Tackling SQuAD 2.0 Using Character Embeddings, Coattention and QANet

Stanford CS224N Default Project: IID Track

Lucas Haarmann
Stanford University
`lh2000@stanford.edu`

Devansh Sharma
Stanford University
`devansh@stanford.edu`

Abstract

Question Answering (QA) systems allow users to retrieve information using natural language queries [1]. In this project, we are training and testing QA models on SQuAD 2.0, a large dataset containing human-labelled question-answer pairings [2], with the goal of evaluating in-domain performance. Using a Bidirectional Attention Flow (BiDAF) model with word embeddings as a baseline, we identified, implemented and evaluated techniques to improve accuracy on the SQuAD task. Our initial experiments, which added character embeddings and a coattention layer to the baseline model, yielded mixed results. Therefore, we started over with a new model using Transformer-style encoder layers, based on the QANet [3]. This model posed many challenges, particularly in adapting to the unanswerable component of the SQuAD 2.0 dataset, and thus did not come close to achieving the performance of BiDAF-based models.

1 Introduction

Question-answering (QA) is a challenging and highly relevant task in the field of Natural Language Processing (NLP). Firstly, QA systems can be a valuable tool for information retrieval, allowing a user to query a large knowledge base in a more intuitive manner [1]. Secondly, the QA task effectively tests the ability of a model to encode meaning. Therefore techniques that are effective at QA may be applicable more broadly to other NLP tasks where capturing semantics is important.

Crucially, a good QA system should be able to distinguish between answerable and non-answerable questions: no answer is preferable to an incorrect answer. Therefore, the dataset we used, SQuAD 2.0, contains 50,000 unanswerable questions in addition to 100,000 human-labelled question-answer pairings [2]. Therefore, to achieve good results on SQuAD, QA systems must have the ability to classify questions as unanswerable, adding an additional challenge.

In this project, we focused on evaluating in-domain performance on the SQuAD 2.0 dataset. Therefore we implemented several different techniques, beginning with modifications to the baseline BiDAF-inspired model [4]. After implementing character-level embeddings and coattention, which replaced single layers of the baseline model, we decided to start over and implement an entirely different model. We settled on the QANet architecture for its demonstrated strong performance on the SQuAD 1.1 dataset [3], making a few changes to handle memory limitations and adapt the model to the SQuAD 2.0 dataset.

2 Related Work

Kim's 2014 paper "Convolutional Neural Networks for Sentence Classification" proposed the use of convolutional neural networks (CNNs) to improve the performance of word embeddings. We applied Kim's methods to generate character-level word embeddings for our paper, albeit in a very different context to Kim's sentence classification task.

Regarding coattention, Xiong et al.'s 2018 paper "Dynamic Coattention Networks For Question Answering" addressed the use of Dynamic Coattention Networks (DCN) for the QA task, and used SQuAD as an evaluation metric. However, while the section on the Coattention Encoder was of relevance to us, we did not implement a full DCN.

A key source for this project was Yu et al.'s 2018 paper "QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension", which we used as a reference for our implementation of a QANet model [3]. However, Yu et al. designed their model for the SQuAD 1.1 task, not the SQuAD 2.0 task with its unanswerable questions. Therefore, the performance of this model on SQuAD 2.0 was uncertain. Secondly, this paper, though providing an indispensable conceptual background, left many implementation details open.

3 Approach

3.1 Baseline

Our baseline model is based on the Bidirectional Attention Flow (BiDAF) model [5]. The code for this model was forked from the CS 224N project starter code repository on GitHub [4]. Crucially, the baseline model differs from Seo et al.'s BiDAF model [5] in that it solely uses word embeddings. These word embeddings are taken from pretrained GloVe word vectors, which are passed through a Highway Network [6] in the embedding layer. The embedding layer then feeds into a bidirectional LSTM encoder layer, then a BiDAF layer and finally a bidirectional LSTM modeling layer.

3.2 Implementation

We can split the changes we made into three categories. The first two are modifications to the baseline code, each replacing a single layer from the BiDAF-based starter code. The third is a significantly different model architecture, sharing only the embedding layer and attention layer with the baseline.

1. **Character-level Embeddings**.

Our implementation retains the GloVe vectors from the baseline model, but concatenates an additional character-level word embedding onto the end of each word embedding. Let e_i represent the embedding for word i, given a GloVe vector g_i and a character-level embedding c_i:

$$e_i = [g_i; c_i]$$

Following Kim, we started with generic character vectors and used a convolutional neural network (CNN) to generate vectors c_i for each word in the input text [7]. Specifically, we used kernels of size 2, 3 and 4 to convolve over the input text. The rationale behind this kernel size choice was to capture important sub-word strings like prefixes and suffixes. After applying each of these 3 single-dimensional convolution layers, we used a ReLU activation function and "max-over-time" pooling [8] to produce a single scalar value from each kernel size. Therefore, a character-level embedding for a single word, c_i, has length 3.

After concatenation, we followed Seo et al. in applying a dropout layer and the passing the combined embedding through a Highway Network [6].

2. **Coattention**

Coattention uses a dual-level attention calculation to compute attention between the context and question representations: after the standard context-question and question-context attentions are calculated, a weighted sum is taken using the two attentions. To implement a coattention layer in place of the baseline bidirectional attention layer, we followed Xiong et al. [9].

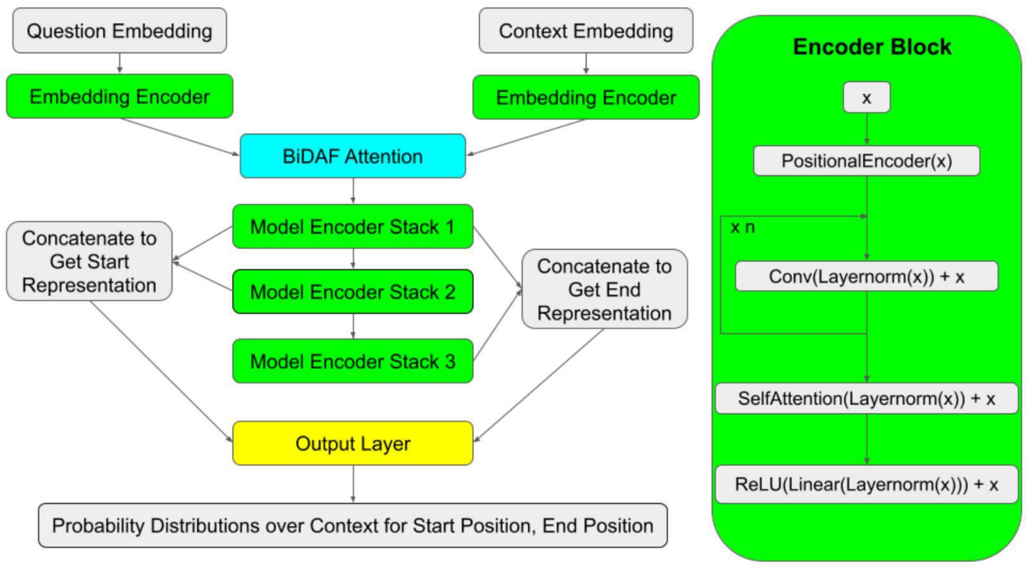
We began by applying a linear layer to the question representation $q \in \mathbb{R}^{qlen \times h}$, applying a tanh nonlinearity to the result, which we will refer to as $q' \in \mathbb{R}^{qlen \times h}$. Then we concatenated randomly-initialized sentinel vectors onto the question and context representations to get $q' \in \mathbb{R}^{qlen+1 \times h}$ and $c \in \mathbb{R}^{clen+1 \times h}$. This allowed a word in the context to attend to none of the words in the question and vice-versa.

Next, we computed an affinity matrix between the context and question: $cq'^T = A \in \mathbb{R}^{clen+1 \times qlen+1}$. Computing the softmax distribution over columns of A produced weights α which were multiplied with the question q' for context-question attention a. Likewise, computing the softmax distribution over rows of A produced weights β which were multiplied with the context c for question-context attention b. The context-question weights α were then multiplied with b to produce a second-level attention s.

Finally, the last dimension of s was concatenated with the last dimension of a to produce a series $[s_1; a_1], .., [s_{clen}; a_{clen}]$, where $[s_{clen}; a_{clen}] \in \mathbb{R}^{2h}$, which was fed into a bidirectional LSTM with dropout to produce the coattention output.

3. **QANet-Based Model**.

The QANet architecture is taken from Xiong et al. [3]. Specific to our implementation is the use of the character embeddings described in part 1, the reuse of the provided BiDAF Attention module from the baseline model, the use of a single linear layer with a Rectified Linear Unit (ReLU) activation for the output of the encoder block and a custom output layer for the SQuAD 2.0 task.



Encoder Block

Shown in green on the diagram above, the encoder block is a critical part of the QANet. Following Vaswani et al., positional encodings are added to the input x [10]. Specifically, we based our implementation of positional encodings on a PyTorch implementation in a tutorial for sequence-to-sequence modeling using Transformers [11], only making modifications to get the positional encodings to work well with Pytorch cuda Tensors.

x is then passed through a series of n 1D convolution layers, each with a fixed kernel size of 7. Each convolution layer is preceded by layer normalization [12]. At the end of each convolution layer, the convolution output is added to x such that $x = \text{Conv}(\text{Layernorm}(x)) + x$.

After n loops through the convolution layer, layer normalization is applied again and the encoder block moves on to a Multi-Head Self-Attention layer, again inspired by Vaswani et al. [10], with a fixed 10 heads. We initially experimented with our own implementation of Multi-Head Self-Attention, however we encountered memory issues due to the size of the matrix multiplication involved. For greater memory efficiency, we therefore decided to use PyTorch's nn.MultiheadAttention function (PyTorch v1.3.0). This assumes that the number of heads is a factor of the hidden layer size, therefore the hidden layer size must be a multiple of 10 (for most tests, we used 100).

The attention output $\text{SelfAttention}(\text{Layernorm}(x))$ is again added to the previous state x. Finally, layer normalization is applied one last time, the attention output is fed into a simple linear layer with a ReLU activation function, and the linear layer output is added to the previous state.

Overall Model

Overall, the model begins by passing the context and question embeddings separately through encoder blocks with $n = 4$ convolutional layers each. Unlike Yu et al., we fed the encoded outputs into a

BiDAF attention layer, which was reused unmodified from the starter code. The BiDAF attention output is then passed sequentially through 3 model encoder stacks, which we will label as Stack 1, Stack 2 and Stack 3. Despite the labels, these 3 stacks share the same parameters. Each stack consists of 7 sequential encoder blocks. Each encoder block has $n = 2$ convolutional layers. The Stack 1 and Stack 2 outputs are concatenated to form a start representation, and the Stack 1 and Stack 3 outputs to form an end reprsentation. Both concatenations occur over the hidden-size dimension, so that the concatenated result has effectively doubled its hidden size. Finally, the start and end representations are fed into an output layer. The output layer applies a linear transformation to each, collapsing the hidden size to size 1 using weight matrices of dimension $W_1, W_2 \in \mathbb{R}^{\text{hidden-size},1}$. The softmax distribution is then computed over the context-length dimension, resulting in probability distributions for the start token and end token.

4 Experiments

4.1 Data

We used the provided CS224n SQuAD IID train, dev and test datasets in our project. These sets are taken from the publicly available SQuAD dataset [2], however we complied with the project guidelines in solely using the provided sets to ensure a level playing field for comparing models.

Specifically, the provided datasets consist of labelled question-context-answer groupings. In each case, the context is an excerpt from a Wikipedia article, the question is a human-generated question based on the article and the answer is a human-labelled section of the context that answers the context. The goal of our model is to predict the answer based on the question and the context. Importantly, the gold /correct answer will always be a segment of text in the context. Therefore there is no language modelling component to the task: our model simply has to predict a start and end position of the answer in the context. In some cases, the question is unanswerable: the correct answer in this case is "N/A".

4.2 Evaluation method

We used all the following quantitative metrics when evaluating on the dev set.

- **NLL:** Sum of Negative Log-Likelihood Loss for the start and end location probability distributions, P_{start} and P_{end}. For instance, if the correct (gold) start location is s and the true end location is e:

$$NLL = -\log(p_{start}(s)) - \log(p_{end}(e))$$

- **F1:** Harmonic mean of Precision ($\frac{tp}{tp+fp}$) and Recall ($\frac{tp}{tp+fn}$) for words in the predicted answers.
- **EM:** Exact Match. Proportion of predicted answers that precisely match the gold answer, word for word. EM is a less forgiving metric than F1, hence scores are always lower.
- **AvNA:** Answer vs No Answer. This metric treats SQuAD 2.0 as a binary classification task between answerable and non-answerable questions. Therefore, this metric provides the answer/no-answer classification accuracy.

Qualitatively, we also considered the convergence behavior of our models and used Tensorboard to look directly at predicted answers from excerpts of the dev set.

Evaluation on the test set focused solely on EM and F1 scores.

4.3 Experimental details

Initially, we ran the baseline BIDAF model with the learning rate of 0.5, and it took about eleven hours to train. The results were quite impressive, and so we began with modifications to this model.

For the character embeddings, which used a one-dimensional convolutional neural network, training took approximately the same time as the baseline model to train and results were very similar to the baseline.

A slightly more complicated effort to implement our own coattention layer in place of the standard BiDAF attention followed. The resulting model took marginally longer to train versus the baseline model, but performed significantly worse.

In order to get better results, and to experiment with a much more comprehensive restructuring of the model, we decided to implement the QANet architecture. Initially, the model seemed to be running on our local machines, but it was producing a host of device conversion errors on our VMs. Therefore, we spent a lot of time adding ".cuda()" and ".cpu()" to effectively convert our various parameters to the GPU and back.

However, this led to several memory issues as CUDA was overflowing due to the memory-intensive matrix multiplication operations in our self-attention module. Therefore, we decided to update our Pytorch version from 1.0.0 to 1.3.0, which includes a library implementation of multi-head self-attention. Using the Pytorch version of multi-head self-attention helped bring our model back on track and we were able to start training.

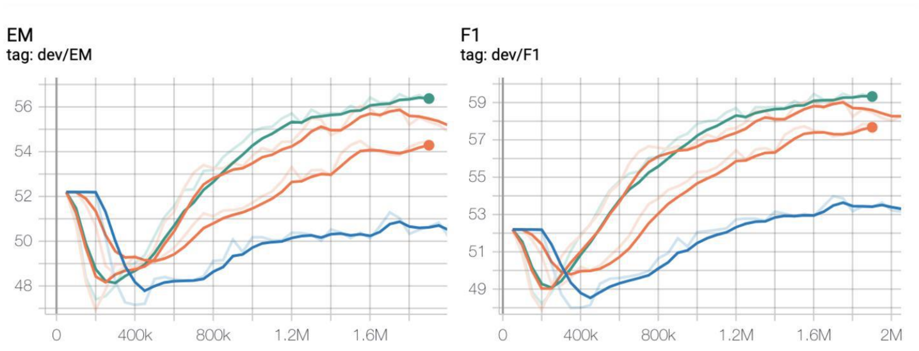
We started with a small hidden layer size of 40 and the initial learning rate of 0.5, but gradually pushed the hidden layer size to 50 in order to get better dev set results. In order to make the training faster, we decreased the number of encoder blocks from 7 (the suggested number in the QANet paper [3]) to 4. However, we still did not get satisfactory outcomes. The model was consistently predicting 'N/A' for every question. Thus all the dev metrics were staying constant throughout training. Therefore, we concluded that we had to make some changes to our parameters.

However, increasing the number of parameters was leading to memory overflow errors. Therefore, we halved the batch size from 64 to 32, ramped up the hidden layer size to 100, increased the number of encoder blocks back to the recommended number of 7 and increased the learning rate to 0.8.

We then found that training was consistently stopping during dev set evaluation at 200,000 iterations due to the OS killing the process, even when we tried reducing the size of the model. We were able to resolve this issue by cutting down the number of workers to 0 on the dev set data loader. At last, we were able to train the QANet with our desired hyperparameters.

4.4 Results

The results are definitely not what we expected. We expected a significant increase in the performance of the models once we changed over to a QANet model, but that did not happen. This was despite initial indications on TensorBoard, seen below, which indicated that NLL, F1, EM, and AvNA were better for BiDAF with character embeddings up to around 15 epochs.



Key from top to bottom of graph:

Char BiDAF	Baseline	Word Coattention	Char Coattention
🟩	🟧	🟧	🟦

Ultimately, however, the character embeddings faltered at 30 epochs, ending up slightly below the baseline model in performance on the dev set. However, the character embeddings performed relatively well on the test set, demonstrating the potential of this method for improved out-of-domain performance. The use of the coattention layer in place of the baseline BiDAF layer failed to offer any improvements, and performed particularly poorly when combined with the character embeddings. The

QANet, due to consistent issues with training and prediction of unanswerable questions, performed the worst by a large margin.

Dev Set Evaluation Results				
Model	NLL	F1	EM	AvNA
BiDAF (Word Embeddings) - 30 EPOCHS	3.07	61.50	58.12	67.97
BiDAF (Char Embeddings) - 30 EPOCHS	3.13	60.93	57.57	67.52
Coattention (Word Embeddings) - 15 EPOCHS	3.19	57.42	54.16	64.64
Coattention (Char Embeddings) - 15 EPOCHS	3.49	54.11	51.24	62.75
QANet (Char Embeddings) - 15 EPOCHS	8.77	42.56	42.28	51.17

Test Set Evaluation Results		
Model	F1	EM
BiDAF (Char Embeddings)	61.63	58.02
QANet (Char Embeddings)	12.72	11.70

5 Analysis

For our character embeddings, the mixed results are likely the result of our decision to add just 3 convolutional layers, with kernel sizes ranging from 2 to 4 characters. This resulted in a very small dimension character embedding output, did not result in any notable improvements when concatenated onto the word embedding. Ultimately, the use of the extra character embeddings even resulted in a slight decrease in performance after the full 30 epochs, perhaps because, with very few convolutional layers and a small embedding size, the character embeddings were potentially noisy. A larger character embedding size, closer in size to the word embeddings, might address this, at the expense of additional training cost.

Notably, however, our character embeddings achieved slightly higher F1 scores on the test set than on the dev set. This seems to indicate that the character embeddings were somewhat successful in helping the model generalize outside of the training and dev sets. This is in keeping with the theoretical promise of character embeddings to capture sub-word meanings, allowing the model to cope better with out-of-domain words.

The use of coattention as a direct replacement for the BiDAF attention brought losses in all evaluation metrics versus the baseline model. The particularly strong reduction for coattention with character embeddings may be an exacerbation of the effects described in the previous paragraph. The coattention models, however, despite their additional complexity and their use of a bidirectional LSTM on their output, were only marginally slower in training when compared to the baseline model.

As discussed in Experiments, the QANet consistently suffered from its tendency to excessively classify questions as unanswerable. This was revealed by analysis of excerpts of text from the dev set, which were consistently being labelled N/A even after may epochs of training. In comparison, the BiDAF model, after an initial period of always predicting N/A to rapidly reduce the loss function, learned to start predicting real answers after just a few epochs. This made the QANet a very difficult model to train, and we were not able to find a satisfactory set of hyperparameters and a suitable training process to enable this model architecture to break out of its tendency to make N/A predictions. As noted in our Related Work section, the unproven nature of the QANet model on the SQuAD 2.0 task brought risks. Ultimately, with our given time and computational resources, we were not able to successfully adapt this model to the newer dataset with its unanswerable questions.

6 Conclusion

Our work has provided many learnings about which models are and are not successful on SQuAD 2.0. Firstly, we demonstrated both the potential and the limitations of low-dimension character embeddings. Though quick to train, this approach yielded mixed results. We believe this technique might be improved with the use of a greater number of convolutional layers to produce larger embeddings, though this would inevitably come at the expense of training cost. We also demonstrated that the replacement of a BiDAF attention with a Coattention layer, although surprisingly fast to train, did not produce satisfactory results.

As far as the QANet model architecture is concerned, we learned that far greater changes would be necessary to achieve top-level results on the SQuAD 2.0 task. As expected, this Transformer-inspired model was very memory-intensive to train, and a great deal of work was necessary merely to begin training. More surprising were the poor results that followed training. Slow training and a tendency to always classify questions as unanswerable were the downfall of QANet on the SQuAD 2.0 task.

One potential response to these problems can be found in Aubet et al.'s Enhanced Question Answer Network (EQuANt) [13], which is optimized for the SQuAD 2.0 task. The EQuANt features an "Answerability Probability Block" that outputs the probability that a question is answerable independently from the base QANet's start and end probability distribution outputs (which are retained). Unfortunately, we did not discover this model in time to implement it for this project.

In conclusion, the BiDAF model with character embeddings was our best-performing model, but we believe that there remains significant room for improvement both with the character embeddings and with the QANet model on the SQuAD 2.0 task.

References

[1] Bolanle Ojokoh and Emmanuel Adebisi. A review of question answering systems. *Journal of Web Engineering*, 2019.

[2] Squad 2.0: The stanford question answering dataset. `https://rajpurkar.github.io/SQuAD-explorer/`. Accessed: 2021-02-16.

[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, 2018.

[4] Mina Lee. squad. `https://github.com/minggg/squad`, 2020.

[5] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *International Conference on Learning Representations (ICLR)*, 2016.

[6] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. In *ICML)*, 2015.

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

[8] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch, 2011.

[9] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering, 2018.

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

[11] PyTorch. Sequence-to-sequence modeling with nn.transformer and torchtext. `https://pytorch.org/tutorials/beginner/transformer_tutorial.html#define-the-model`, Accessed 2021/03/16.

[12] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[13] François-Xavier Aubet, Dominic Danks, and Yuchen Zhu. Equant (enhanced question answer network), 2019.