Question Answering for SQuAD 2.0

Andrew Deveau
adeveau@stanford.edu

Abstract

In this work, we experiment with several approaches to question answering on SQuAD 2.0. We augment a baseline model based on BiDAF with character embeddings and see a gain in performance. Additionally, we experiment with implementing a variant of QANet and a handful of other approaches to adding self-attention to our models, but find they do not improve performance as currently implemented.

1 Introduction

In this work, we investigate several approaches to question answering, with a particular focus on applications of self-attention. In particular, we use SQuAD 2.0, which is a challenging question answering data set where each example consists of a question and a paragraph of context which contains the answer if one exists. Approximately one third of the examples are not answerable; this makes for a substantially more difficult task, as reflected in the gap between performance on SQuAD 1.1 and 2.0. Question answering and reading comprehension have immediate applications in information retrieval, among other areas. Apart from possible applications, they also provide one way to approach natural language understanding, which is of independent interest.

Currently, competitive methods for SQuAD 2.0 make heavy use of pretrained contextual embeddings learned from large unsupervised corpora. In this work, we focus on models which do not leverage such contextual embeddings. Drawing inspiration from the success of the Transformer [2] and similar self-attention based models on a variety of NLP tasks (including learning contextual embeddings), we focus particularly on models which make use of self-attention.

Ultimately we find that our models which incorporate self-attention do not outperform an RNN-based baseline. Our best model, which is simply the BiDAF-inspired baseline augmented with character embeddings, achieves a test F1 score of 64.791.

2 Related work

SQuaD has attracted considerable interest and there are many papers that propose models specifically designed to perform well on it. Our work draws most heavily on BiDAF [1] and QANet [5], both of which gave state-of-the-art results at the time of their introduction. Like many SQuAD architectures, both BiDAF and QANet compute contextual representations of each word in the query and context, combine these representations through an attention mechanism, pass the combined representation to additional encoding layers, and conclude with a relatively simple pointer network to produce distributions over the starting and ending tokens.

BiDAF's core contribution is the introduction of a bidirectional attention layer in which both a representation of the context conditioned on the query *and* of the query conditioned on the context are computed. We use a variant of BiDAF as our baseline in this work. A small modification to that baseline also turns out to be our best performing model.

Although similar to BiDAF in that it also uses a bidirectional attention layer, QANet is distinguished by the use of blocks consisting of convolutions and self-attention in place of RNNs throughout the

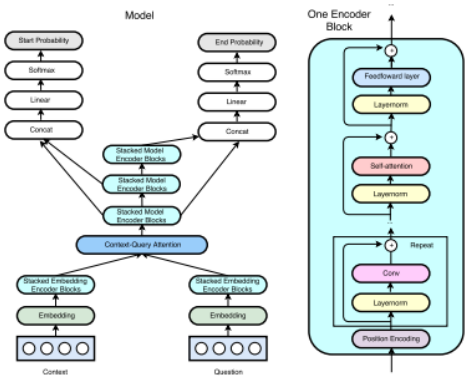


Figure 1: QANet as depicted in the original paper [5]

model. The attention in these blocks is identical to that first proposed for use in the Transformer in Vaswani et al [2]. We experiment both with a re-implementation of QANet and another architecture which incorporates self-attention.

3 Approach

We implement several models, all broadly based on BiDAF and QANet. Our baseline is the provided BiDAF implementation without character embeddings.

3.1 Character Embeddings

Our simplest (and best performing) model is simply this baseline model augmented with character-based word embeddings. We take a relatively standard approach to learning these embeddings. Each character has a trainable embedding in $\mathbb{R}^{64}$. For each word, its character embeddings are passed through a one dimensional convolutional layer and the result is max-pooled over the entire sequence length to give a 64 dimensional embedding for each word.

To combine these embeddings with the pretrained GloVe vectors, we concatenate the two representations, project the concatenation using a linear layer, and then pass the result through a two layer highway network. This approach, (with higher dimensional character embeddings) is also used in QANet. We also train a variant of this model as part of an ablation study.

3.2 QANet and Self-Attention

In an attempt to further improve over this model, we implement two models which make use of self-attention.

The first of these is a (broken) reimplementation of QANet. As mentioned above, on a high level, QANet follows the general approach of many SQuAD models. It begins with word embedding layers for the question and context. This layer is nearly identical to that used in our augmented baseline. It differs only in that the <UNK> word embedding is not frozen during training and that each character embedding is of dimension 200 instead of 64.

These embeddings are then passed through an encoding block which draws heavily on the Transformer encoder block of Vaswani et al [2]. Positional encoding is applied at the start of the block. This is followed by a depthwise convolutional layer, a multiheaded attention layer, and a two layer positionwise feedforward network. Each of these layers is preceded by a layer norm and has a residual connection around it. Notably, the only difference between the last two layers and a Transformer encoder block is that the layer norm is applied before each layer instead of after.

Explicitly, multiheaded attention is computed as follows. For the i^{th} of the n heads, there are trainable matrices $W_Q^i, W_K^i, W_V^i \in \mathbb{R}^{d_v \times d}$ where d is the model dimension and d_v the embedding dimension.

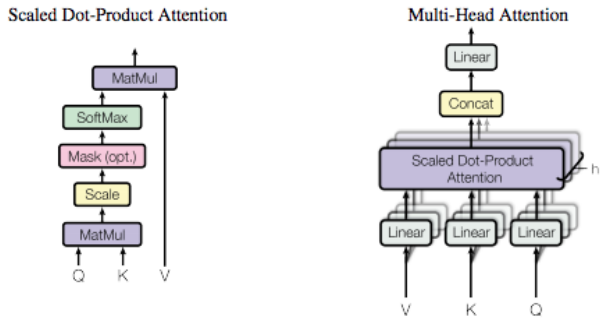


Figure 2: Multiheaded attention as depicted in Attention is All You Need [2]

In our implementation, d_v is fixed to be d/n. For the i^{th} head, the result of the attention is

$$\text{softmax}\left(W_Q^i Q K^T W_K^{i^T} / \sqrt{d_v}\right) W_V^i V$$

where the rows of Q, K, V are the query, key, and value vectors respectively. That is, each of the keys, queries, and values are projected and then scaled dot product attention is applied to the projections. The results from each of the heads are then concatenated to form vectors of dimension d which are passed through a trainable linear layer. This is depicted in Figure 2.

The outputs of the encoding layer are passed through a bidirectional attention layer which is identical to that used in the baseline provided. The output of the attention layer is projected via a linear layer $W_{proj} \in \mathbb{R}^{4d \times d}$ and then passed through three stacked encoding layers. These layers consist of 7 consecutive blocks each identical in architecture to the encoding block preceding the bidirectional attention layer. They differ only in that they use filters of size 5 instead of size 7.

All three layers share weights. Finally the outputs of the first and third of these layers are concatenated and passed to a linear layer and softmaxed to obtain the distribution over starting tokens. Similarly, the outputs of the second and third layers are passed through a linear layer and softmaxed to obtain the distribution over ending tokens. QANet was originally designed for SQuAD 1.1. Each example in the data set has a padding character prepended. In order to handle questions without an answer, we simply declare that the model has predicted the question does not have an answer if the most likely span starts and ends at that character. This is the same approach used for all of our models, including the baseline.

We reduce the model dimension from 128 to 96 and the number of attention heads in each block from 8 to 4, but otherwise (attempt to) replicate the model described in the original QANet paper exactly. We use code from the baseline where appropriate and an open source implementation of the Transformer [6], but otherwise write all code ourselves.

Finally, we experimented with adding Transformer blocks to the baseline augmented with character embeddings in hopes that it would improve performance. We trained a model with a single small Transformer block with two attention heads following the model encoder layer. This performed worse than the model which adds only character embeddings.

4 Experiments

We trained two versions of the baseline model augmented with character embeddings with the first effectively an ablation of the second.

In the first, we froze the initial random character embeddings and learned embeddings of size 50. These embeddings were then concatenated with the GloVe representations and passed through a 2 layer highway network.

In the second, we implemented the exact model described in the approach section above. We used 100 filters, each with kernel size 5 in the CNN and projected the concatenated embeddings to vectors of dimension 100 before passing them through the highway network.

For both training runs, we used the baseline configuration – Adadelta with learning rate .5, no weight decay, and batch size 64. Both models were trained for 30 epochs, although we observed that performance on the dev set plateaued well before the end of training. We do no additional preprocessing and use the same exponential moving averaging on the weights used for the baseline. We use dropout with $p = .1$ in the character embedding layer in both instances. It takes approximately 14 hours to complete 30 epochs on half of an Nvidia M60.

Unsurprisingly, the latter model performs better, though both models outperform the baseline as indicated in Table 1.

For the second model, we additionally performed some basic hyperparameter tuning to select the weight decay penalty. We trained models with weight decay parameter $\lambda = 10^{-5}, 10^{-6}, 10^{-7}$. We find that $\lambda = 10^{-6}$ improves the dev F1 score by .259, but reduces the dev EM score by .05. On the test set, the model trained with weight decay improves even more over the model without, although of course there is randomness involved in the choice of the test set.

For the model where we add a small Transformer encoder block to the baseline with character-based embeddings, we again use the exact configuration used for the baseline.

For QANet, we reduce the model dimension and number of attention heads as mentioned above, but otherwise train with the same configuration used in the original paper – we use Adam with a logarithmic learning rate warmup from 0 to .001 over the first 1000 batches and constant learning rate .001 thereafter, and weight decay parameter $3 \cdot 10^{-7}$. We use a batch size of 32. We additionally use the same dropout and stochastic depth described in the paper. We also experimented with using learning rate .0001 since we observed that the training loss tended to be quite noisy, but did not find it helpful.

We find that over 2.5 million iterations, the dev negative log likelihood fails to go below 3.7 and the dev EM and F1 plateau near 50 and 52, respectively. Needless to say, these results are quite poor and indicate that the model is implemented incorrectly, though we're unable to determine the location of the bug.

For all models, our primary evaluation metrics are the official metrics for SQuAD 2.0. The first of these is just the percentage of answer spans the model gets exactly correct. The second is the F1 score of the predicted answer span relative to the true answer span(s) averaged over all examples.

4.1 Results

Table 1: Dev set model performance (Non-PCE)

Model	EM	F1
Baseline	57.86	61.21
CharEmbeddings 1	59.822	63.145
CharEmbeddings 2	**61.149**	64.14
CharEmbeddings 2 + weight decay	61.099	**64.672**
CharEmbeddings + Self Attn	60.544	63.928
QANet	50.16	53.03

Table 2: Test set model performance (Non-PCE)

Model	EM	F1
CharEmbeddings	59.983	63.643
CharEmbeddings + weight decay	**61.183**	**64.791**

Ultimately, as indicated in Table 1, a relatively simple model consisting of just the baseline augmented with character embeddings performs best on the dev set. Adding an additional self-attention layer after the second RNN encoder layer slightly degrades performance. Our version of QANet performs poorly, but is almost certainly implemented incorrectly. Apart from the poor performance of QANet, these results are roughly in line with what might be expected. Adding character embeddings yields a substantial, but by no means spectacular improvement. Perhaps most surprising is how much a fairly basic and restricted hyperparameter search improved performance, especially on the test set. We qualitatively investigate these results in the next section.

5 Analysis

We focus on comparing the baseline to our best performing model. We expect that including character embeddings might improve performance on questions whether either key pieces of the context, especially the answer span, are out of the vocabulary. For instance, the following context makes heavy use of abbreviations (and rare characters like "#"):

> Other important complexity classes include BPP, ZPP and RP, which are defined using probabilistic Turing machines; AC and NC, which are defined using Boolean circuits; and BQP and QMA, which are defined using quantum Turing machines. #P is an important complexity class of counting problems (not decision problems). Classes like IP and AM are defined using Interactive proof systems. ALL is the class of all decision problems.'

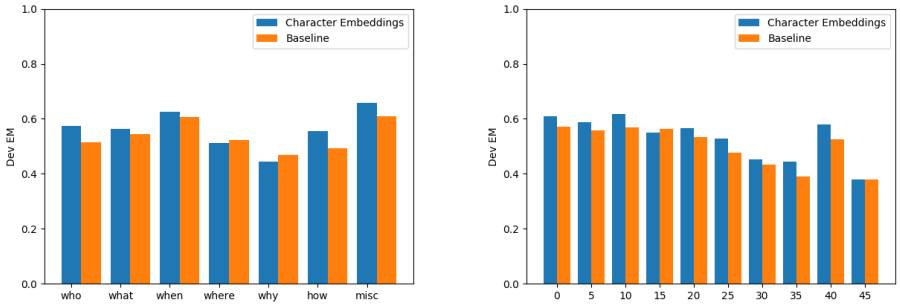
And indeed, the model with character-based embeddings correctly answers $\#P$ while the baseline answers N/A.

The model still struggles in many ways, of course. One of this is in handling co-reference resolution. Given the context

> Genghis Khan united the Mongol and Turkic tribes of the steppes and became Great Khan in 1206. He and his successors expanded the Mongol empire across Asia. Under the reign of Genghis' third son, Ogedei Khan, the Mongols destroyed the weakened Jin dynasty in 1234, conquering most of northern China. Ogedei offered his nephew Kublai a position in Xingzhou, Hebei. Kublai was unable to read Chinese but had several Han Chinese teachers attached to him since his early years by his mother Sorghaghtani. He sought the counsel of Chinese Buddhist and Confucian advisers. Mongke Khan succeeded Ogedei's son, Guyuk, as Great Khan in 1251. He granted his brother Kublai control over Mongol held territories in China. Kublai built schools for Confucian scholars, issued paper money, revived Chinese rituals, and endorsed policies that stimulated agricultural and commercial growth. He adopted as his capital city Kaiping in Inner Mongolia, later renamed Shangdu.

the model incorrectly answers that Kublai was Ogedei's brother instead of his nephew. This presumably stems from a failure to correctly resolve "He" in the third to last sentence.

Although individual examples can be instructive, we also examine the aggregated performance of the baseline and the model with character embeddings on subsets of the dev set. Figure 3 displays the exact match accuracy of each model on examples where the query with each of the words "who", "what", "when", "where", "why", and "how" and queries beginning with any other word. Surprisingly, both models perform best on the "misc" category, which consists of all queries which do not begin with one of the other six words.



(a) Performance by the first word of the query (b) Performance by length of the shortest answer

Figure 3: Performance on subsets of the dev set

We see that the model with character embeddings outperforms in all categories except for queries beginning with "where" and "why". The outperformance on questions beginning with "who", in particular, is intuitive. People's names are unlikely to be in the vocabulary, and being able to learn an embedding for them would seem to be particularly useful for these questions. The underperformance for questions beginning with "why" is somewhat perplexing for the same reason. One might expect that learning embeddings for place names would yield a similar increase in performance.

We also investigate performance based on the length of the shortest correct answer. Questions without an answer are considered to have a shortest answer length of zero. As we might expect, both models tend to have greater difficulty with longer answer spans, though the decline in performance is fairly gradual. There does not appear to be any significant difference in how the models degrade.

6 Conclusion

In this work we described several approaches to SQuAD 2.0, ultimately finding that a model based on BiDAF performed best. This model achieves an F1 score of 64.791 on the test set. We additionally experimented with several models incorporating self-attention; with additional effort and fine-tuning, we believe that these models would likely outperform our RNN based model, though they do not currently. This is one potential avenue for future work.

References

[1] Seo et al. Bidirectional Attention Flow for Machine Comprehension. *https://arxiv.org/abs/1611.01603*, 2016.

[2] Vaswani et al. Attention is All You Need. *https://arxiv.org/abs/1706.03762*, 2017.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding *https://arxiv.org/abs/1810.04805*, 2018.

[4] Peters et al. Deep contextualized word representations. *https://arxiv.org/abs/1802.05365*, 2018.

[5] Yu et al. QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension *https://arxiv.org/pdf/1804.09541.pdf*, 2018.

[6] https://github.com/jadore801120/attention-is-all-you-need-pytorch