $\boldsymbol{M}$ is fed into another bi-directional RNN in order to find the softmax probability distribution of the end index.

3.7. Training Layer

The algorithm used to train the our model is summing the cross-entropy loss and optimizing with the Adam optimizer, an extension of stochastic gradient descent. The Adam optimization algorithm adds momentum to the standard SGD to speed up the learning rate and dampen oscillations [5].

The model then finds the output that minimizes the loss to return as the answer.

3.8. Model Tuning

In addition to modifying the model implementation, we tested with different hyperparameters as well. This includes batch size, hidden size of the RNN, maximum lengths based off of the input data, and the type of complex RNN used.

3.8.1 Lengths

On the development set of our dataset, we generated histograms for context length, question length, and word length. Informed from the histograms, we set the hyperparameter for the maximums of context length to 300, for question length to 25, and word length (when using Char-CNN) to 15. This decreased training time.

The context length was chosen jointly because few contexts had beyond 300 words 2 and less than 90 percent of answers are located after the 300 word mark 3. The maximum question length and answer lengths were chosen at a value that represents more than 90 percent of the dev dataset.

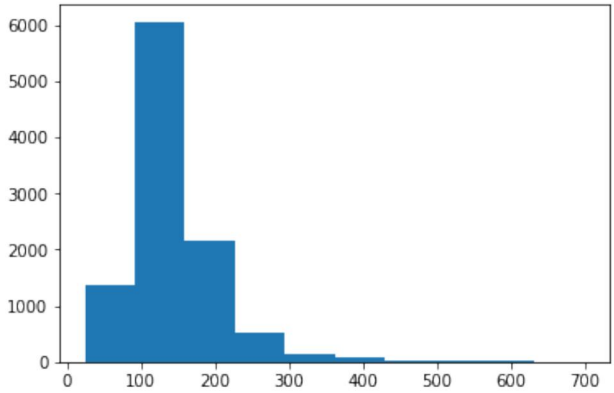

Figure 2. Context Lengths

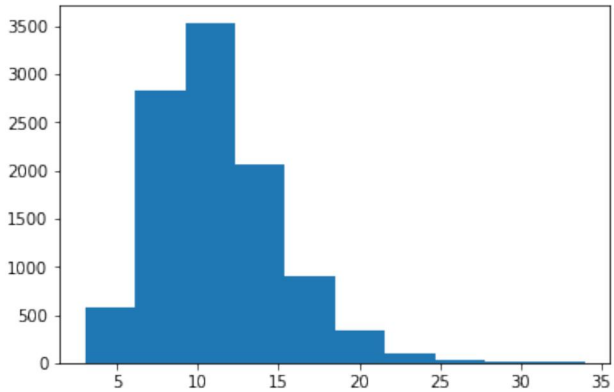

Figure 3. Question Lengths

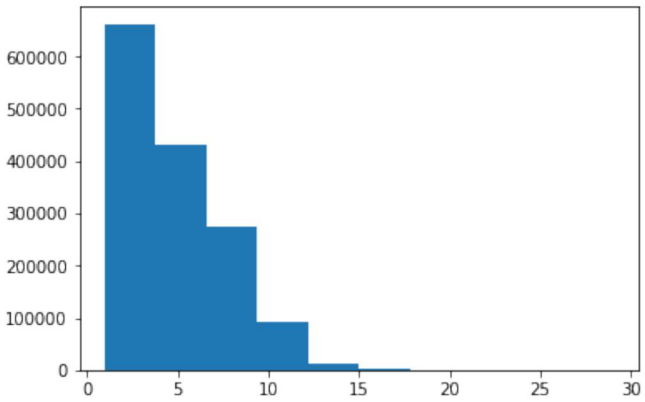

Figure 4. Word Lengths

3.8.2 Batch Size and Hidden Size

The baseline parameters include batch size 100 and hidden size 200. The BiDAF implementation by Seo *et al.* uses batch size 60 and hidden size 100. However, running with these parameters significantly decreased performance compared with the baseline parameters.

In addition, we tried to increase batch size to 150 and 120 to increase performance. Although this was feasible using the baseline model, there was not enough memory for us to increase batch size for our final model.

3.8.3 Type of RNN

The baseline uses a bi-directional GRU to encode the word embeddings. We ran it with a bi-directional LSTM as well, but there was minimal difference with the GRU performing slightly better as seen in Figure 5. Therefore, we chose to keep the baseline GRU.

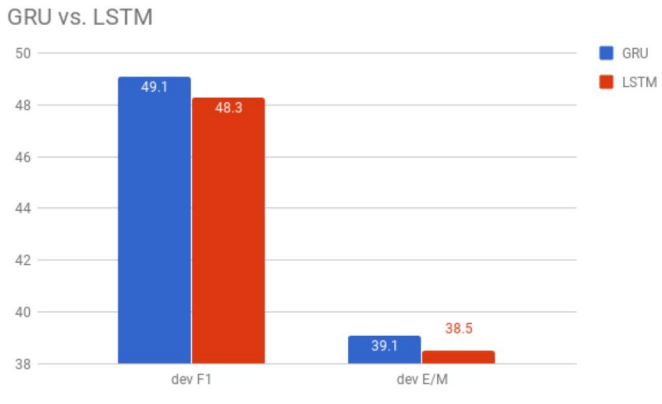

Figure 5. Comparison of GRU and LSTM

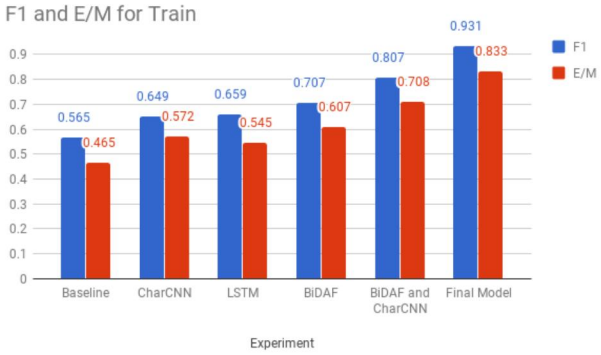

Figure 6. Training Scores

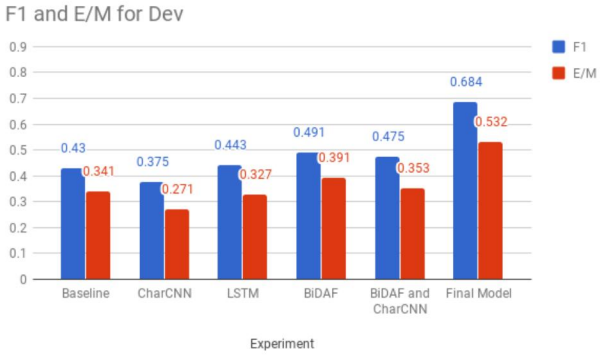

Figure 7. Dev Scores

Model	dev F1	dev EM
Baseline	42.959	34.059
CharCNN	42.887	33.595
BiDAF	49.117	39.073
CharCNN + BiDAF	51.572	41.580
BiDAF + Modeling + Output	72.960	62.999
CharCNN + BiDAF + Modeling + Output	73.976	63.841

Table 1. F1/EM scores for various models run on dev set

4. Experiments

In this section, we describe the experiments that were run during the development of the model. The two evaluation metrics being used are the F1 and Exact Match (EM) scores. Exact Match requires the returned answer to be exactly the same as the ground truth. F1 is less strict and takes the harmonic mean of the precision and recall, where precision is the ratio of words correct within the outputted answer and recall is the ratio of words correct compared to the ground truth.

Figure 6 shows the F1/EM scores for our models on the training set. Figure 7 and Table 1 shows the F1/EM scores for our models when run on the dev set.

After evaluating dev scores locally, we submitted our final model (CharCNN + BiDAF + Modeling + Output) to the CS224N dev leaderboard and test leaderboard. The scores are displayed in Table 2.

Evaluation	dev F1	dev EM
dev (local)	73.976	63.841
dev(leaderboard)	74.336	64.305
test (leaderboard)	75.158	65.184

Table 2. F1/EM scores of final model evaluated on various data and sources

5. Evaluation

The BiDAF model performs a lot better than the baseline implementation. Table 3 and Figure 8 shows the percent increase of each model compared to the baseline scores.

Model	%Δ dev F1	%Δ dev EM
CharCNN	-0.168	-1.361
BiDAF	14.333	14.722
CharCNN + BiDAF	20.049	22.083
BiDAF + Modeling + Output	69.833	84.972
CharCNN + BiDAF + Modeling + Output	72.199	87.444

Table 3. Percent change of F1/EM scores for dev set compared to the baseline

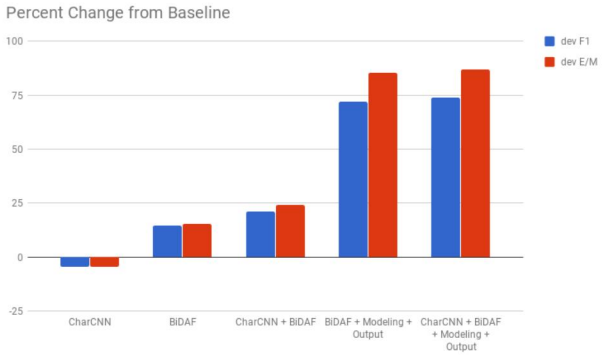

Figure 8. Percent change of F1/EM scores for dev set compared to the baseline

5.1. CharCNN Layer

When CharCNN is added to the baseline with no other changes, we see that although the training set produces

marginally better results, the dev set performs worse. This can be seen in the negative dev percent change in Table 3.

However, CharCNN boosts dev set performance when it is added to the modified implementations. With respect to the baseline, CharCNN increases F1 by 5.716% and EM by 7.361% when added to the BiDAF model. Similarly, CharCNN increases F1 by 2.366% and EM by 2.472% when added to the BiDAF + Modeling + Output model.

5.2. BiDAF Layer

Changing the basic attention to bi-directional attention flow improved performance more than CharCNN did. F1 increased by 14.333% and EM increased by 14.722%. Because of its large improvement, we decided to continue using BiDAF attention during the subsequent experiments.

5.3. Modeling and Output Layers

Adding the modeling and output layers showed the most significant improvement during our experiments since it adds three more layers of RNN to extract the answer from the context and query representations. This addition improved F1 by around 50-55% and EM by around 65-70%.

5.4. Sample Questions and Answers

Context: to remedy the causes of the fire , changes were made in the block ii spacecraft and operational procedures , the most important of which were use of a _nitrogen/oxygen_ mixture instead of pure oxygen before and during launch , and removal of flammable cabin and space suit materials . the block ii design already called for replacement of the block i _plug-type_ hatch cover with a quick-release , outward opening door . nasa discontinued the manned block i program , using the block i spacecraft only for unmanned saturn v flights . crew members would also exclusively wear modified , fire-resistant block ii space suits , and would be designated by the block ii titles , regardless of whether a lm was present on the flight or not .
Question: what type of materials inside the cabin were removed to help prevent more fire hazards in the future ?

Model	Answer
CharCNN	(end before start)
CharCNN + BiDAF	flammable cabin and space
BiDAF + Modeling + Output	space suit materials
CharCNN + BiDAF + Modeling + Output	flammable cabin and space suit materials
True Answer	flammable cabin and space suit materials

Table 4. Improvement in question responses

Context:there are infinitely many primes , as demonstrated by euclid around 300 bc . there is no known simple formula that separates prime numbers from composite numbers . however , the distribution of primes , that is to say , the statistical behaviour of primes in the large , can be modelled . the first result in that direction is the prime number theorem , proven at the end of the 19th century , which says that the probability that a given , randomly chosen number n is prime is inversely proportional to its number of digits , or to the logarithm of n .
Question:what theorem states that the probability that a number n is prime is inversely proportional to its logarithm ?

Model	Answer
CharCNN	the prime number theorem
CharCNN + BiDAF	n
BiDAF + Modeling + Output	direction
CharCNN + BiDAF + Modeling + Output	proven at the end of the 19th century
True Answer	the prime number theorem

Table 5. Unexpected changes in quality question responses

In Table 4, we can observe improvement qualitatively after successively adding features to our model. This is the predicted behavior, where additional features aid the model's ability to determine the answer. The lone addition of CharCNN to the baseline model proves to hinder the quality of answer. The model is not complex enough to handle the more detailed character level features and the consequential overfitting on the training data is indicated in the model selecting a faulty format of answer to this question. The addition of BiDAF improves the model's ability to relevantly focus on certain words. By additionally associating the query to context, BiDAF captures a subsequence of the answer. Using BiDAF, modeling, and output also suffers from imprecise answer boundaries. However, with all implemented features, our model was able to arrive at the correct answer.
At the same time, additional features can make the quality of answers worse as demonstrated in Table 5. Adding features to the model clouds the model as it selects answers that are related to syntactic ambiguities in the context paragraph and question.

6. Conclusion

We implemented the bi-directional attention flow model proposed by Seo *et al* (2017) [7]. Beyond the baseline implementation, we implemented additional layers. As input to the GRU along with word embeddings, we added a char-

acter CNN to represent words with character-level encodings. In addition, we added bi-directional attention flow so that attention flows both forwards and backwards, and gets inputted into the next layer. Finally, we stacked more GRU layers to contextually determine the probability distributions of the start and end indices for the predicted answer. Together, these modifications significantly improved performance from the baseline. Our final dev scores were 63.8% EM and 84.0% F1. However, there is still much room for improvement compared to the state-of-the-art models at the top of the SQuAD leaderboard (Top EM: 82.849, Top F1: 89.281) and compared to human performance (EM: 82.304, F1: 91.221).

6.1. Future Work

6.1.1 Hyperparameters

For CharCNN, we used the default values provided. With additional time, we would tune the character embeddings size, hidden size, and window size. We also considered experimenting with different training optimizers, such as Adadelta or RMSProp, as described by Ruder [5].

6.1.2 Answer Pointer

Our model selected the start and end positions of the answer by taking the largest value in each respective probability distribution. In a number of cases, this meant selecting an end word that preceded the start word. If we prevented our model from choosing an end position of an answer prior to the start of the answer, our scores may improve. Further work on this idea would be to to determine the end prediction with knowledge of the start prediction. This is implemented by Wang *et al.* (2017) [8].

6.1.3 Error Analysis

We could better understand the faults of our model through more thorough error analysis in our final model. We could classify the errors our final model made into categories and adjust our model to fit the type of error. We could also utilize error categories that have been defined by other works such as by Seo *et al.* (2017) [7].

6.1.4 Ensemble

In addition to BiDAF, there are plenty of other models developed to tackle the Question Answering problem. Many of the models at the top of the SQuAD leaderboard are ensemble models, which combines multiple models together. This is more expensive to run but it boosts performance significantly since there are multiple evaluation algorithms being checked against each other.

References

[1] Y. Cui, Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu. Attention-over-attention neural networks for reading comprehension. *arXiv preprint arXiv:1607.04423*, 2016.

[2] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014. 1

[3] Y. Kim, Y. Jernite, D. Sontag, and A. M. Rush. Character-aware neural language models. In *AAAI*, pages 2741–2749, 2016. 1

[4] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016. 1

[5] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. 2, 6

[6] C. D. Santos and B. Zadrozny. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826, 2014. 1

[7] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016. 1, 2, 5, 6

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

[9] W. Wang, N. Yang, F. Wei, B. Chang, and M. Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 189–198, 2017.

[10] C. Xiong, V. Zhong, and R. Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*, 2016.

[11] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015. 1