SQuAD 2.0 with BiDAF++ and QANet

Stanford CS224N Default Project

Xiaoteng Lu
Department of Electrical Engineering
Stanford University
luxt@stanford.edu

Yiwen Jin
Department of Electrical Engineering
Stanford University
yiwenjin@stanford.edu

Abstract

In this project, we produced a question answering system on SQuAD 2.0. To enhance the task performance, we explored two kinds of models. One is baseline BiDAF [1] model. We modified the baseline by adding character embeddings and implementing Co-Attention layers. We conducted the experiments thoroughly to evaluate the effects of each component. The other is QANet[2], which is a Transformer-based model, only including convolutional and self-attention layers and free of RNN component. We implemented the model from scratch and got some results in experiments. We found our best result is from the BiDAF-related model and achieved **F1 score 64.96, EM score 61.70** in validation set and **F1 score 64.712, EM score 60.997** in test set.

1 Key Information to include

- Mentor: None.
- External Collaborators (if you have any): None.
- Sharing Project: None.
- Late Day Usage: Used 6 late days from Xiaoteng Lu, equal to 3 late days for the group.

2 Introduction

The goal of our project is to produce a question answering system that works well on SQuAD 2.0. This is one of the default projects for the course and the task is that given a paragraph and a question about it as input, the model needs to answer the question correctly. It is interesting and meaningful because it provides an efficient approach for us to evaluate how well the model can really "understand" the paragraphs by seeing how accurately the model can answer the question given a target paragraph. As we known, "What", "How" and "Why" are all basic questions that we always ask to test if the students really understand the materials they learned. We used the queries and the corresponding ground true answer to train model and teach it how to understand the given paragraph, which is imitating the learning curve of the person by simply answering the questions and get responses.

This task has gain a lot advancement of using neural attention mechanism, which enables the model to focus on some target part. However, for the previous attention models, they are usually uni-directional, where the query attends on the context. The work [1] came up with a new attention flow mechanism to improve the performance of the model on question answering task. The author introduces the Bi-Directional Attention Flow network, which is a hierarchical multi-stage architecture for modeling context and query together. And this work is our baseline. In our code base, there is no character embeddings implementation. Therefore, we modified our baseline by adding character embeddings and substituting the attention layer with Co-Attention layer to see whether the new model can obtain higher scores.

We also implemented a new model without any RNN structure. We adopted the structure from paper [3]. In this work, the author proposes to remove the recurrent nature of the previous models and

use convolutions and self-attentions as building blocks of encoder. We can say this is a Transformer-based model instead of LSTM-related. The motivation behind the model's encoder is that convolution captures the local structure of the text, while self-attention learns the global interaction between each pair of words. We implemented the QANet from scratch and show the result in section 5.

In summary, we modified the baseline model by implementing character embeddings and Co-Attention layers and built a new Transformer-base model QANet from scratch. The details of the approaches are elaborated in Section 4. The details of experiment and performance scores are shown in Section 5. The error analysis part is in Section 6 and we draw a conclusion in Section 7.

3 Related Work

Reading comprehension and question answering has become an important topic in the NLP domain. There are a great number of papers proposed to tackle this task. We mainly explore two kinds of models. One is RNN-based model and the other is Transformer-based model with full convolution or full attention architectures.

There are several different approaches to perform attention mechanisms in the area of machine comprehension. The first approach uses dynamics attention mechanism. The work [4] provides large scale supervised reading comprehension data and argues that dynamic attention mechanism allows the model to continuously accumulate information form the context. This approach will take over lots of space. Instead, the second way proposed in the paper [5] introduces the "attended attention" for final answer predictions, which allows the model to compute the attention weights only once. Although it saves space and achieves good performance, it still get some limitations since the attention cannot flow into the LSTM layers. The third approach is to compute the attention vector between the context and the query repeatedly, which is also called "multi-hop" [6, 7]. Not like the approaches above, the BiDAF model [1] not only uses a memory-efficient mechanism but also allows the attention to flows into the LSTM layer.

More recently, although Long Short Term Memory architectures [8] has shown its powerful ability in NLP tasks, it is still not clear whether such method can tackle complicated tasks, like question answering. Since in the question answering tasks, the context may be a long text with hundreds of words. Therefore, there are some works have been made to use full convolution and self-attention layers instead of RNN-related component [9, 10, 11]. Those models have shown the result that is not only faster than RNN architectures, but also effective in many other tasks. And QANet [3] is the first work to achieve both fast and accurate reading comprehension model by using non-RNN architectures. This work is also the first to mix self-attention and convolution layers and proves to be quite effective and achieves a high performance score.

4 Approach

As stated in the default project handout, the baseline model BiDAF [1] has five layers in total. Each layer has its own function to do reading comprehension. For the embedding layer, it only implements the word embedding without the character embeddings. There are two optimization approaches we implemented so far: character embeddings [1] and co-attention layer [12].

4.1 Character Embeddings

When implementing word embedding, we treated words as the smallest component and try to find out the rules of word combinations. However, there are some prefixes, roots, suffixes with specific meanings used to compose words. It is natural to think about decomposing words into sub-words, which is character-level embedding. It allows us to take a look at the internal structure of the words. It has been mentioned in the original BiDAF model but not been included in the code base. First, we embedded each character in the word and use a 1D convolutional layer with d filters and window size of 3 to encode the character embedding matrix; then, we took a max-pooling for each channel. In this way, we could obtain a vector with length of d and concatenate it with output from the word embedding layer to get a new input vector for the encode layer. The figure 1 demonstrates how the character-level embedding works.

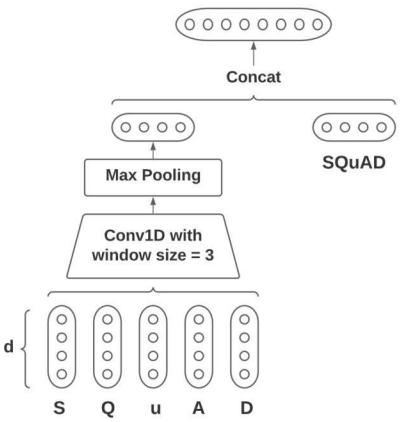

Figure 1: Character Embedding

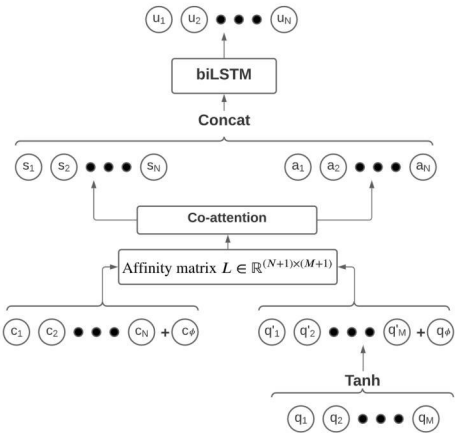

Figure 2: Mechanism of Co-Attention

4.2 Co-Attention

For the co-attention mechanism, we simply substituted the original attention layer with the co-attention layer. As shown in the figure 2, we added sentinel vectors c_ϕ and q_ϕ to both context and question states and then calculated out the affinity matrix for computing attentions output for both directions. To compute the attention, we used the basic way to first get the softmax weight and obtained the weighted sum for each row or column. Then we concatenated both Context-to-Question Attention and Question-to-Context Attention as the input of a biLSTM layer and produced overall output of the Co-Attention layer. The equations and the detailed descriptions are provided in the handout.

4.3 QANet

QANet is a Transformer-based reading comprehension model that built upon convolutions and self-attentions. For model's encoder, convolution captures the local structure of the text, while self-attention learns the global interaction between each pair of words. We implemented QANet from scratch based on the paper [3] and wrote all the code by ourselves except a small module, which is already cited in our code base. In the following several sub-sections, we will introduce the QANet by the order of layers and describe more implementation details than the original paper. The overall structure of QANet is in Figure 3.

4.3.1 Input Embedding Layer

Similar to the character embedding technique we used when modified the baseline, we used both word embeddings and character embeddings in QANet to better evaluate the internal structure of the words. Instead of just setting character embedding dimension to 200, which is the parameter in the original paper [3], we also tried using the embedding dimension directly from the character embedding vectors dimension, which is 64 in the data preprocessing section in the code base. Then, we concatenated both results to get the output tensor $[x_w; x_c] \in \mathbf{R}^{364}$. After completing the embedding, we passed both context and query into a depth-wise separable convolutional layer [13] and mapped them into the dimension of 128. This is quite similar to the traditional 1D convolutional layer but is memory efficient and has better generalization. The module "DepthwiseSeparableConv" is adopted from open-source repository [2], which is also cited in our code base.

4.3.2 Embedding Encode Layer

The input of this layer is from the result after depthwise separate convolutional layer. For this layer, we used embedding encode block. There are one positional encoding layer, four convolutional layers, one self-attention layer and one feed-forward layer stacked on each other and each operation is placed in a residual block, which solves the problem of vanishing gradients. Positional embedding provides the position information. For the convolutional layer, we adopted the same depth-wise separable

convolution mentioned in the previous layer with window size of 7 and filter number of 128. For the self-attention layer, we used multi-head attention defined in paper [10] with only 2 heads and implemented it similarly to the baseline with the specific masks for zeroing out the padding characters. This allowed us to explore the correlation between each tokens of the context and the query. We also implemented layer normalization between each layers. The feed-forward layer is relatively straight-forward, which is simply a linear layer with bias. All these operational layer consists of one encoding block and we only use one block in embedding layer.

4.3.3 Context-Query Attention Layer

Before computing the attention, we first obtained the similarity matrix between the context and the question. Each pair (c_i, q_j) has a similarity score stored in S_{ij}. The information include both context and question and also interaction between those two parts. The matrix can be obtained from the following formula:

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

where w_{sim}^T is the weight vector for the similarity matrix. The exact implementation can be found in the `get_similarity_matrix` method of the baseline code and it is a little different from the formula to be more memory efficient. For Context-to-Question Attention, we applied softmax function to each row of S and used that to obtain weighted sum of query hidden state q_j. The equations are as follow:

$$\bar{S}_{i,:} = softmax(S_{i,:})$$
$$A = \bar{S}Q^T \in \mathbf{R}^N$$

For Question-to-Context Attention, we applied softmax function to each column of S, which is marked as $\bar{\bar{S}}$ and got a new matrix S' by multiplying $\bar{S}$ and $\bar{\bar{S}}^T$. Then, we can use that to get the Q2C attention output by computing the weighted sums of context hidden states c_i. The equations are as follow:

$$\bar{\bar{S}}_{:,j} = softmax(S_{:,j})$$
$$S' = \bar{S}\bar{\bar{S}}^T \in \mathbf{R}^{N \times N}$$
$$B = S'C^T \in \mathbf{R}^N$$

Finally, we concatenated all values at each position to generate the output:

$$g_i = [c_i, a_i, c_i \circ a_i, c_i \circ b_i] \in \mathbf{R}^{512} \qquad \forall i \in \{1, ..., N\}$$

4.3.4 Model Encoder Layer

This layer is to encode the Context-Question Attention information and obtain the result to be used in the output layer. In this part, we used the exact same encoder block as that in embedding encoder layer except that we used 2 convolutional layers in each block. The other parameters are the same. Also, we repeated the same encoder block for 7 times to produce one model encoder block. Besides, we stacked 3 model encoder blocks together to produce the tensor can be used in the next layer.

4.3.5 Output Layer

As shown in Figure 3, we have 3 model encoder blocks stacked on each other. We can name the output of each as $M1$, $M2$, $M3$ from bottom to top. We obtained the information from encoding blocks, passed it in a linear layer and get the logits result. Then, we obtained the output probability of starting position and ending position separately from the outputs of the model encoder blocks with the following equations:

$$p^{start} = softmax(W_1[M_1; M_2])$$
$$p^{end} = softmax(W_2[M_1; M_3])$$

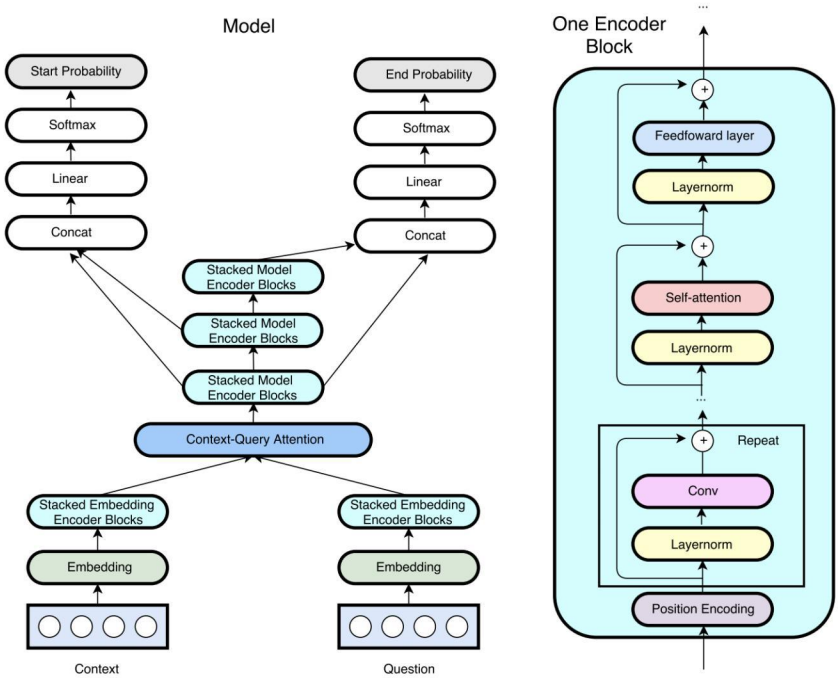

Figure 3: QANet Architecture [2]

Finally, we used the objective function that is defined as the average of the negative sum of the log probability of the predictions given the ground true starting positions and ending positions:

$$L(\theta) = -\frac{1}{N}\sum_{i=1}^{N}[\log p_i^{start} + \log p_i^{end}]$$

5 Experiments

5.1 Data

We use the official SQuAD 2.0 dataset [14] to deal with the task of question answering. For the training data, it is taken from the official SQuAD 2.0 training set. For the dev set, it is roughly half of the official dev set, randomly selected. For the test set, it is from the remaining examples from the official dev set, plus hand-labeled examples. For more details, it is in the Section 3 from handout.

5.2 Evaluation method

The performance is measured via two metrics, which are Exact Match (EM) score and F1 score. With different models and modifications based on the baseline model, we compared those two metric scores. In addition, we broke down the scores and did some error analysis on different kinds of questions. To be more specific, we divided the questions into different categories and inspected what kind of model can perform better on what category of questions.

5.3 Experimental details

First of all, we ran the baseline model and used the default arguments in the code base for 30 epochs. For each epoch, it took 22 minutes to train and it took 11 hours in total. Then, we modified the baseline model by adding character embedding, also using the default arguments in the code base for 30 epochs. In this experiment, it took about 24 minutes to train one epoch. We also tried another experiment with character embeddings only change the dimension of embeddings from 64 to 200,

and we also trained the model for 30 epochs and each took 27 minutes. After that, we implemented the new co-attention layer to substitute the attention layer in the baseline model. With the same parameters, we trained the model for 25 epochs and 24 minutes for each epoch. At last, we combined the both character embedding and co-attention layer in our model and we trained it for 25 epochs and each epoch took 26 minutes. For the QANet, we used the network architecture that is same as the original paper and trained the model for 25 epochs and each epoch took about 84 minutes. For the optimizer, we used Adam with beta parameters 0.8 and 0.999, decay weight 3e-7 and eps 1e-7. And for the learning rate, we also use 1000 steps to warm up.

5.4 Results

For the baseline model and each modified model based on the baseline, we can obtain the evaluation scores. And we list the scores in the following table.

Table 1: The performance among different models (dev set)

Models	F1	EM	AvNA
Baseline (25 epochs)	59.00	55.55	65.97
Baseline (30 epochs)	59.98	56.28	66.86
Baseline + Char Embed (30 epochs)	63.69	60.19	69.89
Baseline + Char Embed 2 (30 epochs)	**64.96**	**61.70**	**71.37**
Baseline + Char + CoAtt (25 epochs)	62.38	58.81	68.58
Baseline + CoAtt (25 epochs)	59.18	55.37	66.11
QANet (25 epochs)	54.77	52.97	62.14

We can see from the table that after adding character embedding to the baseline, we obtain about **3.7** increase in F1 score and almost **4** in EM score. The reason might related to that character-level embedding allow us to condition on the internal structure of the words. In this case, the model can get information from both word-level and subword-level, and better deal with the context comprehension and predict the answer span from it. From that, we have seen the power of character embeddings, so we decided to increase the embedding vectors dimension from original 64 to 200. In the implementation, we can just change the filter number of the convolutional layer. Suprisingly, the result increases by **1.3** in F1 score and **1.5** in EM score. The reason of the performance enhancement might be that with higher embedding dimension, the model can encode more character-level information and understand the context and question better.

The result of substituting the original attention layer with co-attention layer is not as good as we expected. We can see from the table that the performance of the model with co-attention is almost same as baseline model. Besides, we also run the experiment with the model equipped with both character embedding and co-attention, and it is not as good as the model only with character embedding. Besides, we also implemented the QANet from scratch and trained for 25 epochs. However, the result is out of our expectation, with a way worse performance than the baseline. During the experiment, we found that it took much more time to train for one epoch, compared to the baseline-related models, which is different from the argument in the paper that this model is efficient since it gets rid of RNN. And we also did not train the model to performance as good as that in the original paper.

From the experiment above, we choose our best performance model to be evaluated on the test dataset and get the result of **F1 score: 64.712, EM score: 60.997 on non-PCE leaderboard**. In summary, we can find that character embedding is more effective than co-attention mechanism in the baseline model structure. Since we only substitute part of the baseline model with co-attention layer, we cannot say that co-attention is useless but it should not be suitable in the BiDAF model. For the QANet, we cannot conclude its effects that was stated in the original paper, since with our re-implementation, we do not see any efficiency and higher performance.

6 Analysis

After reading through the text results in details, we found that there are several scenarios the system would like to fail. We show some examples along with analysis as follows and context is shorten due the its length.

6.1 Answer no-answer question

Question: How much did it cost to build Harvard Stadium?
Context: The University is organized into eleven separate academic units—ten faculties and the Radcliffe Institute for Advanced Study—with campuses throughout the Boston metropolitan area: ... Harvard's $37.6 billion financial endowment is the largest of any academic institution.
Answer: N/A
Prediction: $37.6 billion

When there is no answer, the model still tries to give an answer for the question. In this case, it searches for a number in the paragraph because the question contains the key words "How much". Also, the prediction phrase is very close to another key word "Harvard" of the query in the context. Besides, "cost" and "financial" are closely related in meaning. So, it is understandable for the model to give such a prediction.

6.2 Give insufficient answer

Question: What tribes supported British?
Context: Further south the Southeast interior was dominated by Siouan-speaking Catawba, ... The British were supported in the war by the Iroquois Six Nations, and also by the Cherokee – until differences sparked the Anglo-Cherokee War in 1758...
Answer: Iroquois Six Nations, and also by the Cherokee
Prediction: Iroquois Six Nations

The model correctly gives the prediction for the starting position, but it cut off the answer too quickly such that there is one point missing from the prediction. The reason might be that the model cannot distinguish the meaning of preposition words "and" clearly. Also, it might recognize the first "by" but cannot understand the second "by" thoroughly since the structure of the sentence is complicated.

6.3 Fail to give the answer

Question: What is one avenue being compensated for by having committees serve such a large role?
Context: Much of the work of the Scottish Parliament is done in committee. ... partly as a means of strengthening the role of backbenchers in their scrutiny of the government and partly to compensate for the fact that there is no revising chamber. ...
Answer: no revising chamber
Prediction: N/A

In this case, the model fails to give the answer for a answerable problem. The reason might be that the correct answer is a little far from the key word. In specific, "compensate" is the key word in the question, the answer "no revising chamber" in the context is after a phrase "the fact that ...". It is possible that the model dose not understand this phrase and predicts both start and end positions with quite low probabilities, which leads to "N/A" prediction.

7 Conclusion

In our project, we implemented and evaluated several end-to-end deep learning models to tackle question answering task on SQuAD 2.0 dataset. And the models can be classified to RNN-based model and Transformer-based model. Firstly, we experimented on the baseline model and modified it by adding character embeddings and implementing Co-Attention layers and achieved a quite better result than the baseline. Secondly, we implemented the QANet from scratch, from which we accumulated a lot of hand-on experience on building deep learning models. We also conducted experiment on the QANet model, though not obtaining the performance as good as that in the original paper, which states that QANet will not only speed up the training process, but also increase the

performance significantly. Overall, we achieved the result of **F1 score: 64.712, EM score: 60.997 on non-PCE leaderboard**. For the future, we may use some data augmentation or model ensemble techniques to enhance the model performance and switch to a better GPU to iterate the model quickly.

References

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

[2] https://github.com/heliumsea/qanet-pytorch.

[3] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*, 2018.

[4] Karl Moritz Hermann, Tomáš Kočiskỳ, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *arXiv preprint arXiv:1506.03340*, 2015.

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

[6] Alessandro Sordoni, Philip Bachman, Adam Trischler, and Yoshua Bengio. Iterative alternating neural attention for machine reading. *arXiv preprint arXiv:1606.02245*, 2016.

[7] Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. *arXiv preprint arXiv:1606.01549*, 2016.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252. PMLR, 2017.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[11] Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. Disan: Directional self-attention network for rnn/cnn-free language understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

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

[13] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[14] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.