Character-Level Encoding with S4 for QA

Stanford CS224N Default Project

Peng Chen
Department of Computer Science
Stanford University
pengc@stanford.edu

Abstract

We propose a question answering model that encodes text inputs at character level to utilize subword structures and mitigate the out-of-vocabulary problem. Character-level inputs are quite long compared to word sequences, so the model encoding them should be able to handle long-range dependencies effectively and efficiently. S4 (Structured State Space sequence model) is such a model that fulfills this requirement. We build our model based on QANet, replacing all the CNN and self-attention layers with S4 layers, and feeding it with character embeddings. To make the model benefit both from character-level information and word-level information, our model first encodes inputs at character-level, and then aggregates the character-level hidden states to word-level hidden states by a max pooling operation, so that word-level modeling can be applied in the deeper stages of the network. Experiments on SQuAD 2.0 show that both S4 and character-level encoding improve the model performance on the question answering task. Case studies attribute part of this improvement to the out-of-vocabulary problem.

1 Key Information to include

- Mentor: Fenglu Hong
- External Collaborators (if you have any): No
- Sharing project: No

2 Introduction

Character embedding was proven to be beneficial to Question Answering (QA) tasks [1, 2]. It mitigates the out-of-vocabulary problem, and may take advantage of subword-level structures. However, the character-level embeddings in these models lack contextual information. The embeddings of characters are processed inside each word or between nearby words. There are limited inter-word interactions between embeddings of characters. One of the reasons is that all characters in a paragraph constitute a very long sequence, which CNNs and RNNs cannot model effectively due to long-range dependency, and self-attention cannot model efficiently due to quadratic complexity.

In this work, we adapt S4 (Structured State Space sequence model) [3], a model proposed recently, in our QA system to handle character sequences effectively and efficiently. S4 has shown its effectiveness on text classification tasks with character-level inputs. On several tasks the accuracy is improved by a large margin. [3] But, as far as we know, S4 has not been applied to QA tasks yet. S4 models sequences in a way fundamentally different from Transformers. It transforms input sequence into hidden state space with a first-order differential equation, whose parameters are trainable. [1] Therefore, the complexity of S4 is almost linear to the sequence length, rathar than the quadratic complexity

[1] An intuitive explanation. S4 is like the motion of an object, with the input sequence as the force applied to the object over time, and the hidden state as the position of the object. The velocity of the object is determined by the input sequences and its current position. This explanation is not mathematically correct, because S4

of self-atttention. This makes S4 extremely suitable for long sequences like character-level text processing. S4 is kind of similar to vanilla RNN that repeatedly applies transformation to the hidden state and adds the input signal to the hidden state. Actually it can be converted to an RNN after training. However, S4 is designed in a way that 1) operations on the hidden state over time can be done in parallel by fast Fourier transform, and 2) the transformation applied to the hidden state is initialized in a way that the hidden state can hold long-term memory. These features overcome the weakness of RNNs.

Our work is based on QANet [2] without applying the data augmentation method in that paper. We first replace self-attention layers in QANet with S4 layers to make sure it works properly. Then, we replace both self-attention layers and CNN layers with S4 layers. We find that it improves the performance of the model and reduces memory footprint so the model can be trained with a larger batch size. Finally, we modify the encoder in QANet to encode embeddings of characters with S4. We conduct experiments on the SQuAD 2.0 dataset [4]. Experiment results show that both S4 and character-level encoding can improve the model's performance on the QA task.

3 Related Work

Question Answering A question answering system answers questions asked by human with natural language responses. Question answering gains popularity as various large QA datasets are available publicly, such as SQuAD [5, 4], TriviaQA [6], HotpotQA [7], etc. They emphasize different aspects of question answering. In this work, we focus on the SQuAD dataset.

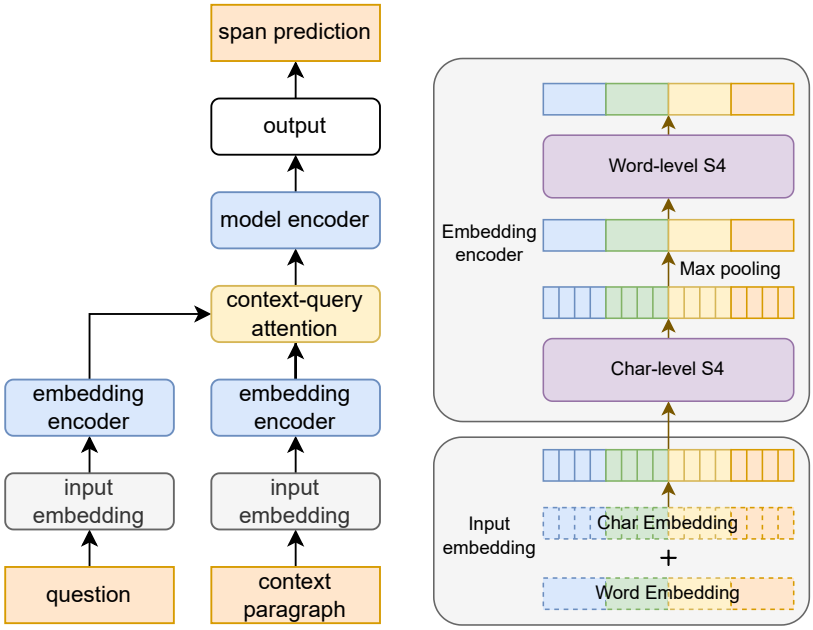
Character-Level Model There are several advantages to model texts in character level. It does not require tokenizing sentences into words or subwords; the dictionary is small and unknown tokens may be avoided; it is more robust to typos. A character-level model (or layer) can be local, i.e., a character only interacts with characters in the same word or in nearby words; or be global, i.e., every two characters interact with each other. Character-aware models, e.g. [8], are local. Recent models, e.g. CANINE [9], CharFormer [10], are global, but global layers are only applied after downsampling the sequence to reduce computational cost. CharFormer [10] groups characters to subword tokens with a trainable tokenizer. ByT5 [11] is a byte-level encoder-decoder model that does not downsample sequences, but the authors suggested that it should be used on short-to-medium length texts. There is a previous work [12] that encodes characters directly for question answering, but they use LSTM and CNN which have weakness on encoding long-range dependency. Different to these approaches, we use S4 layers to encode the inputs at character-level globally before downsampling, so more character-level information are retained in the global layers.

Long Sequence Modeling Transformer [13] has a stronger ability to model long-range dependency than RNNs like LSTM, by computing pairwise interactions of tokens directly. However, this has a price of quadratic complexity, which limits its application on long sequences like long paragraphs or character-level text. Recently, a great number of works try to solve this problem. Most of them propose variants of self-attention that have linear or almost linear complexity. Examples include Longformer [14], Reformer [15], Big Bird [16], and Performer [17]. Meanwhile, another line of research proposes methods fundamentally different from self-attention. HiPPO [18] leverages polynomial projections to capture long-range dependency. It uses a first-order differential equation to do this projection efficiently in linear time. S4 [3] takes a step further by making the parameters of the first-order differential equation trainable. The S4 paper shows that it gets significant improvements on the Long-Range Arena dataset [19] compared to variants of Transformers. Therefore, we follow the second line of research and use S4 layers in our question answering model.

4 Approach

We build our system incrementally based on the reference implementation of BiDAF [1] given in the starter code of the course, which is also our baseline. We first implement QANet without character embeddings, and then add character embeddings as described in the QANet paper [2]. Then, we replace self-attention layers, CNN layers, and feed-forward layers in the QANet with S4 layers.

involves first-order differential equations, while dynamics involves second-order differential equations, but their ideas are similar.



(a) QANet overall structure (b) Character-level encoder

Figure 1: Overall structure of network. (Left) The question and the context paragraph are first individually encoded into contextual representations. Then, a context-query attention is performed to make the context paragraph aware of the question. Finally, the representation of the context paragraph is further encoded by more neural network layers to predict the answer span. (Right) For character-level encoder, the input embedding is encoded at character-level; aggregated to word-level with max pooling; and encoded at word-level.

Finally, we add a stack of character-level S4 layers in the encoder, to take advantage of contextual embeddings of each character.

We implement all the models based on the reference implementation of BiDAF. All codes except the BiDAF baseline are written by ourselves. We refer (but not copy or import) the S4 official code `https://github.com/HazyResearch/state-spaces` for some details that are not explained in the S4 paper.

4.1 Problem Formulation and Notations

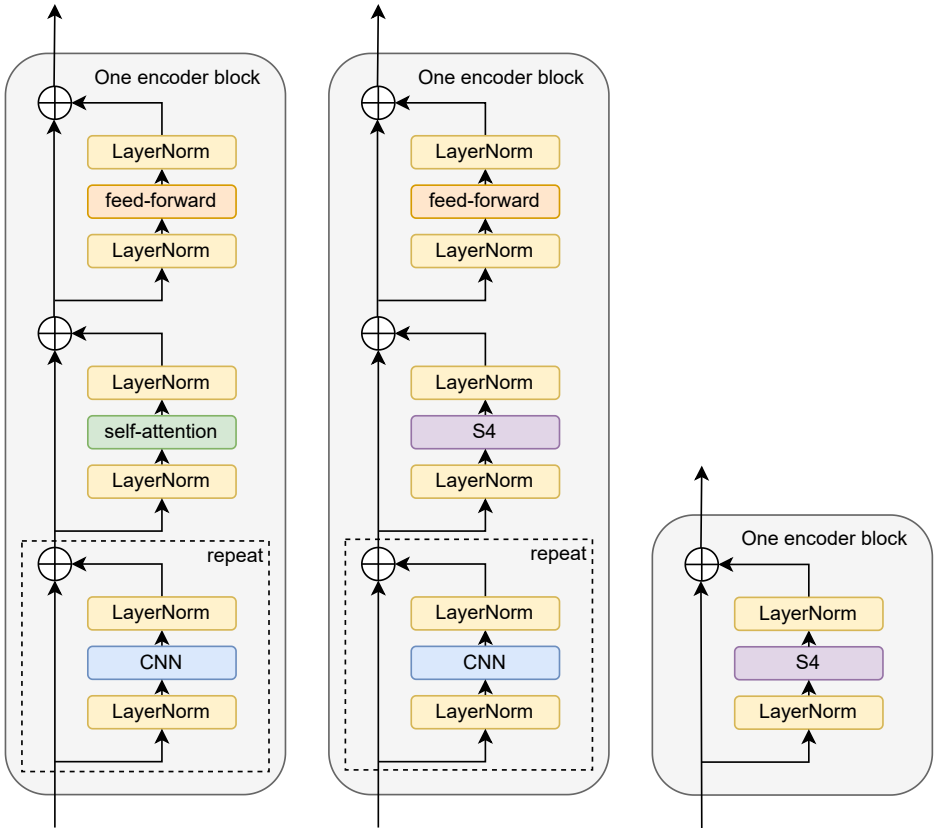
Our work targets the SQuAD dataset. The task is defined as follows. Given a context paragraph $C = \{c_1, c_2, \ldots, c_n\}$ and a question (query) $Q = \{q_1, q_2, \ldots, q_m\}$, find a span $S = \{c_B, c_{B+1}, \ldots, c_E\}$ of the context that answers the question. An empty span means the question is unanswerable given the context. Here, c_i and q_i are words in the context paragraph and the question respectively. When we want to refer characters in a text, the notation c_{ik} represents the k-th character in the word c_i, and q_{ik} represents the k-th character in the word q_i.

4.2 QANet

Our implementation of the QANet is almost the same with the original QANet paper [2], except that we introduce a novel normalization method to balance contributions of CNN layers and self-attention layers. QANet consists of five modules as shown in Figure 1a. The input data flow through these five modules step-by-step to a prediction of the answer span.

Input embedding This module encodes words into vector representations through embedding lookup table. The embeddings are initialized with GloVe [20] word vectors and fixed during training. A two-layer highway network [21] is then applied to the embeddings.

$$\hat{c}_i = \text{Highway}(\text{Emb}(c_i)) \qquad \hat{q}_i = \text{Highway}(\text{Emb}(q_i)).$$



(a) Encoder in QANet (b) QANet-with-S4 (c) QANet-S4-only

Figure 2: Encoder blocks. The encoder of QANet consists of CNN, self-attention and feed-forward. We first replace the self-attention layer with an S4 layer. Then, we remove CNN and feed-forward layers so there is only S4 layers in the encoder.

Embedding encoder The embedding encoder is illustrated in Figure 2a. The encoder consists of a stack of encoder blocks. In each encoder block, there are several CNN layers, a self-attention layer, and a feed-forward layer. Each of these layers is wrapped in a residual block. In this stage, the context paragraph and the question are processed separately with no interactions between them.

$$\tilde{c}_i = \text{Encoder}(\hat{c}_i) \qquad \tilde{q}_i = \text{Encoder}(\hat{q}_i).$$

Context-query attention This step gathers query information into each word of the context paragraph. First a similarity value for each pair of query word q_j and context word c_i is computed as $S_{ij} = W_S[\tilde{q}_j, \tilde{c}_i, \tilde{q}_j \odot \tilde{c}_i]$ where $\odot$ is element-wise multiplication and W_S is a trainable linear transformation. The context-to-query attention for context word c_i is defined as $a_i = \sum_j \tilde{q}_j \exp(S_{ij}) / \sum_j \exp(S_{ij})$. The query-to-context attention is defined as $\hat{b}_j = \sum_i \tilde{c}_i \exp(S_{ij}) / \sum_i \exp(S_{ij})$, $b_i = \sum_j \hat{b}_j \exp(S_{ij}) / \sum_j \exp(S_{ij})$. The results of these attention operations are concatenated into a single vector for each context word as $x_i = [\tilde{c}_i, a_i, \tilde{c}_i \odot a_i, \tilde{c}_i \odot b_i]$.

Model encoder The model encoder is the same as the embedding encoder except the depth of network. We apply it three times to the output of the last step.

$$m_i^0 = \text{Model}(x_i) \qquad m_i^1 = \text{Model}(m_i^0) \qquad m_i^2 = \text{Model}(m_i^1)$$

Output This step predicts the span to answer the question. For context word c_i, the probability that it is the start of the span p_i^B and the probability that it is the end of the span p_i^E are modelled as

$$p_i^B = \text{softmax}(W_B[m_i^0, m_i^1]) \qquad p_i^E = \text{softmax}(W_E[m_i^0, m_i^2])$$

where W_B and W_E are trainable linear transformations. The probability that the answer span starts at word c_i and ends at word c_j is therefore $p_{ij} = p_i^B p_j^E$.

4.2.1 Normalization

In the embedding encoder and the model encoder, we modify normalization layers to make the model easier to train. The original QANet applies layer normalization as

$$z = x + f(\text{LayerNorm}(x)),$$

where f is CNN, self-attention or feed-forward layer. We propose

$$z = x + g \cdot \text{LayerNorm}(f(\text{LayerNorm}(x))),$$

where $g = 0.5$ is a fixed factor. Figure 2 demonstrates the structure of our normalization scheme. We insert this extra layer normalization to ensure that the two terms of the addition in residual connection have similar scales; otherwise, the residual connection will be degenerated. It also balances the contribution of CNN and self-attention in the network. These effects can also be achieved by carefully setting the initial values of parameters, but adding an extra normalization layer eliminates the need of tuning initialization-related hyper-parameters.

4.2.2 Character Embeddings

We add character embeddings to the model the same way as QANet [2]. Each character is assigned a 200-dim vector. All the character embeddings in a word are aggregated by max pooling. It is then concatenated with the word embedding before fed into the highway network.

$$\hat{c}_i = \text{Highway}([\text{Emb}(c_i)), \max_k(\text{Emb}(c_{ik}))]) \qquad \hat{q}_i = \text{Highway}([\text{Emb}(q_i)), \max_k(\text{Emb}(q_{ik}))]).$$

We treat all low-frequency characters as UNK to reduce the number of parameters and make it generalize better.

4.3 S4 (Structured State Space Sequence Model)

4.3.1 A Brief Introduction of S4

We give a brief introduction of S4 in this section. Readers familiar with S4 may skip this section. Some implementation details, as well as mathematical proofs and tricks, are omitted. Please refer to the S4 paper [3] for these details.

S4 [3] is a sequence model that models long sequences much more efficiently than self-attention and delivers a comparable performance. S4 is based on the differential equation 1. It maps the input signal $u(t)$ to an internal hidden state $x(t)$, and projects them into the output signal $y(t)$. The coefficient matrices A, B, C, D are trainable parameters.

$$\begin{aligned} x'(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{1}$$

Discretizing the equation 1 results in the equation 2 where Δ is a trainable step length. The discrete form is suitable for text inputs, as the discrete time k corresponds to the position of word/character in the text and u_k is the embedding of the word/character.

$$\begin{aligned} x_k &= \bar{A}x_{k-1} + \bar{B}u_k \quad & \bar{A} &= (I - \Delta A/2)^{-1}(I + \Delta A/2) \\ y_k &= Cx_k + Du_k & \bar{B} &= (I - \Delta A/2)^{-1}\Delta B \end{aligned} \tag{2}$$

S4 initializes the matrix A with the HiPPO matrix [18], which minimizes the error if we try to restore $x(t)$ from $u(t)$. HiPPO matrix has a nice property that can be leveraged to compute equation 2 efficiently. The HiPPO matrix equals to a normal matrix [2] plus a low-rank matrix (NPLR), i.e.

$$A = V^{-1}(D + pq^T)V$$

where V is a unitary matrix, D is a diagonal matrix, and p, q are vectors. To keep this NPLR property hold during training process, the matrix A is parameterized by D, p, q.

[2] A normal matrix is a matrix that unitarily similar to a diagonal matrix.

Noticing that y is a convolution of u by expanding the equation 2 over time, S4 can be calculated efficiently by discrete Fourier transform. The convolution kernel contains powers of the matrix $\bar{A}$ which are expensive to calculate, but its Fourier coefficients are related to matrix resolvent, which converts matrix power to matrix inverse. With Woodbury identity $(D + pq^T)^{-1} = D^{-1} - D^{-1}pq^T D^{-1}/(1+q^T p)$, solving the inverse of an NPLR matrix can be reduced to solving the inverse of its diagonal part. Computing the inverse of the diagonal matrices for all the Fourier frequencies is equivalent to the computation of a Cauchy kernel, which can be implemented very efficiently. [3]

4.3.2 Application of S4 on QANet

S4 is a unidirectional model as indicated by the equation 2. Therefore, we apply one S4 forward in time and another S4 backward in time to capture the context better, exactly the same as what bidirectional LSTM does.

We first replace self-attention layers in the QANet with S4 layers, as S4 is a competitor of Transformers. We call this model QANet-with-S4. Then, we take a step further to replace CNN and feed-forward layers with S4 layers, so that there are only S4 layers left in the network. It is reasonable for S4 to replace CNN and feed-forward layers because S4 itself applies linear transformation to hidden states. We call this model with S4 layers only QANet-S4-only.

4.4 Character-Level Encoding

In this section, we introduce character-level encoding to QANet-S4-only by modifying the input embedding and embedding encoder. The modified input embedding together with embedding encoder is illustrated in Figure 1b. The input embedding module generates one vector per character rather than producing one embedding per word, by concatenating the embedding of the character and the embedding of the word as follows.

$$\hat{c}_{ij} = \text{Highway}([\text{Emb}(c_i), \text{Emb}(c_{ij})]) \qquad \hat{q}_{ij} = \text{Highway}([\text{Emb}(q_i), \text{Emb}(q_{ij})]).$$

The embedding encoder is broken into a character-level part and a word-level part. The character-level part encodes the character-level embedding from the input embedding module to produce a contextual character embedding. This contextual embedding is then aggregated inside each word with max pooling. The resultant word-level embedding is further processed by a word-level encoder. The whole embedding encoder can be formulated as the following equations, where WordEncoder and CharEncoder are composed of S4 layers as shown in Figure 2c.

$$\tilde{c}_i = \text{WordEncoder}(\max_j \text{CharEncoder}(\hat{c}_{ij})) \qquad \tilde{q}_i = \text{WordEncoder}(\max_j \text{CharEncoder}(\hat{q}_{ij})).$$

5 Experiments

5.1 Data

We evaluate our method on the SQuAD 2.0 dataset [4], an extractive reading comprehension dataset. Given a paragraph and a question, the model is required to fetch an answer to the question from the paragraph. As an improvement to SQuAD 1.0 [5], the question may be unanswerable given the paragraph. In this case, the model is required to output an empty answer.

5.2 Evaluation method

We evaluate models with EM, F1 and AvNA scores.

- **EM (Exact Match)** measures whether the model prediction matches the ground truth exactly. The EM score is 1 if they match and 0 if not.
- **F1** is the harmonic average of precision and recall of the answer. Larger the overlap of the model prediction and the ground truth, higher the F1 score.
- **AvNA (Answer vs No Answer)** is the accuracy on whether the question is answerable.

[3] As there is no implementation of Cauchy kernel on GPUs, we follow the authors of the S4 paper to implement it with the `pykeops` library. A naive implementation in `pytorch`, however, costs significantly more GPU memory and slows down training.

5.3 Experimental details

Starting from the BiDAF baseline provided by the reference implementation, we compare several models we proposed in the previous section. We first evaluate our implementation of QANet without character embeddings, and then evaluate it with character embeddings. Next, we train the QANet-with-S4 model which replaces self-attention layers with S4 layers and the QANet-S4-only model which replaces all CNN, self-attention and feed-forward layers with S4 layers. Finally, we evaluate QANet-S4-char which is based on QANet-S4-only and encodes input texts at character level.

For BiDAF, we train the model with the default hyper-parameters given in the reference implementation. For QANet and all models based on QANet, following the paper [2], we use the Adam [22] optimizer. The size of the network is determined by the GPU memory usage so that the model can be fit into a 12 GB GPU. We find that S4 is more memory efficient than self-attention, so we double the batch size for models with S4. More details of the hyper-parameters can be found in the appendix.

5.4 Results

Model	EM	F1	AvNA
BiDAF (by [4])	59.8	62.6	–
BiDAF w/o char emb (our impl.)	57.64	60.90	67.55
QANet w/o char emb	62.12	65.69	73.15
QANet	64.46	67.58	73.57
QANet-with-S4	62.12	66.03	73.53
QANet-S4-only	64.70	68.16	73.70
QANet-S4-char	66.17	69.48	75.77
QANet-S4-char (ensemble)	69.10	71.90	76.58

Table 1: EM (Exact Match), F1 and AvNA scores on the dev set.

We conduct experiments on the dev set to compare different models. The results of our experiments are listed in Table 1.

Self-attention vs S4 We find that QANet outperforms BiDAF by a margin of about 5 points on both EM and F1. This coincides with the results on SQuAD v1 [5] provided by QANet paper [2], that self-attention combined with CNN is more effective than LSTMs. There is a performance drop from QANet to QANet-with-S4, when replaceing self-attention with S4 layers. But after replacing all layers with S4, QANet-S4-only outperforms QANet by around 0.6 points on F1. The reason for the drop may be that there are not enough S4 layers in QANet-with-S4, and the performance catches up self-attention if we use the computational budget of CNN and feed-forward on S4. Therefore, the performance of S4 is comparable with self-attention.

Character-level encoding QANet with character embeddings outperforms QANet without character embeddings by about 2 points on the EM and F1 scores. This shows that adding character embeddings is very useful, even if it is used in a simple way, max-pooling the character embeddings in each word before feeding them into the encoder. Furthermore, feeding character embeddings into the encoder and encoding them at character level leads to an additional improvement of 1.3 points on F1 from QANet-S4-only to QANet-S4-char. This shows that modeling the long-range dependency among characters in texts benefits the question answering task.

Test set results We submit the QANet-S4-char model to the leaderboard. A single model achieves an F1 of 66.80 and an EM of 63.47, and an ensemble of 20 models achieves an F1 of 69.70 and an EM of 66.78.

6 Analysis

Case study on character-level encoding We show a case that character-level encoding is necessary for the model in Figure 3. Experiments on this case show that the model with character-level encoding generates an answer closer to the correct answer. In this example, the word "nonfriendly" in the

Context Although lacking historical connections to the Middle East, Japan was the country most dependent on Arab oil. 71% of its imported oil came from the Middle East in 1970. On November 7, 1973, the Saudi and Kuwaiti governments declared Japan a "nonfriendly" country to encourage it to change its noninvolvement policy. It received a 5% production cut in December, causing a panic. On November 22, Japan issued a statement "asserting that Israel should withdraw from all of the 1967 territories, advocating Palestinian self-determination, and threatening to reconsider its policy toward Israel if Israel refused to accept these preconditions". By December 25, Japan was considered an Arab-friendly state.
Question To force Japan to be more involved in the crisis, what did Saudi and Kuwaiti government do?
Answer by QANet without character embeddings change its noninvolvement policy
Answer by QANet-S4-only encourage it to change its noninvolvement policy
Answer by QANet-S4-char nonfriendly

Figure 3: A case that only the model with character-level encoding answers correctly.

answer is not appeared in the GloVe. But it is part of the answer, so understanding this word is useful for the model to make a correct prediction. A word-level model treats the word as UNK, while a character-level model can get its meaning by noticing that it consists of the prefix “non” and the stem “friendly”. Although the word “noninvolvement” appears in the GloVe, it is easier for a character-level model to recognize that it has an opposite meaning to the word “involved” in the question than a word-level word, so the character-level model will not include it in the answer because it is a reiteration of the question.

		Ground truth	
		Answerable	Unanswerable
System output	Answerable	2314 (38.88%)	908 (15.26%)
	Unanswerable	534 (8.97%)	2195 (36.88%)

Table 2: Confusion matrix of AvNA on the dev set.

Unanswerable Questions We analyze the behavior of our model (QANet-S4-char) on dealing with unanswerable questions. The confusion matrix on the dev set is shown in Table 1. The model tends to answer unanswerable questions more frequently than to refuse to answer answerable questions. This result suggests that the model is biased towards generating an answer rather than asserting that the question is unanswerable. This behavior may be changed by setting a threshold that if the predicted probabilities of all possible answer spans are lower than the threshold the question is considered unanswerable. A more general approach is to train a separate output layer that predicts whether the question has an answer. This approach may be more flexible than our model when we want to control the behavior of the model in an application.

7 Conclusion

We build a question answering system that encodes texts at character level. Our model is based on the QANet structure, with S4 layers to efficiently and effectively model long sequences formed by all the characters in the text input. The network with S4 has a performance comparable to the network with self-attention, and is more memory-efficient. The model with character-level encoding outperforms the model without it. We suppose the reason is that character-level encoding can handle out-of-vocabulary issue. It would be interesting to further investigate whether our method works with pretraining language models, and whether more sophisticated character-encoding methods, e.g. upsampling the sequence from word-level to character-level at some deeper stage of the network, will be helpful.

References

[1] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[2] 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. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[3] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *CoRR*, abs/2111.00396, 2021.

[4] Pranav Rajpurkar, Robin Jia, and Percy Liang. 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, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[5] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. In Jian Su, Xavier Carreras, and Kevin Duh, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2383–2392. The Association for Computational Linguistics, 2016.

[6] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1601–1611. Association for Computational Linguistics, 2017.

[7] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2369–2380. Association for Computational Linguistics, 2018.

[8] Yoon Kim, Yacine Jernite, David A. Sontag, and Alexander M. Rush. Character-aware neural language models. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2741–2749. AAAI Press, 2016.

[9] Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. CANINE: pre-training an efficient tokenization-free encoder for language representation. *CoRR*, abs/2103.06874, 2021.

[10] Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Prakash Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. Charformer: Fast character transformers via gradient-based subword tokenization. *CoRR*, abs/2106.12672, 2021.

[11] Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *CoRR*, abs/2105.13626, 2021.

[12] Xiaodong He and David Golub. Character-level question answering with attention. In Jian Su, Xavier Carreras, and Kevin Duh, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1598–1607. The Association for Computational Linguistics, 2016.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[14] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *CoRR*, abs/2004.05150, 2020.

[15] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[16] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[17] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[18] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[19] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena : A benchmark for efficient transformers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[20] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL, 2014.

[21] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

A More Experiment Details

For the encoders in all the models we choose a hidden size of 128. For QANet and QANet-with-S4, there is one encoder block in the embedding encoder and 7 encoder blocks in the model encoder. For QANet-S4-only, there are 6 encoder blocks in the embedding encoder and 12 encoder blocks in the model encoder. (Note that there is only one layer in an S4-only block but at least 4 layers in a QANet block, so QANet-S4-only is shallower than QANet.) For QANet-S4-char, there are 4 character-level blocks and 3 word-level blocks in the embedding encoder, and 8 blocks in the model encoder. The CNN layers are repeated 4 times in the embedding encoder with a kernel size of 7, and are repeated 2

times in the model encoder with a kernel size of 5. The self-attention has 8 heads. The dimension of hidden states inside S4 layers is 64.

The parameter of the Adam optimizer is $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We manually tune the learning rate and settle with 5e-4, with a warm-up of 2000 steps. We set a batch size of 16 for QANet to fit into 12 GB GPU memory, but we double both the batch size and the learning rate for QANet-S4-only and QANet-S4-char because S4 is more memory efficient than self-attention. The model is trained on the SQuAD 2.0 dataset for 30 epochs. For character embeddings, we randomly initialize them with normal distribution whose standard deviation is $1/4$ of that of word embeddings, so that they have similar scale after the max pooling. For character embeddings in QANet-S4-char, however, we randomly initialize them with normal distribution whose standard deviation is 4 times of that of word embeddings, to prevent the model focusing on word embeddings and ignoring character embeddings.

All the training is done on a single 3080Ti GPU.