Reimplementing QANet

Stanford CS224N Default Project

Adrian Ng
Department of Computer Science
Stanford University
adriannng@stanford.edu

Abstract

QANet achieved state-of-the-art performance back in 2017. It is based on key NLP ideas such as self-attention and convolution. One key QANet feature is it has up-to-13x speedup in training and inference time by replacing RNNs (Recurrent Neural Networks) with convolution-based layers such as Encoder Block. My project goal is to develop a deep understanding of QANet by re-implementing the model from scratch and re-evaluating model performance. My evaluation results show that, while my implementation has the expected model accuracy improvement, it has an unexpected 4x slowdown in training/inference time compared to the baseline. My analysis shows RNN indeed runs 2x faster than an Encoder Blocker. This brings up an observation that convolution-based encoders are not necessarily faster than RNNs like conventional wisdom suggests and the execution time boils down to the implementation.

1 Key Information to include

- Mentor: None
- External Collaborators (if you have any): None
- Sharing project: None

2 Introduction

Machine reading comprehension and automated question answering (Q&A) are two sub-areas of NLP (Natural Language Processing). Over the past years, significant progress has been made in these areas. The task of Q&A involves building a model such that given a short paragraph and a question, the model can output the answer location in the given paragraph. In other words, we need to teach a model to "read" a paragraph, "comprehend" and "answer" it.

QANet [1] is a very successful model that achieved state-of-the-art performance back in 2017. It is based on key NLP ideas such as self-attention and convolution. One key QANet feature is it has up-to-13x speedup in training and inference time by replacing RNNs with convolution-based layers such as Encoder Block.

My project goal is to develop a deep understanding of QANet by re-implementing the model from scratch and re-evaluating model performance. In this paper, I show that my implementation of QANet has the expected model accuracy compared to the baseline. However, it has an unexpected 4x slowdown in training/inference time. This is contradictory to the 13x speedup that I was expecting from QANet. My analysis shows that this is a result of the fact that RNN indeed runs 2x faster than an Encoder Blocker in my implementation. This brings up a point that convolution-based encoders are not necessarily faster than RNNs unlike what conventional wisdom suggests.

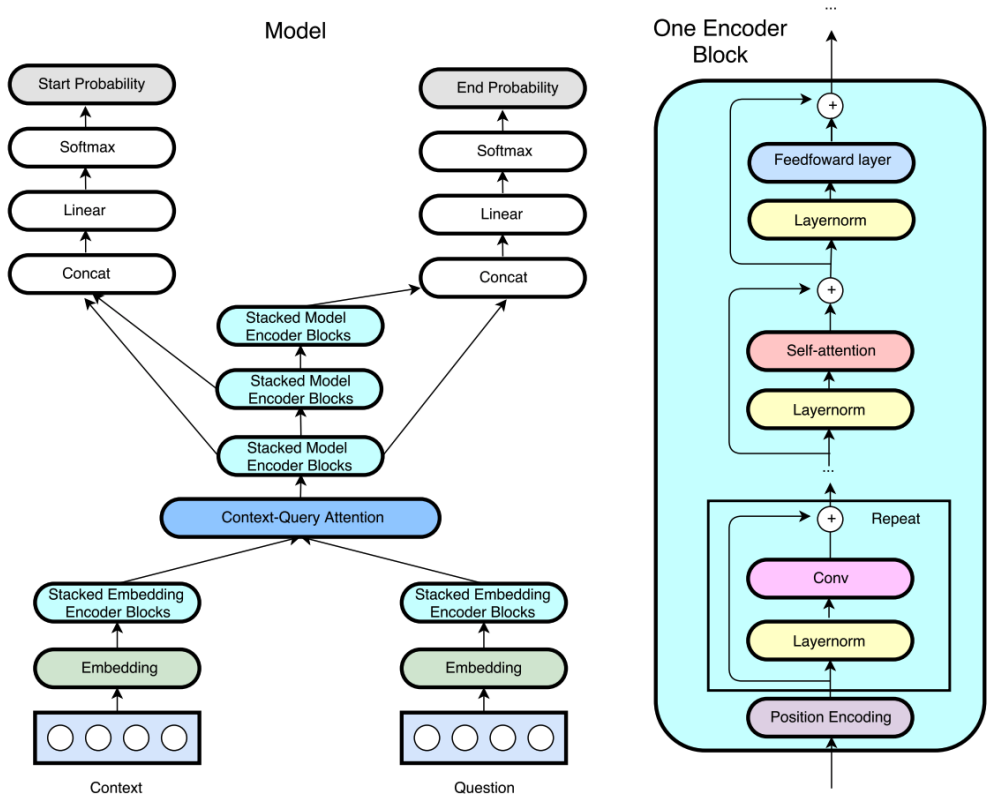
3 Related Work

Over the past years, significant progress has been made in Q&A models on challenging datasets like SQuAD [6]. BiDAF [2] is a classical neural Q&A model. It uses a bi-directional RNN layer to encode both a paragraph and a question, pass the encoded paragraph and question through an attention layer and a final modeling layer to output the answer. The QANet model [1] borrows ideas from the Transformer model [4] to replace the RNN layer with an Encoder Block, which is built based on a series of convolution layer and self-attention layer. Details for Encoder Block is shown in figure 1.

4 Approach

As shown in figure 1, similar to a modern QA model like BiDAF [2], QANet has an Input Embedding Layer, Encoding Layer, Context-Query Attention Layer, Model Encoder Layer and an Output Layer. The key differentiator for QANet is its Encoding Layer, which consists of convolution and self-attention exclusively in order to allow convolution to model local interaction and self-attention to model global interactions. For this project, I reimplemented QANet and have a successful implementation. All the model hyper-parameters that I use are the same as the paper - e.g. kernel size of 7 and 128 filters for the Conv layers in, 8 heads for multi-head attention. Regarding performance evaluation for my implementation of QANet, I am using BiDAF [2] as my baseline. BiDAF is another class QA model that achieved state-of-the-art performance before QANet. tt uses a bi-directional RNN layer to encode both a paragraph and a question, pass the encoded paragraph and question through an attention layer and a final modeling layer to output the answer.

Figure 1: QANet model architecture from the QANet paper



5 Experiments

5.1 Data

I use the SQuAD 2.0 [6] dataset and framework to evaluate model performance. SQuAD contains 107.7K query-answer pairs, with 87.5K for training, 10.1K for validation, and another 10.1K for testing. Each paragraph has an average length of around 250 words and the questions are usually around 15 words. Performance metrics for evaluating SQuAD are usually F1 and EM.

5.2 Evaluation method

I ran the same experiments as the original paper[1] to evaluate QANet. The goals are two-fold. First, use the SQuAD metrics of F1/EM to evaluate model accuracy. Second, use train time and inference time as characterized by sample/s to evaluate training speed and inference speed.

5.3 Experimental details

For fairness reason, I trained both models with the same amount of training data (30 epochs worth of data with the same epoch size). In addition, I used the same optimizer (Adadelta) with same constant learning rate (0.5), same EMA rate decay rate (0.999) and same batch size of 16. Then, I ran my model against the SQuAD Dev Set and Test Set. Table 1 captures the F1/EM scores for QANet and BiDAF on the dev set. These results are submitted to the non-PCE SquAD leaderboard (under "Reimplementing QANet"). Table 2 captures the training and inference speed as measured by sample/s.

5.4 Results

As shown in table 1, the model accuracy for QANet is higher than BiDAF by $\tilde{8}$ points the F1/EM score, which is very significant. This aligns with my expectation as QANet uses other state-of-the-art techniques like self-attention which would significantly improve model accuracy.

	F1	EM
QANet	63.316	59.848
BiDAF	54.884	50.955

Table 1: SQuAD 2.0 model performance

However, my QANet implementation is $\tilde{4}$x slower in training time and inference time compared to BiDAF, as shown in table 2. For example, BiDAF can make inference for 601 samples per second while QANet can only make inference for 139 samples per second. And BiDAF can train 450 samples per seconds while QANet can only train 98 samples per second.

This is not expected as the QANet author claims a 4-7 times improvement compared to BiDAF in training/inference speed due to RNN elimination. To understand the 4x slowdown, I instrumented the code to profile the two models. For this experiment, I used the models to perform inference for 5951 samples and measure the aggregate execution time for each layer. Table 4 has a summary of the breakdown of execution time as well as the total execution time.

	Inference time	Training time
QANet	139 samples/s	98 samples/s
BiDAF	601 sample/s	450 samples/s

Table 2: Inference/Training speed performance

	F1	EM
QANet (Test set)	63.316	59.848
QANet (Dev set)	67.388	64.073

Table 3: QANet model performance on SQuAD 2.0

	BiDAF	QANet
Input Embedding layer	0.4s	0.8s
Embedding Encoder Layer	2.3s	4.3s
Attention Layer	0.2s	0.3s
Model Encoder Layer	3.0s	37.1s
Output Layer	3.9s	0.2s
Total Exec Time	9.9s	42.7s

Table 4: Latency Breakdown between two models

6 Analysis

6.1 4x slowdown in training and inference time

The 4x slowdown is the most surprising result. Note that, despite the 4x slowdown, this is not an apple-to-apple comparison between the two models. This is because the QANet model performs significantly better (by $\tilde{8}$ points) so it is a more complex model, which justifies some execution slowdown as a engineering trade-off (Ideally, if I had more time, I would have got a implementation of BiDAF model that matched the same score as QANet and then do a execution time analysis. Nevertheless, this experiment gives an interesting observation and direction which is as described below).

To understand this further, I profiled the execution time for each layer, along with a brief description of the techniques used in both of the models. As shown in table 4, the most time-consuming layer for QANet is the Model Encoder Layer, which takes a total of 37.1 seconds. As explained in table 5, such a layer involves three rounds of encoding with Encoder Block, with each Encoder Block seven-layer deep. Just one round of Encoder Block takes 10s whereas this implementation of BiDAF just take

10s to finish end-to-end. This points out that execution using Encoder Block is not necessarily fast despite use of convolutions instead. Table 5 summarizes the NLP techniques used by either model.

For a more direct execution time comparison between a RNN and a Encoder block, we can look at the timing for the Embedding Encoder Layer. In this layer, both model have a simple layer which makes comparison easier. More specifically, BiDAF uses one a single-layer bidirectional RNN whereas QANet uses a single-layer Encoder Block. RNN takes 2.3s whereas an Encoder Block takes 4.3s. So, RNN runs 2x faster.

While the slowdown can be implementation-specific, one interesting observation from this experiment is, unlike what conventional wisdom suggests, RNN is not necessarily slower than other neural layer to compute convolution-based encoder. Execution time is very implementation-specific.

	Techniques used by BiDAF	Techniques used by QANet
Input Embedding layer	Linear layers	Two rounds of conv layers
Embedding Encoder Layer	One Bi-directional single-layer RNN encoder	Single-layer Encoder Block with self-attention and 2 convs
Attention Layer	Various matrix multiplications	Various matrix multiplications
Model Encoder Layer	One Bi-directional single-layer RNN encoder	Three rounds of 7-layer Encoder Blocks
Output Layer	Bi-directional single-layer RNN encoder + other Linear layers	Various convs layer

Table 5: Techniques used by the two models for each layer

6.2 Test/Dev set score discrepancy

Another observation from the results is, as shown in table 3, QANet has a F1/EM score of 67.4/64.0 with the test set whereas it has a F1/EM score of 63.2/59.8. In other word, my test score is lower than the dev score by 4.2/4.3. My suspicion is that my QANet model may have some level of over-fitting to the test data. And one potential way to shorten the gap is by using data augmentation to train my model.

Upon more investigation, I see that the authors of the original QANet paper also had this issue and they solved this by using data augmentation as well. On the paper, without data augmentation, the F1/FM score of the QANet model has a F1/EM score of 73.6/82.7. With data augmentation, the scores are 76.2/ 84.6. This shows that data augmentation resulted in 2.6/1.9 point gain, which reduces the performance gap between the test and dev score.

7 Conclusion

For this project, I was able to reimplement QANet and have one successful implementation of the model. This implementation outperformed the BiDAF model but it is ~4X slower in training/inference time. Through the heavy coding involved, I learned a lot more deeply on how each of these state-of-the-art machine comprehension and Q&A components such as self-attention and context-query-attention-layer works. One key observation from my experiments is that while conventional wisdom suggests that RNN is slow and other convolution-based encoders are faster, but in practice, it depends on the implementation.

One key limitation of my model is training time/inference time. The QANet author claims a 4-7 times improvement compared to BiDAF but I am seeing a 4x regression. I would like to investigate

this further - first by getting BiDAF and QANet to perform equally on SQuAD (with the same F1/EM score), then by profiling the individual layer involved and finally do a execution time comparision between Encoder block and RNN. It is possible that I didn't use an efficient implementation of EncoderBlock. In addition to that, I would like to use data augmentation to improve my QANet implementation.

References

[1] QANET: Combining Local Convolution with Global self-attention for Reading Comprehension,.Adams Yu , David Dohan , Minh-Thang Luong, ICLR 2018 URL: https://arxiv.org/pdf/1804.09541.pdf

[2] Bidirectional Attention Flow for Machine Comprehension. Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, Hannaneh Hajishirzi, ICLR 2017 URL: https://arxiv.org/pdf/1611.01603.pdf

[3] Layer Normalization. Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton, URL: https://arxiv.org/abs/1607.06450

[4] Attention is all you need. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin. NIPS 2017 URL: https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[5] Xception: Deep Learning with Depthwise Separable Convolutions. François Chollet. URL: https://arxiv.org/pdf/1610.02357.pdf

[6] Know What You Don't Know: Unanswerable Questions for SQuAD. Pranav Rajpurkar, Robin Jia, Percy Liang. URL: https://arxiv.org/pdf/1806.03822.pdf