Improved QA systems for SQuAD 2.0

Stanford CS224N Default Project IID SQuAD Track

Pablo Diaz
Department of Chemical Engineering
Stanford University
pablo98@stanford.edu

Chloe He
Department of Biomedical Informatics
Stanford University
chloehe@stanford.edu

Akshay Nalla
Department of Aeronautics and Astronautics
Stanford University
analla@stanford.edu

Abstract

Question answering (QA) is a cornerstone in the NLP domain that has produced innovations that span many other tasks. We developed a question answering system with improved performance compared to the baseline Bi-Directional Attention Flow (BiDAF) starter model provided to us. We implemented two versions of character embeddings, an answer pointer decoder, and a self-attention layer into the original baseline model. We then evaluated the effects of each of these implementations. Our best model, which contained all three layers, produced an EM score of 59.83 and F1 score of 63.37 on the test sets, while also showing a significant increase in the model learning speed. We anticipate that this work will aid in the continuing development of efficient question answering systems.

1 Introduction

Question answering (QA) has been an active research area within the field of deep learning over the past years. One of the models that has pushed the field of question answering is the Bi-Directional Attention Flow (BiDAF), which is a multi-stage hierarchical network that represents context as different types of embeddings and uses bi-directional attention flow to obtain a query-aware context representation. Although the performance of this model has been surpassed, this model remains influential in the domain of QA.

BiDAF is a closed-domain, extractive question-answering model. In order to answer a query, BiDAF must have access to a context text that contains information to answer the query. BiDAF contains three main parts: embedding layers, attention and modeling layers, and an output layer. The first layers map representation of words from strings into high-dimensional vectors of numbers. The second layers transforms these number-based vectors into "query-aware context representation". Finally, the output layer transforms these representations into a probability vector that determines the answer. A graphic of the BiDAF architecture is shown in Figure 1.

The goal of this project is to train a Bidirectional Attention Flow (BiDAF) neural network model that performs better than the baseline model. We are interested specifically in investigating the use of character-level embeddings, conditioning end prediction on start prediction, and self-attention. We chose this set of techniques because they frequently appear in high-performing machine comprehension systems. Self-attention in particular has been incredibly popular in recent years and proven to be one of the foundations for transformers. Our secondary goal is to explore the hyperparameter search space and experiment with other types of word embeddings as well as ensembling methods. Hyperparameter tuning includes regularization, optimization algorithms, and changing learning rate and batch size.

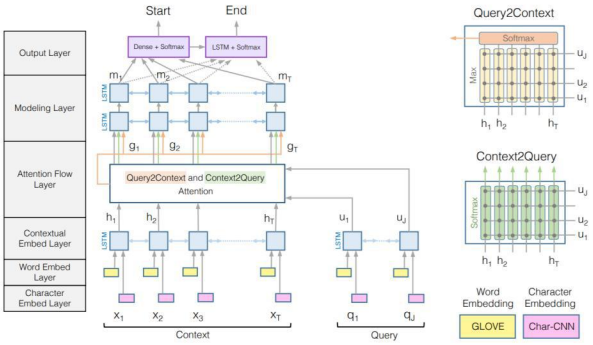

Figure 1: Bi-Directional Attention Flow model architecture [1].

2 Related Work

2.1 BiDAF

Seo et al. introduced the Bi-directional Attention Flow for Machine Comprehension model, otherwise known as BiDAF, in a paper published in 2018 [1]. The authors evaluated BiDAF's question-answering capability using the SQuAD dataset and cloze-style reading comprehension through the CNN and Daily Mail datasets. At the time of its publication, BiDAF outperformed traditional baselines and bi-directional attention flow proved to be a valuable tool in improving NLP model performance.

The BiDAF model consists of six layers: character embedding layer, word embedding layer, contextual embedding layer, attention flow layer, modeling layer, and output layer. In subsequent work, researchers explored alternatives to replace or add to many of the original components, attempting to further improve the performance of the state-of-the-art model.

The implementation of BiDAF was intended to improve machine comprehension (MC) for query answering. While the BiDAF model proved to have much promise in the field of MC, its value expanded past the initial domain. Further research was done to incorporate bi-direction attention flow into models for tasks such as image-text matching [2]. The BiDAF model could prove to be pivotal to incorporating MC into research areas such as computer vision.

2.2 Character Embeddings

While the original BiDAF model included a character-level embedding layer in addition to word embedding and contextual embedding, the character embedding was excluded in the implementation of the baseline model. Character-level embeddings are based on one-dimensional convolutional neural networks (1D-CNN) that generate embeddings based on character-level compositions [3]. The advantage of 1D-CNN is that it can extract information on a subword level from shorter segments of input sequences. This approach allows the model to condition on the internal structure of words, and is useful in case we find words that are outside of the training vocabulary. Character-level embeddings have been shown to improve performance in NLP models. Since the baseline model excluded character-level embeddings from the BiDAF model, a natural extension would be to implement the character embeddings into our model.

2.3 Answer Pointer

In their "Machine Comprehension Using Match-LSTM and Answer Pointer" paper, Wang and Jiang proposed two answer pointer models that were used in place of the typical decoder layer as a final layer, which convert the hidden states from the previous layer to probabilities as outputs [4]. In the original BiDAF model, the output layer uses a combination of attention outputs and modeling outputs to compute probability distributions of the start index and the end index over the entire context, separately. The Boundary Answer Pointer model provides an alternative, conditioning the probability

for the end location on the distribution of the start location. This could be an advantageous approach specifically in the domain of extractive question answering.

2.4 Self-Attention

Since the "Attention is All You Need" paper was published in 2017, the attention mechanism attracted considerable attention for use in various domains. Self-attention is a variation of it and was proposed by Wang et al. in 2017 in their R-NET model [5]. Self-attention allows a hidden state in a certain timestep to attend the previous hidden layers. This means that the inputs are interacting with each other in order to calculate attention scores of certain inputs with respect to others in different parts of the entire passage. This allows the model to choose which parts of the passage to pay more attention to, which is useful especially when dealing with long sequences of contexts.

3 Approach

We began by implementing a character embedding layer in addition to the word embedding layer, such that the character embeddings and the word embeddings were concatenated and fed together to the highway network in the last step of the embedding layer. Next, we added an answer pointer decoder in place of the original output layer and a self-attention layer immediately after the attention flow layer in the original BiDAF model. In the following sections, we describe each of the three components in detail.

3.1 Character embedding

We implemented two versions of the character embedding layers. The first version involves passing the pretrained character embedding through 2D convolutions followed by ReLU activation and learning a feature map for every given filter width. Then, for each filter of the given filter width, we used a maxpooling operation over all the outputs of that filter. We now have various feature matrices taken from each convolution that we concatenate to be the character matrices for the input word. We repeat this process for all the words in the sequence, learning an character embedding for every word. The number of convolutions will be determined by the maximum filter width and the number of output channels, both of which are hyperparameters. The first convolution starts with a kernel width of one. The next convolution will have a kernel width of two, and so on. The last convolution will be the one that has the maximum kernel width. We used 25 to 100 filters for every filter width. The kernel height is the embedding dimension and is the same for all convolutions.

The second version involves passing the embedding through three 1D convolutional layers. The difference is that the output of one convolution is the input to the next convolution. The kernel size gets smaller as the input gets deeper into the network. The kernel size starts at 7, then 5 and finally 3. After each convolution, a ReLU function is applied to the embedding followed by a dropout layer with a 20% probability. We also implemented 1D batch normalization layers after each dropout layer. A feature matrix is then created by taking the maximum values of the embedding through the width. At the end of this network, the output will be the concatenation of the three feature matrices.

3.2 Answer pointer

Our implementation of the answer pointer takes the set of representations $m_1, ..., m_N \in \mathbb{R}^{2H}$ produced by the previous modeling layer and runs for two timesteps. At each timestep, the layer learns an attention weight vector β_k through the formulas below, which is used as the probability distribution for the start or end location.

$$F_k = \text{tanh}(VM + (W^a h_{k-1}^a + b^a) \otimes e_N) \tag{1}$$

$$\beta_k = \text{softmax}(v^T F_k + c \otimes e_N) \tag{2}$$

where N is length of the context and $M \in \mathbb{R}^{N \times 2H}$ is a matrix representation of the representations $m_1, ..., m_n$. $\cdot \otimes e_N$ repeats the scalar or vector N times to produce a vector or matrix with the appropriate dimensions. $V \in \mathbb{R}^{2H \times H}$, $W^a \in \mathbb{R}^{H \times H}$, $b^a \in \mathbb{R}^H$, $v \in \mathbb{R}^H$, and $c \in \mathbb{R}$ are all learnable parameters.

The probability distribution of the second time step is dependent on that of the first time step through both β_k and the hidden and cell states of the LSTM, such that

$$h_k^a = \overrightarrow{\text{LSTM}}(M\beta_k^T, h_{k-1}^a) \tag{3}$$

Lastly, we model the probability of generating an answer as

$$p(a_s|M)p(a_e|a_s, M) = \beta_s \cdot \beta_e \tag{4}$$

such that β_s and β_e are the probability distributions (β_k above for each of the two timesteps) for the start and end locations, respectively. The loss function is the same:

$$\text{loss} = -\log p(a_s|M) - \log p(a_e|a_s, M) \tag{5}$$

3.3 Scaled Dot-Product Self-attention

We implemented self-attention after the bidirectional attention flow layer. The self-attention layer takes the bidirectional attention output as input and uses key, query, and value weight matrices to calculate attention scores. The key and query weight matrices are initialized with a shape of `hidden size` $\times p$, where p was a hyperparameter set to be 284. The value weight matrix is of shape `hidden size` $\times$ `hidden size`. The input attention score is multiplied by these weights to produce key, query, and value matrices. Attention scores are obtained for each input from the query and key matrices, which are then put through a softmax function.

$$\text{Attention Scores} = \text{softmax}(QK^T) \tag{6}$$

These attention scores are multiplied by the value matrix to produce a final attention output.

$$\text{Output} = \text{softmax}(QK^T)V \tag{7}$$

The attention outputs are passed through ReLU activation layer before being passed further down the model to the answer pointer layer.

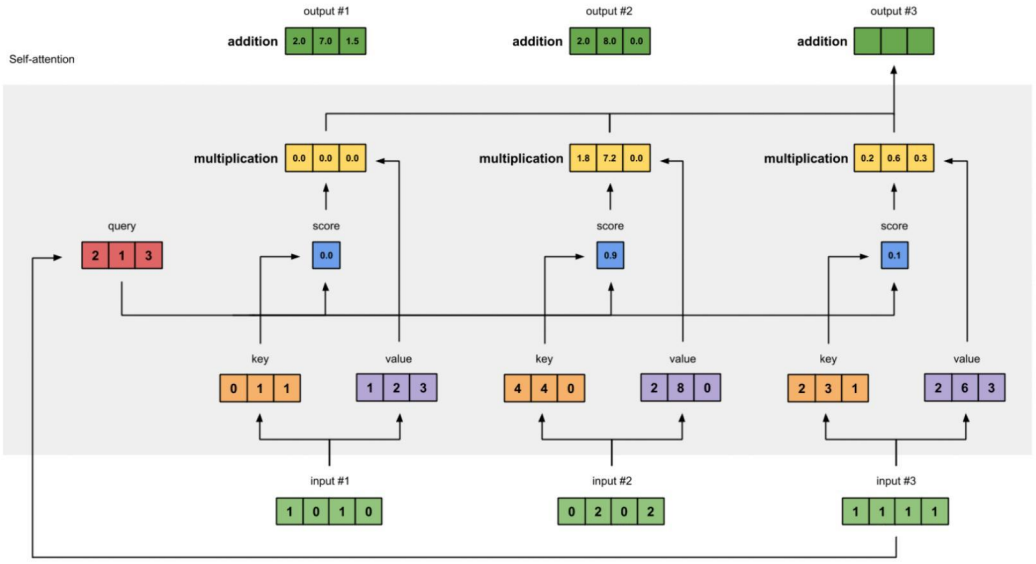

Figure 2: Self-attention calculation example [6]

4 Experiments

4.1 Data

We used the Stanford Question Answering Dataset (SQuAD) 2.0 [7]. SQuAD 2.0 contains 150k questions crowd-sourced on a database of Wikipedia articles. Among these 150k questions, 100k

are answerable questions, whereas the rest do not have answers in the contexts provided. The 50k unanswerable questions were written to look similar to the answerable questions. They are relevant to the given paragraphs, which contain plausible answers or similar wording, but not actually an answer to the question. This makes the question answering task much more challenging - in addition to finding the relevant piece of context information, the model must also determine if the question is answerable by any of the given contexts. This also means that, by always predicting no-answer, a model can already achieve ∼50% EM/F1 very early in training.

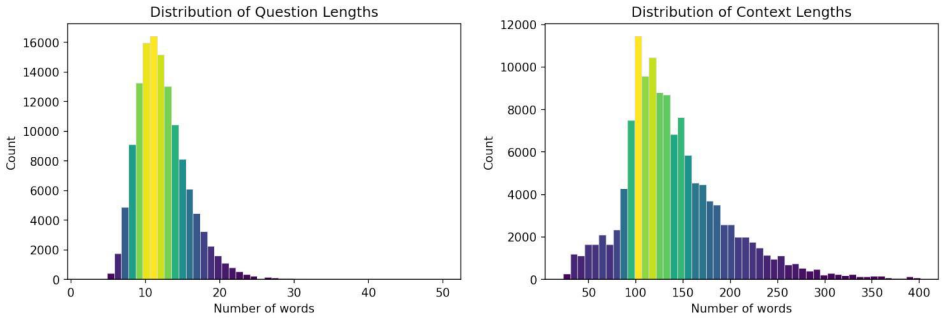

Figure 3: Distributions of question lengths and context lengths in the training dataset

4.2 Evaluation method

We used Exact Match (EM) score and F1 score for evaluation, with EM being a binary measure strictly counting the number of exact matches, and F1 being a less strict metric measuring the harmonic mean of precision and recall. For any given question, we will score the generated output against all three human-provided answers and take the highest EM and F1 scores.

4.3 Experimental details

By adding in each of the three components individually and in combinations, we conducted experiments that mimic an ablation study. Specifically, we ran experiments with each of the two aforementioned character embeddings, answer pointer, and self-attention separately, and also experiments with all three implementions.

We experimented with several different hyperparameters such as learning rate, training time, and batch size. Ultimately, we settled upon a learning rate of 0.5 and batch size of 32, which consistently produced the best results in our models. We trained each model for up to 30 epochs, but in the interest of time and efficient model tuning, once the model's EM and F1 scores started to plateau or decrease and the loss suggested overfitting, we terminated training sessions.

4.4 Results

The resulting EM and F1 metrics for each model iteration can be found in Table 1. While the original implementation of character embeddings did not result in a notable change from the baseline, the second iteration of the character embeddings led to a significant improvement. When implemented separately, both the self-attention and answer pointer model provided improved results as well. As expected, combining all three implementations (using the second version of the character embeddings) produced the best results for both metrics. We also added batch normalization layers after every convolutional layer in the character embedding layer to help stabilize training. This addition produced mixed results since, although both EM and F1 scores increased on the validation set, these scores decreased on the test set. Visualizations of EM and F1 scores on the validation set over training can be seen in Fig. 4.

We also made some unexpected observations regarding our results. In particular, the implementations of self-attention and answer pointer did not improve the model performance as much as we expected, based on performance outcomes put forth by the respective papers. Our first version of character embeddings, though more complex than the second version and actually more similar to the original

Table 1: Model Results

Model	Test EM	Test F1	Dev EM	Dev F1
Baseline	-	-	56.98	60.28
Character embeddings V1	56.89	60.84	56.97	60.55
Character embeddings V2	-	-	60.04	63.39
Self-Attention	-	-	59.43	62.7
Answer Pointer	-	-	57.56	61.09
All implementations	**59.83**	**63.37**	60.58	63.77
All implementations with batch normalization	58.99	62.56	**61.57**	**64.83**

implementation in the BiDAF paper, produced negligible performance improvements. With many of the experiments that we ran, the improved models learned much faster (the curve was steeper at the beginning), but the scores also started plateauing earlier on in training. Lastly, among all the models we tested, the self-attention model seemed to be the only one that did not start overfitting to the training set. Its negative log likelihood (NLL) on the dev set continued to decline or stayed mostly flat even after 1.5M iterations, which is where all the other models started overfitting and had rising NLLs.

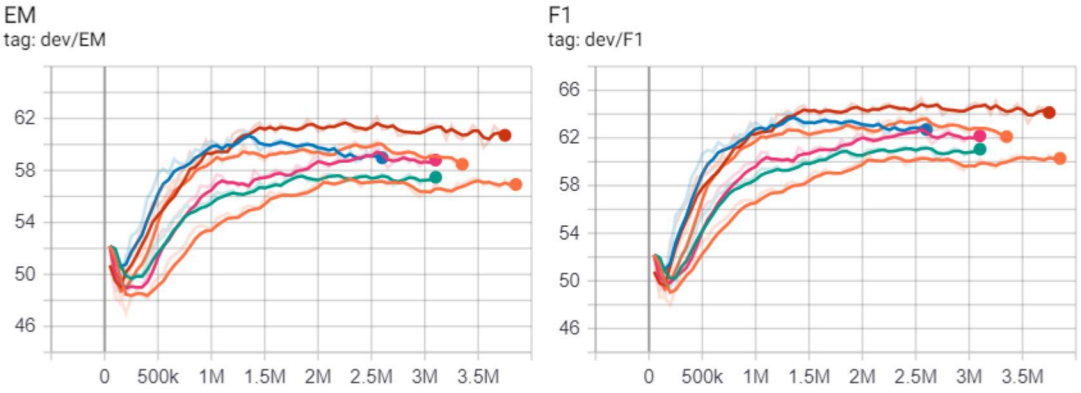

Figure 4: EM and F1 metrics for each model: baseline (orange - lower), character embeddings V2 (orange - upper), self-attention (pink), answer pointer (green), combined model (blue), combined model with batch normalization (red)

5 Analysis

We tried to better analyze and understand the successes and weaknesses of our models by inspecting the predicted outputs of each model.

We saw that the no-answer questions were challenging for our models. These questions are well-constructed to have very similar phrasing and mentions of key words frequently repeated in the context. For that reason, they were able to sometimes fool the model into producing answers, even though the questions are actually unanswerable.

By analyzing the text outputs of our models in Tensorboard, we hoped to get a better understanding about our model's question answering capabilities. Unfortunately, there did not seem to be a consistent pattern in which a single model performed better on a specific topic compared to other models. It was interesting to note that our final model correctly answered questions about about Scottish and Roman languages which the baseline could not. However, the sample size was too small to draw a meaningful conclusion from.

Overall, the model performed equally well for questions and contexts of various lengths, as seen in Fig. 5.

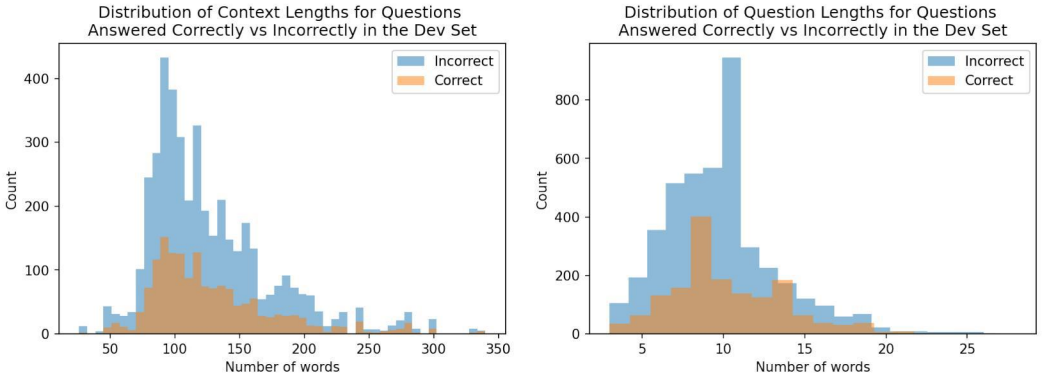

Figure 5: Context length and answer length distributions for questions answered incorrectly (blue) or correctly (orange) in the dev set

6 Conclusion

In this project, we developed a question-answering system using the BiDAF model as a backbone. We implemented two types of character-level embeddings, a self-attention layer immediately after the bidirectional attention flow layer, and an answer pointer decoder motivated by the Pointer Network to replace the original output layer. With the addition of each of these three components, we saw improvements in EM and F1 scores. Our best-performing model was a model incorporating all three layers, achieving an EM score of 59.83 and an F1 score of 63.37 on the test set. In further analysis of the predictions, we found that the well-constructed no-answer questions were challenging for our model, but that the model performed equally well across questions and contexts of different lengths.

Potential avenues for future work could be to explore additional hyperparameter tuning such as dropout rate, a wider range of learning rates, number of convolutional layers, kernel size and number of output channels. In addition, an alteration to the self attention model would be to include a scaling factor before the softmax layer. This scaling factor would help prevent the gradients of the softmax function from becoming extremely small [8].

7 Contributions

Chloe: Implemented both versions of character embeddings and answer pointer and worked on the final report.

Pablo: Contributed to the second version of the character embeddings, prepared the first virtual machine environment, performed initial tests of the baseline in the virtual machine, analyzed the performance of predictions versus answers and worked on the final report.

Akshay: Worked on the initial character embeddings, self-attention implementation, test output submissions and the final report.

Every member contributed equally to data analysis.

References

[1] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.

[2] Yiling Wu, Shuhui Wang, Guoli Song, and Qingming Huang. Learning fragment self-attention embeddings for image-text matching. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, page 2088–2096, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Yoon Kim. Convolutional neural networks for sentence classification, 2014.

[4] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer, 2016.

[5] W Wang, N Yang, F Wei, B Chang, and M Zhou. R-net: Machine reading comprehension with self-matching networks. *Natural Lang. Comput. Group, Microsoft Res. Asia, Beijing, China, Tech. Rep*, 5, 2017.

[6] Raimi Karim. Illustrated: Self-attention. *https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a*, 2019.

[7] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

[8] A Vaswani, N Shazeer, and et al. Attention is all you need. *31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA*, 2017.