Hmmm. That's a Good Question!

Stanford CS224N Default IID Project

Brian Rhindress
Department of Computer Science
Stanford University
`brhindre@stanford.edu`

Abstract

In this paper, I attempt to replicate the QANet architecture for the IID QA task. To maximize informational value within the training corpus, I generate new questions using the T5-small model, increasing the dataset by 72%. Through testing of 8 models of increasing complexity, I inherently design experiments that serve as ablation tests. This experimentation yields modest gains over two vanilla baselines, BiDAF and QANet, respectively, using simple techniques such as character-level embeddings and positional encoding.

1 Key Information

I, Brian Rhindress (brhindre@stanford.edu) am pursuing a solo project, using the default IID SQuAD track. I am not sharing this project with another course.

2 Introduction

While the QANet architecture and backtranslation data augmentation approach was state of the art at the time of publication in 2018, much training data was left on the table by not including Question Generation (QG) approaches. First, the QANet model architecture (and BiDAF) is designed only for learning of answer spans within the contexts. These are not seq2seq models that utilize training data to learn language structure, which could conceivably help create more robust reading comprehension models. Secondly, the original QANet paper used backtranslation to augment data. However, backtranslation does not maximally mine the context for potential questions. Rather, it rearranges the existing context and questions. Overall, we are left wondering – how might performance improve, if we were to harness the power of QQ for QA?

I initially sought to address both of these questions. However, after spending a bit too much time re-purposing CS224N Assignments 4 and 5 to create a QA/QG seq2seq model, I ultimately decided to focus on QG-based data augmentation with the original QANet architecture. I attempted to re-construct QANet, precisely as described in the paper (I eventually made modifications to get a working implementation). For data augmentation, I used the T5-small QG model from HuggingFace on the SQuAD 2.0 training data, generating over 90,000 new questions from the dataset and increasing the training size by 72%. With this set up, I tested several baselines and models for comparison including the BiDAF starter code, BiDAF with character embeddings, QANet Vanilla, QANet with Positional Encoding/Dropout/Full Convolutional Layers in Output Encoder Blocks, and QANet with QG-styled Data Augmentation. Overall, my working implementations performed a bit below expectations, but I was able to replicate certain smaller findings such as the utility of additional embedding types and transformer speedup.

3 Related Work

The QANet paper, published in 2018 by Yu et al, was one of the first successful efforts to use transformers for end-to-end QA tasks. [1] The primary benefits of using transformers for QA are 1) training is faster for time-critical tasks and 2) with more available time, training data can be augmented for improved performance. The QANet team used this available training capacity to utilize a data augmentation process, called "Backtranslation," that uses a machine translation model to convert training examples to another language, then back to English to grow the corpus. Overall, QANet trains 3-13x faster and infers 4-9x faster, while achieving the best published F1 score at the time of publishing, of 84.6,an increase of 2.6.

There have also been many recent advancements in Question Generation tasks. Wang et al. (2017) [2] implement an RNN-style encoder-decoder system that integrates QA/QG in one seq2seq model using a control bit to toggle between tasks. In order to balance between QG and drawing on spans in the context, this implementation utilizes a pointer-softmax in the deocder layer, which allows selective answer/question generation from either a vocabulary or context spans (Gulcehre et al., 2016). [3] Zhu et al. (2019) [4] take QG a step further, augmenting the SQuAD 2.0 dataset by generating unanswerable questions with a pair-to-sequence model, generating 1.9 absolute improvement in F1 score with the BERT-base model. Thus, there is precedent for both integrated QA/QG models as well as QG-based data augmentation, with both approaches demonstrating performance enhancements over existing baselines.

Naturally, each of these models draw on more foundational reading comprehension and encoder-decoder archetypes such as self-attention, positional embeddings (Vaswani, et al., 2017) [5], and context2query (and query2context) attention (Seo et al., 2016). [6] I draw on these standard implementations for my home-grown QANet implementation.

4 Approach

4.1 Brief QA/QG Model Attempt

Before pivoting purely to the replicating QANet with data augmentation, I briefly constructed a seq2seq model based on the Neural Machine Translation (NMT) model from CS224N Assignment 4. The key difference in my model was instead of reading in a singular sentence, it read in both a question and a context, with the decoder attention focused on the context. The loss function I used was the same as the NMT model, the log probability of generating true target words (see Assignment 4 or my deprecated source code for more details). Additionally, I used a small beam search of 2, to improve lengthy evaluation, which also reduced prediction quality. I did not implement pointer-softmax, one of the key structures used by Wang et al. (2017), [3] which may have affected performance. Overall, due to lengthy development time and poor initial results (deviating from the Default QA IID starter code), I pivoted back to implementing QANet with augmented data.

4.2 BiDAF Baselines

The task pursued was minimizing negative log-likelihood of the predicted start and end probabilities of a given answer span. For my primary baseline, I used the BiDAF starter code provided in the IID Final Project repository. This model includes a standard SQuAD architecture including 1) an Embedding Layer, 2) an Input Encoder Layer, 3) the Attention Layer, 4) the Model encoder layer, and 5) the Output layer. I detail more about each of these layers with respect to QANet, in the next subsection.

I enhanced this code by adding character-level embeddings. To do this, I simply took the first 16 characters in a given word, then took the max in each character embedding dimension as the full subword embedding. I concatenated this with each word embedding in the embedding direction as so,

$$w_i \in \mathbb{R}^{1x300}$$
$$c_{i,j} \in \mathbb{R}^{1x64}$$
$$c_i = \max_j c_{i,j} \in \mathbb{R}^{1x64}$$
$$wc_i = concat(w_i, ci) \in \mathbb{R}^{1x(300+64)}$$

4.3 QANet Implementations

4.4 QANet

I built my QANet model using the BiDAF starter code framework. The primary differences between BiDAF and QANet are in the input and output encoder layers, which utilize stacked encoder blocks in place of the RNNs used in BiDAF. See Figure 1 below to see the full QANet architecture, under which I detail my implementation.

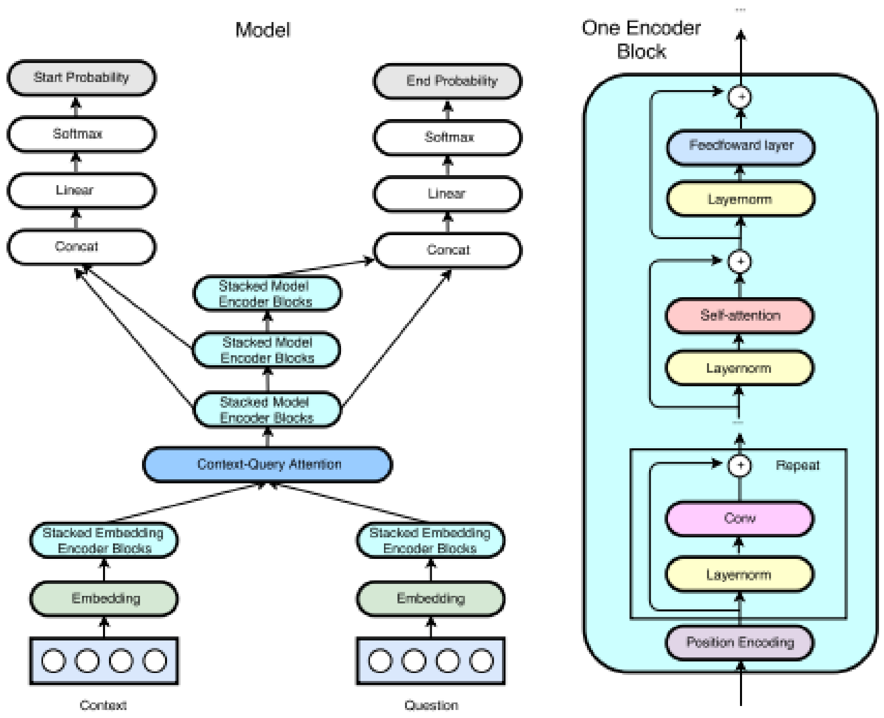

Figure 1: Architecture for QANet (Yu et al., 2018).[**?**]

Each layer was constructed as follows (including variations).

1) The Encoder layer is exactly the same as that of the BiDAF implementation, using concatenation of word and character-level embeddings, followed by a 2-layer Highway Encoder. Each the context and question are passed through the embedding layer separately, however weights are shared.

2) The input encoder level begins with positional encoding, using the absolute (non-learned) position embeddings used by Vaswani, et al., (2017). [5] After positional encoding, the full word-character-position embeddings are layer normalized, then passed through a single Separable Depthwise Convolutional layer. For this, I utilize a two 1D Convolutions – a depthwise, followed by pointwise layer – in a segment of code I cite by Bang Liu (2018). [7] Keeping with the QANet paper, convolutional blocks transform the embedding dimension down to 128 dimensions. There are four convolutions in the input encoder layer. Next, self-attention is implemented, using the Causal Self-Attention from CS224N Assignment 5. Masking is turned off, since it is not necessary in the encoding phase. Lastly, the encoder includes a fast-forward layer that maintains 128 dimensions. Each of these stages includes a pre-transformation layer normalization and a post-transformation path to the pre-normalized layer. These layers consist of an encoder block, and only 1 is used in this level. Each the context and question pass through encoder blocks separately, but share weights.

3) The Attention Layer utilizes the BiDAF Context-Query Attention, as is, utilizing the tri-linear function (Seo et al., 2016). [6] The input of this layer are two variable-length encoded sequences

of dimensions (context/query, d = 128), which are transformed into an encoding of length (context length, 128*4).

4) The Output encoder layer is similar to the input layer in that it uses stacked encoder blocks with the same configuration as 2), with a few exceptions. The number of convolutions per layer is 7 (this differs from the QANet configuration of 2), and the number of stacked blocks is 7. Further, three sets of these blocks are stacked together. The layer of the first and second blocks are concatenated and passed to a "start position" output layer and the outputs of the first and third blocks are passed to an "end position" output layer.

5) The Output layer is the same as that implemented in the BiDAF starter code. After computing a softmax over a feed-forward layer using the concatenations from 4) to generate the probability of the answer starting and ending on each word in the context, the model produces the loss function,

$$loss = -\log p_{start}(i) - \log\left(p_{end}(j)\right)$$

At inference time, I keep the BiDAF model implementation, which chooses the answer span that maximizes the product of $p_{i,start} * p_{i,end}$.

As shown in the experiment section, several configurations were tested, varying some architectural parameters. I also implemented several implementations of dropout (p=0.1), including between every layer, including convolutions, as well as only between the major layers.

4.5 Data Augmentation

For data augmentation, I used the T5-small QG model from the HuggingFace API.[8] The SQuAD 2.0 training dataset is split into 442 articles, composed of a total of 19,006 contexts, in which there began a total of 130,067 questions.. Each context was mined for new potential questions, yielding 93,145 new questions. Duplicate questions were somewhat rare, only occurring 1,269 times (all were removed). Overall, this question generation process proved efficient, yielding nearly 5 new questions per context and increasing the overall training corpus by approximately 72%. Augmented data was neatly integrated into the setup and training pipeline for experimentation, as shown below.

5 Evaluation Method

5.1 Experimental Details

As shown in Table 1 below, I tested 8 different models, with 3 BiDAF configurations and 5 QANet configurations. See the table for full configuration details. These models were developed sequentially with the hope of establishing successful baselines and then adding complexity. In terms of development timeline, BiDAF models were established first, followed by the QANet models, in roughly the order described in Table 1. In general, hyperparameters were fixed to the Default IID starter code with 30 epochs and Adam optimization with a learning rate of 0.5. For QANet models, batch sizes were decreased from 64 to 16 to help prevent memory overflow. Additionally, when augmenting the dataset, the number of epochs was increased to up to 35 in the case of BiDAF with character embeddings.

5.2 Results

The best performing model on the dev set was BiDAF with character embeddings, with an EM score of 57.86 and an F1 score of 61.02. This was the primary model used on the test set, where it achieved an EM score of 52.325 and an F1 score of 56.769. Each of the three BiDAF implementations outperformed the QANet implementations, suggesting that my QANet implementations did not nearly meet the expectations of top performing models in the original paper.

Examining the QANet model results more closely, the vanilla model and subsequent version with positional embeddings did not perform nearly as well as the BiDAF baselines. However, they did show some promising results, as their EM and F1 scores followed the same training pattern, first beginning at 52.19 (due to predicting all "No-Answer"s), then decreasing as the models began to make real predictions. See Figure 2 for this trend. This is also clearly shown by the monotonically

Model Name	EM	F1	Hyperparameters	Time to 3M Steps
BiDAF Baseline	56.16	59.74	lr: 0.5, adam optimzation, epochs: 30, dropout: 0.2, ema: 0.999	8h 43m
BiDAF w/ Char. Embeddings	**57.86**	**61.02**	lr: 0.5, epochs: 30, dropout: 0.2, ema: 0.999, full training set	8h 51m
BiDAF w/ Char. Embeddings + Data Augmentation	54.7.86	58.8	lr: 0.5, epochs: 35, dropout: 0.1, ema: 0.999, full training set + QGs	2h 29m
QANet Vanilla	41.29	43.47	lr: 0.5, epochs: 30, dropout: 0.2, ema: 0.999, full training set	5h 28m
QANet Positional Embeddings	44.47	45.92	lr: 0.5, epochs: 30, dropout: 0.2, ema: 0.999, full training set, batch size = 16 (due to memory constraints)	8h 14m
QANet Full Convolutions + Max Dropout + Data Augmentation	52.19	52.19	lr: 0.5, epochs: 30+, dropout: 0.1, ema: 0.999, full training set, batch size = 16 (due to memory constraints)	14h 11m
QANet Full - Fewer Convolutions	52.19	52.19	lr: 0.5, epochs: <15, dropout: 0.1, ema: 0.999, full training set, batch size = 16 (due to memory constraints	N/A
QANet Full - Less Dropout	52.19	52.19	lr: 0.5, epochs: <15, dropout: 0.1, ema: 0.999, full training set, batch size = 16 (due to memory constraints	N/A

Table 1: Experiments and Results

increasing AvNA scores for these two QANet models. However, unlike the BiDAF models, the EM and F1 scores never improved. At the time, I hypothesized that this could be the result of one of two phenomena: either the model needed more training, or more complexity. So, I opted for both, perhaps too soon.

The next developed model was far more complex – this time including 7 convolutional layers per encoder block (the paper specified just two) and 7 encoder blocks, as well as dropout between all layers. I had planned to train this model on the entire augmented dataset for up to 120 epochs. However, after only a fraction of epochs, the loss function did not decrease and the EM/F1 scores remained at 52.19. There was a bug somewhere in the model, likely from introducing too much complexity all at once.

5.3 Analysis - Ablation and Debugging

To pinpoint the issue, I began ablation testing to try and eliminate one of two potential sources of error introduced from the simple baselines to the more complex model. First, I removed 6 out of 7 of the stacked encoder blocks in the output encoding layer. Next, I removed all dropout layers between convolutions. Unfortunately, the problem persisted throughout both of these tests, as shown by the 52.19 EM and F1 scores on the last two models. I did perform one additional test to pinpoint the issue – over-fitting the full model to an 8-example training set to make sure that outputs are reasonable. Achieving perfect classification in about 750 epochs, the problem was not as simple as a bug in the general train/evaluation pipeline. Rather, there was some training noise and error introduced somewhere in the architecture. Given the excess complexity to the convolutional set up after the initial two baselines, I believe the encoder blocks were likely the issue.

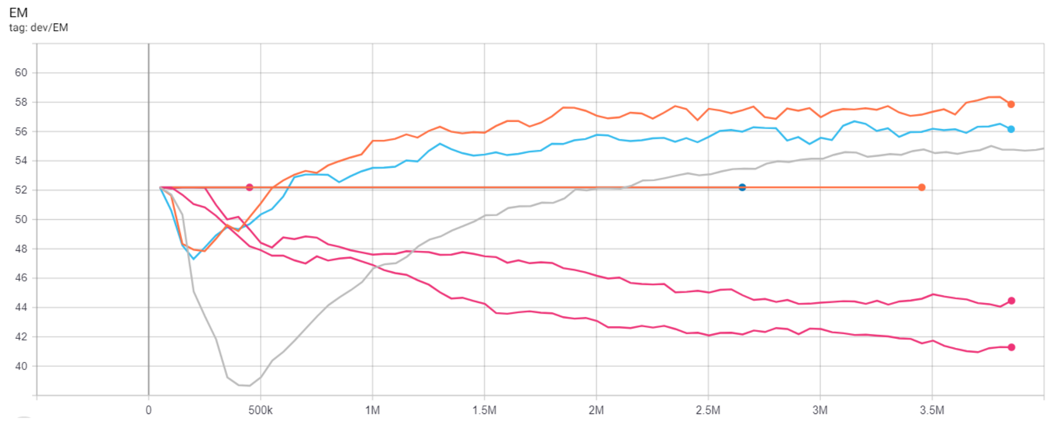

Figure 2: EM on Dev Set (F1 shows same pattern and orderings). From top to bottom: BiDAF w/ Char. Embeddings, BiDAF Baseline, BiDAF w/ Char. Embeddings + Data Augmentation, QANet Full, QANet w/ Fewer Convolutions, QANet w/ Less Dropout, QANet w/ Positional Embeddings, Vanilla QANet

While my overall performance for QANet is lower than I would have expected, there are still insights to be learned from these experiments. First, character embeddings improved absolute EM/F1 scores by 1.7/1.28 respectively on BiDAF. Positional embeddings improved EM/F1 by 3.17/2.52 on QANet. And both of the successful QANet implementations were faster in steps than BiDAF, with the QANet Vanilla making it through 3M iterations in 3 h 15 minutes quicker than the BiDAF baseline. Alas, this came at the consequence of lower performance as well. Perhaps it would have been constructive to train this model for an additional 3 hours for a more fair comparison. Either way, this does show that transformers are faster than RNN-based models. Lastly, there is currently no evidence from my results to show that QG-inspired data augmentation improves performance. The shorter time to train on BiDAF with data augmentation is merely due to an upgraded GPU, not the model.

5.4 Analysis - Sample text

The quantitative analysis above is confirmed by visual inspection of the text output. As expected from the stagnant EM/F1 scores, the output for the most complex models are all "N/A." However, there are some promising results for the Vanilla and Positional Embeddings QANet models that are worth highlighting. On two date questions, the model seems to be performing decently well. On one occasion it answers "the 1950s" when the answer should just be "1950s" and on another it answers "1 September 1939" instead of "September 1939." In some other cases it still predicts NAs where there should be answer spans. The more challenging case is when the QANet predicts a reasonable answer span, when the answer is actually NA. For instance, the model predicts that "90%" of electrical power in the US is made by generators, while that is actually the amount produced by various heat sources. Obviously this answer is incorrect, but the model is not totally off-base with this prediction. All together, this qualitative analysis suggests that the basic QANet models I developed were on a path to learning, and perhaps could have benefitted from more training, or a more gentle increase in complexity.

6 Conclusion

Overall, I find here the modest benefits to EM/F1 provided by character embeddings, positional embeddings, and some training speedup due to transformers. I additionally provide a framework for improving the Question Answering task by integration of Question Generation techniques.

I leave this project having learned many techniques and good practices for using deep learning for NLP tasks. If I could repeat the project, I'd likely do a better job ironing out a stable model to build on, perhaps replicating a model from a paper with code, rather than trying to build a model totally from scratch, or from pieces of disparate systems. Additionally, I would be more careful about adding complexity before slowly and surely validating each step of the neural network architecture. While

I began with this approach, the big jump I made from the QANet with Positional Encoding to the many-convolution/dropout/data augmentation version introduced some bias into my model. Finally, I would probably be a bit more careful with my scoping as a one-person team, trying not to attempt two separate tasks (building a novel QA/QG system and also testing a new data augmentation method). These are all lessons I will carry with me into the next project.

References

[1] 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. volume abs/1804.09541, 2018.

[2] Tong Wang, Xingdi Yuan, and Adam Trischler. A joint model for question answering and question generation. volume abs/1706.01450, 2017.

[3] Çaglar Gülçehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. Pointing the unknown words. *CoRR*, abs/1603.08148, 2016.

[4] Haichao Zhu, Li Dong, Furu Wei, Wenhui Wang, Bing Qin, and Ting Liu. Learning to ask unanswerable questions for machine reading comprehension. volume abs/1906.06045, 2019.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[6] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[7] Bangliu/qanet-pytorch. As a part of a QANet replication, available at `https://github.com/BangLiu/QANet-PyTorch/blob/master/model/QANet.py`,url=https://github.com/BangLiu/QANet-PyTorch/blob/master/model/QANet.py, journal=GitHub, author=BangLiu, year=2018, month=Sep.

[8] Hugging face t5-small transformer for multi-task qa and qg. Available at `https://huggingface.co/valhalla/t5-small-qa-qg-hl`.