Adding Char Embeddings and Self-Attention to SQuAD

NLP 2020 Spring Project
Xiaowei Yao and Zhao Han (Alphabetical Order)

Abstract

The Stanford Question Answering Dataset (SQuAD) [1] is one of the most popular dataset in Natural Language Process (NLP), which consists more than million questions posed by crowdworkers on a set of Wikipedia articles. It is one of the most benchmarks particularly in solving question answering task and reading comprehension task. In this project, we aim to use Non Pre-Trained Contextual Embedding methods to improve the baseline Bi-Directional Attention Flow model on solving question answering task based on SQuAD 2.0 dataset. Specifically, we focus on the embedding layer and attention layer to make improvement. Rather than singly using word-level embeddings in the base model, we extended the embedding layer by adding character-level embeddings. Another enhancement technique is adding an additional self-attention layer to the BiDAF model based on R-Net model. In addition to experiment each method, we also did a cross-experimentation to find out the best model. Finally, adding character-level embedding with additional self-attention layer achieved the best performance.

1 Introduction

In recent years, question answering as a machine reading comprehension (MRC) task has been the focus of many recent advancements in NLP research. The SQuAD 2.0 dataset is one of the most popular reading comprehension benchmarks. It consists of context paragraph, question, and answer triplets. A model that solves this task takes in the context paragraph and question as input, and predicts the answer to this question as a span of text from the context paragraph. "Exact-Match" (EM) is used to measure success of a model, the percent of predicted answers that match the ground truth perfectly, and F1, the harmonic mean of precision and recall. SQuAD 2.0 also introduces unanswerable questions and motivates an additional metric, "Answer vs No Answer" (AvNA), which measures the models accuracy at classifying questions as answerable or unanswerable. The goal of the challenge is to train a computer to answer the questions as correctly as possible, providing a measure for how well the computer can "understand" text. In past several years most research fall into two categories: Thoes use Pre-trained Contextual Embeddings (PCE) such as ELMo and BERT, and those do not. PCE models tend to have much higher performance than non-PCE models, but also come with an more complexity. In this project we primarily focus on project on improving existing non-PCE models.

For a baseline, the Bidirectional Attention Flow (BiDAF) model only used word-level embeddings. We added character-level embeddings to the current word-level embeddings. This improvement follows directly from the original BiDAF model, which also used character-level embeddings. We also worked to improve the attention layer by adding an additional self-attention layer as described by R-net (Wang et al., 2017). In experimentation, we individually tested each approach, and also tested when combing two approaches together. Our final results were done by using both quantitative measurement of EM and F1 scores and qualitative observation of answer predictions and their corresponding question and context paragraph.

2 Problem Statement

We are building a Question Answering (QA) model for reading comprehension task based on the Stanford SQuAD 2.0 dataset (Rajpurkar et al., 2018). Specifically we focus on the text-based and unanswerable questions. For QA, input of the model is a sentence of question for given context, output of the model will be words or span of text that answer the question.

3 Related Work

The baseline model is based on the BiDAF model developed by Seo et al (Seo et al., 2016). The difference is the original BiDAF model uses learned character-level word embeddings in addition to the word-level embeddings, while the baseline model doesn't include the character-level embedding layer. The original BiDAF model consists six layers:

1. Character Embedding Layer: map each word to a vector space using character-level CNNs
2. Word Embedding Layer: map each word to a vector space using a pre-trained word embedding model
3. Contextual Embedding Layer: utilize contextual cues from surrounding words to refine the embedding of the words (the first three layers are applied to both the query and context)
4. Attention Flow Layer: couple the query and context vectors and produces a set of query-aware feature vectors for each word in the context
5. Modeling Layer employs a Recurrent Neural Network to scan the context
6. Output Layer provides an answer to the query.

The primary contribution of this 2016 paper is that it extends on previous work utilizing attention mechanisms in machine comprehension by introducing a hierarchical multi-stage architecture for modeling the representations of the context paragraph at different levels of granularity. The proposed attention layer in this paper is not used to summarize the context paragraph into a fixed-size vector. Instead, the attention is computed for every time step, and the attended vector at each time step, along with the representations from previous layers, is allowed to flow through to the subsequent modeling layer. This reduces the information loss caused by early summarization. Additionally, it uses use a memory-less attention mechanism. That is, while we iteratively compute attention through time, the attention at each time step is a function of only the query and the context paragraph at the current time step and does not directly depend on the attention at the previous time step. This simplification leads to the division of labor between the attention layer and the modeling layer. It forces the attention layer to focus on learning the attention between the query and the context, and enables the modeling layer to focus on learning the interaction within the query-aware context representation (the output of the attention layer). It also allows the attention at each time step to be unaffected from incorrect attendances at previous time steps. The attention mechanisms flow in both directions, query-to-context and context-to-query, which provide complimentary information to each other. By using these techniques, this model achieved an F1 score of 77.3 and an EM score of 68.0 on the test set of SQuAD 1.0 dataset.

We planned to implement the following but we think exploring character-level embedding and self-attention is more reasonable. Since BERT (Devlin et al., 2019) was introduced last year, further improvements have been made by other researchers. Some of BERT's variants include RoBERTa (Liu et al., 2019) and ALBERT (Lan et al., 2019). RoBERTa optimized BERT by training it longer, using a larger batch size and using more data. ALBERT, however, applies parameter reduction techniques to reduce the memory usage and for really large models. There is also MLNet (Yang et al., 2019), which is apparently not a variant of BERT.

4 Approach

4.1 Character-Level Embeddings

The first improvement we made is adding an additional character-level embeddings. In the original BiDAF paper the authors use both character-level and word-level embedddings for their embedding layer. Adding character-level embedings can better address the issue of vocabularies that who is out of dictionary. The first step of the encoder is to embedding each word into a character-level, then feed these vector to a 1-dimensional convolutional

neural network. To calculate the i_{th} output feature for the t_{th} window of the input, a convolution is done between input window $X_{reshaped}[:, t : t+k1]$ and weights $W[i, :, :]$, and bias term b_i:

$(Xconv)i, t = sum(W[i, :, :]X_{reshaped}[:, t : t + k1]) + b_i$

The ReLU function and max-pooling are applied to the overall convolution output to produce the final embedding $Xconv.out$. Concatenating character-level embedding with word-level embedding to produce final embeddings.

$Xconv.out = MaxPool(ReLU(Conv1D(X_{reshaped})))$

4.2 Self-Attention

The second improvement we made is adding an additional self-attention layer. The implementation of self-attention is based on R-Net's "Self-Matching Attention". In the R-Net paper, the self-attention layer first computes an attention pooling vector c_t for each of n question-aware context passage word representations. Then v_t^P and v_t^P are concatenated and fed through a Bi-directional Recurrent Neural Network (BiRNN) to produce a final passage representation h_t^P.

$$s_j^t = \nu^T \tanh(W_1 v_j^P + W_2 v_t^P)$$
$$a_i^t = \frac{\exp(s_i^t)}{\sum_{j=1}^n \exp(s_j^t)}$$
$$c_t = \sum_{i=1}^n a_i^t v_i^P$$

The result is a new vector representation of the context passage where each word vector representation has information from the entire context passage. The self-attention layer in our model first computes the initial attention based on the BiDAF attention. Then this initial attention output is fed into the self-attention layer to obtain a self-attended attention. Finally the initial attention and self-attended attention are concatenated to obtain the final attention, which will then be fed through the original modeling layer.

5 Experiments

We evaluate our implementation on the datasets in the this section. We used EM (Exact Match) and F1 as metrics. For analysis purpose, we also plotted the metrics of AvNA (**A**nswer **v**s. **N**o **A**nswer) and NLL (negative log likelihood) loss. We will discuss each in the metrics section.

5.1 Dataset

The dataset for this project is Stanford Question Answering Dataset (SQuAD) 2.0 (Rajpurkar et al., 2018). SQuAD is a popular reading comprehension dataset, which consists of sentence-answer pairs. It also might be the case that the answer does not exist.

The context paragraphs in SQuAD are crawled form Wikipedia and the questions and answers are crowdsourced from workers on Amazon Mechanical Turk (MTurk). MTurk workers were asked to spent 7 minutes per paragraph (Rajpurkar et al., 2018). The answers are directly substrings in the context paragraphs. Thus it is more like highlighting the answers from paragraphs (TA, 2000). Multiple workers worked on answering the questions and additional workers answered all question for quality control purpose. There are approximately 150k (0.1M) questions and they are splitted to half development set and half test set. For word embeddings

5.2 Dataset Example

One example in the SQuAD dataset:

Context Paragraph: Apollo ran from 1961 to 1972, and was supported by the two-man Gemini program which ran concurrently with it from 1962 to 1966. Gemini missions developed some of the space travel techniques that were necessary for the success of the Apollo missions. Apollo used Saturn family rockets as launch vehicles. Apollo/Saturn vehicles were also used for an Apollo Applications Program, which consisted of **Skylab**, a space station that **supported three manned missions in 1973–74**, and the Apollo–Soyuz Test Project, a joint Earth orbit mission with the Soviet Union in 1975.

Question: What space station supported three manned missions in 1973–1974?

Answer: Skylab

5.3 Metrics

We have 2 metrics to evaluate our model.

1. EM (Exact Match): the metric measures the percentage of predictions that match any one of the ground truth answers exactly.

2. F1 score: this metric measures the average overlap between the prediction and ground truth answer.

	AvNA	EM	F1
Baseline	66.8	56.51	59.83
Char Emb.	69.08	59.07	62.5
Char Emb. + Self Attn.	**70.32**	**61.32**	**64.46**

Table 1: AvNA, EM, and F1 scores for all 3 approaches. Our combined approach (last row) achieved around 5 points more compared to baseline approach.

We also have 2 metrics to analyze our model performance, especially giving us insights at training time.

1. AvNA (**A**nswer **v**s. **N**o **A**nswer): this metric measures the classification accuracy, the predicted answers and no-answers.

2. NLL: this is the negative log likelihood loss. It is very similar to the cross entropy loss, which was used extensively in our assignments except that log softmax is not combined.

6 Experiment Results

We ran 4 million steps on all the models:

1. the baseline model,

2. the addition of char embedding and

3. our improved model with both char-level embeddings and self-attention.

We also ran all the model on Dandeneu 417 machine with Nvidia Quadro P4000 GPU, 1792 CUDA cores, 8 GB GDDR5 memory. The memory bandwidth is up to 243 GB/s.

1 shows

6.1 Baseline

The baseline training took 3 hours 40 minutes.

Figure 1 and Figure 2 are the NLL loss and AvNA metric for the dev set. Figure is 1 less interesting as it look similar to the loss we have seen in the class. The line became stable at around 70k steps. The baseline achieves 65% AvNA.

As seen from Figure 3 and 4, the values first drop at around 250k steps and increase steadily afterwards. This is because EM and F1 scores are not the loss functions which only affect them after time (TA, 2000).

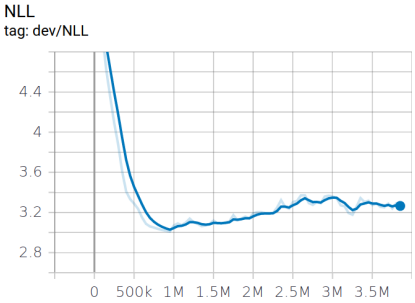

Figure 1: Baseline dev set: NLL loss. X axis shows the number of steps. Y axis shows the loss value.

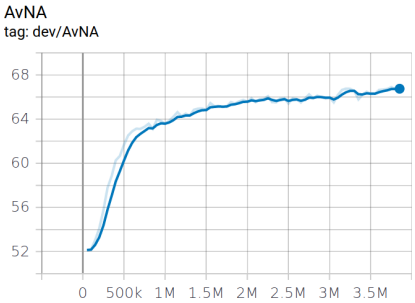

Figure 2: Baseline dev set: AvNA. X axis shows the number of steps. Y axis shows the percentage ratio.

6.2 With Char Embedding

The training with char embedding took over 5 hours, which is 1 hour slower than the baseline.

As you can see from Figure 7 to 9, all three metrics are increased, compared to the baseline. This shows that the character-level embedding did bring improvement, even though it is not very much unfortunately.

6.3 Char Embeddings and Self-Attention

Figure 11 to 12 show the visualization of the metrics when both char embeddings and self-attention are added.

Note that the color is changed. Baseline is in orange, char embedding is in blue, and char embedding with self-attention is in red.

As seen, all scores are improved.

It took 5.5 hours to train, which is an additional of 0.5 hour compared to char embedding only and an additional 1.5 hours compared to the baseline.

6.4 Prediction Comparison

Here we compare the prediction of 3 approaches.

Question: Why is Warsaw's flora very rich in species?

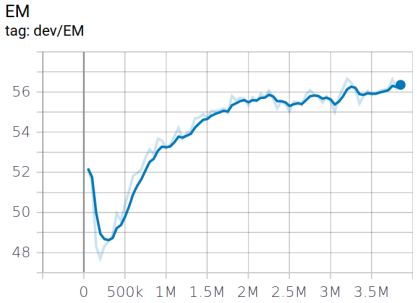

Figure 3: Baseline dev set: EM. X axis shows the number of steps.

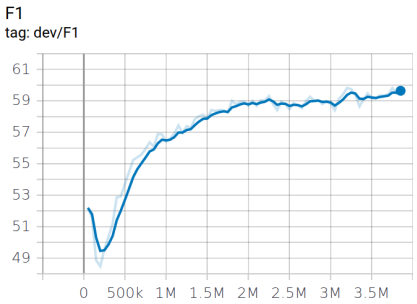

Figure 4: Baseline dev set: F1. X axis shows the number of steps.

Context: The flora of the city may be considered very rich in species. The species richness is mainly due to the location of Warsaw within the border region of several big floral regions comprising substantial proportions of close-to-wilderness areas (natural forests, wetlands along the Vistula) as well as arable land, meadows and forests. Bielany Forest, located within the borders of Warsaw, is the remaining part of the Masovian Primeval Forest. Bielany Forest nature reserve is connected with Kampinos Forest. It is home to rich fauna and flora. Within the forest there are three cycling and walking trails. Other big forest area is Kabaty Forest by the southern city border. Warsaw has also two botanic gardens: by the Łazienki park (a didactic-research unit of the University of Warsaw) as well as by the Park of Culture and Rest in Powsin (a unit of the Polish Academy of Science).

Answer: location of Warsaw

Here is the prediction for each approach:

Baseline: the location of Warsaw within the border region of several big floral regions

Char-embedding: the location of Warsaw within the border region of several big floral regions

Both: due to the location of Warsaw within the border region of several big floral regions

As seen, the self-attention mechanism gives "due to" some attention weight.

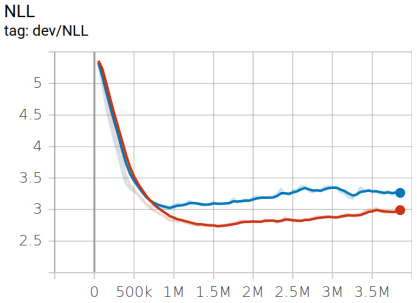

Figure 5: Char embedding dev set: NLL loss. X axis shows the number of steps. Y axis shows the loss value.

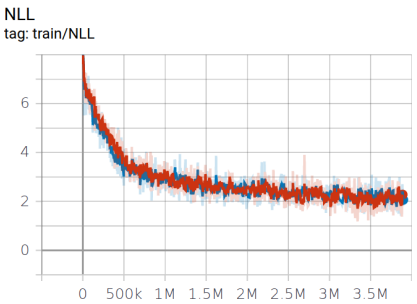

Figure 6: Char embedding dev set: NLL loss. X axis shows the number of steps. Y axis shows the loss value.

7 Discussion

Our the method with both char embedding and the self-attention achieved the best result in terms of EM, F1, and AvNA.

However, the improvement is not much: it only improved around 5 points in all three metrics. It would be interesting to see after tuning hyper-parameters, but each training session takes half-day, making it less favorable.

Adding more layers, like char embeddings & self-attention, make the model use more memory and take more time to run.

8 Conclusion

We added character-level embeddings and self attention mechanism. Our improved model performed better than the baseline in terms of EM and F1.

However, more layers increases memory usage but should be fine because it affects more at training time than the testing time

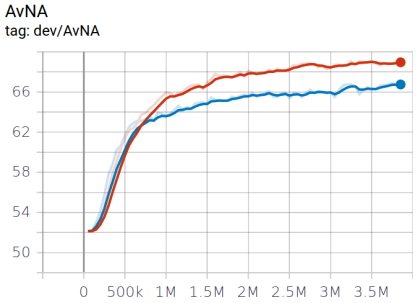

Figure 7: Char embedding dev set: AvNA. X axis shows the number of steps. Y axis shows the percentage ratio.

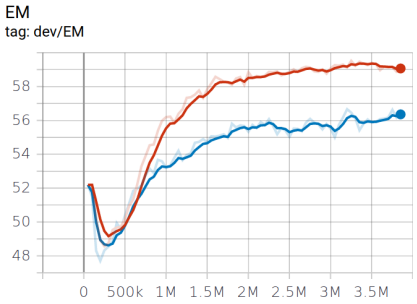

Figure 8: Char embedding dev set: EM. X axis shows the number of steps.

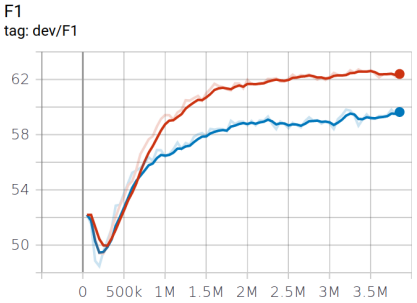

Figure 9: Char embedding dev set: F1. X axis shows the number of steps.

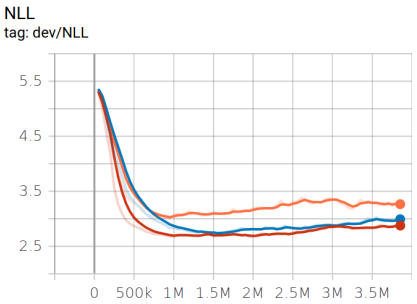

Figure 10: Char embedding and self-attention dev set: NLL loss. X axis shows the number of steps. Y axis shows the loss value. Note that the color is changed. Baseline is in orange, char embedding is in blue, and char embedding with self-attention is in red.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. pages 4171–4186.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* .

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. pages 784–789.

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

CS 224N TA. 2000. CS 224N Default Final Project: Question Answering on SQuAD 2.0. https://web.stanford.edu/class/cs224n/project/default-final-project-handout.pdf.

Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. 2017. 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)*. pages 189–198.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*. pages 5754–5764.

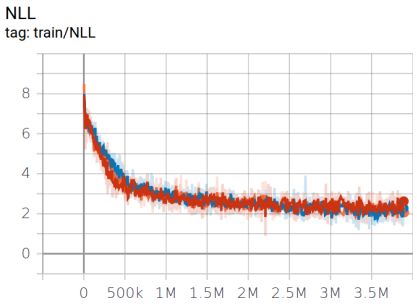

Figure 11: Char embedding and self-attention dev set: NLL loss. X axis shows the number of steps. Y axis shows the loss value. Note that the color is changed. Baseline is in orange, char embedding is in blue, and char embedding with self-attention is in red.

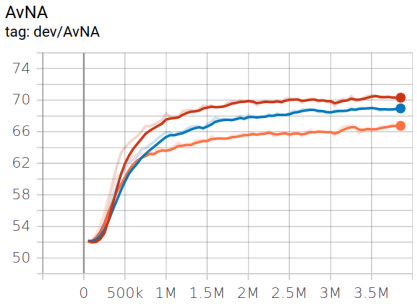

Figure 12: Char embedding and self-attention dev set: AvNA. X axis shows the number of steps. Y axis shows the percentage ratio. Note that the color is changed. Baseline is in orange, char embedding is in blue, and char embedding with self-attention is in red.

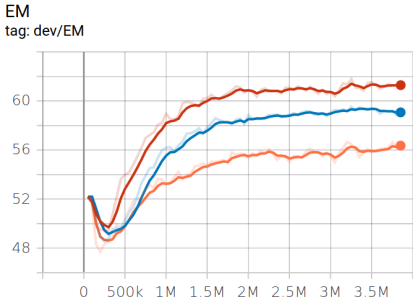

Figure 13: Char embedding and self-attention dev set: EM. X axis shows the number of steps. Note that the color is changed. Baseline is in orange, char embedding is in blue, and char embedding with self-attention is in red.

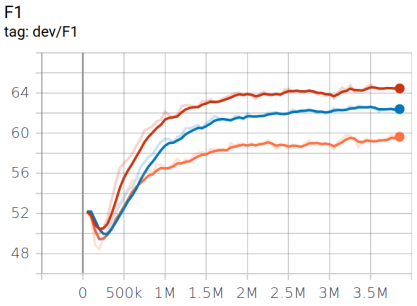

Figure 14: Char embedding and self-attention dev set: F1. X axis shows the number of steps. Note that the color is changed. Baseline is in orange, char embedding is in blue, and char embedding with self-attention is in red.