Question Answering System Implementation Using QANet Architecture

Stanford CS224N, Custom Project

Chafik Taiebennefs
Department of Computer Science
Stanford University
tchafik@stanford.edu

Abstract

QANet [10] is one of the most important papers in the domain of machine reading comprehension and automated question answering (QA). The main goal of the QANet architecture is efficiency and speed which results in significant training and inference performance gains compared to RNN based architectures. In this project, my goal is to 1) implement, from scratch, the baseline QANet model, as described in the QANet original paper, using Pytorch, 2) implement all the required dataset pre-processing, and 3) evaluate my model on the SQuAD 1.1 data set [6].

1 Introduction

Reading comprehension and open domain question answering are critical natural language tasks that many modern NLP models benchmark against. Early reading comprehension models were mostly recurrent (RNN) based, as a result, their training and inference speed was terribly slow. More recently, there has been a shift to make use of attention layers to improve on the performance of these models by using bidirectional attention. While these newer systems performed well, they were still relatively slow and did not leverage the newest state of the art NLP architectures, such as Transformers [7], that exclusively use convolutions or self-attention.

QANet attempts to bring the latest NLP neural networks innovations to the Question Answering NLP problem domain. The paper authors introduce a novel reading comprehension model that achieves both fast and accurate performance. Drawing inspiration from "Attention Is All You Need" [7], The QANet encoder consists exclusively of convolution and self-attention, where convolution captures the local structure of the text, while the self-attention learns the global interaction between each pair of words. The feed-forward nature of this architecture drastically improves model training and inference performance, while maintaining good accuracy.

The main goal of the QANet architecture is efficiency and speed which results in performance gain. The authors QANet demonstrated the efficiency and performance improvements on the SQuAD dataset. My main focus in this project is understanding and implementing the baseline QANet architecture, as well as baselining its performance against the SQuAD 1.1 dataset. In addition, I will investigate hyperparameter configurations to improve the baseline performance.

2 Related Work

Prior to QANet, question answering systems were of two major variations: (1) systems using recurrent architectures such as LSTM for capturing sequential input and (2) systems leveraging attention mechanisms for capturing long term interactions. For example, popular models such as, Multi-Paragraph Reading Comprehension [1], Gated Self-Matching Networks (R-Net) [9] and Match-LSTM [8] use such approaches. However, due to their sequential nature, (1) cannot be parallelized to exploit hardware efficiencies and as such are terribly slow.

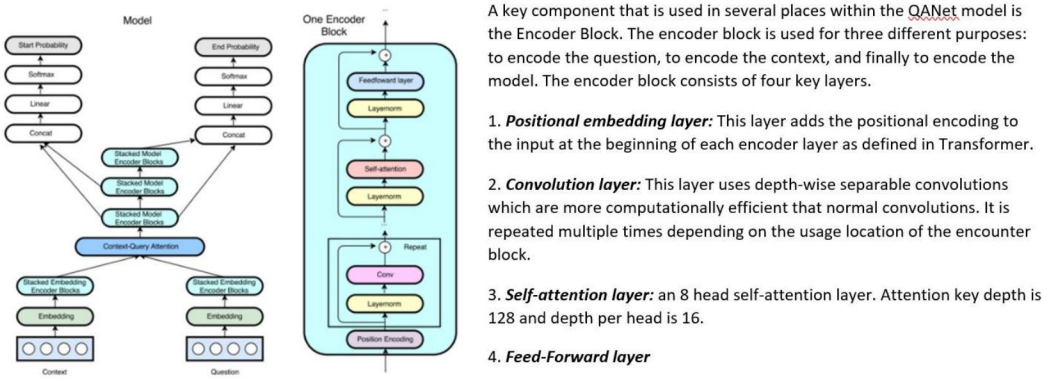
To speed up training and inference, QANet took inspiration from previous reading comprehension work such as BiDAF [4]. Prior to BiDAF, most work in this space used a unidirectional attention mechanism where the query attended to the context. BiDAF introduced bidirectional attention from context to query and query to context. QANet departs from BiDAF in that the encoder blocks have no recurrence but rather just convolutions and multihead self-attention. That is very similar to the Transformer paper with the addition of convolutional layers. Also, Transformers first introduced the idea of using just self-attention and feed forward blocks to model natural language constructs and also introduced the concept of multihead attention which is also used in QANet.

In addition, since QANet has no concept of recurrence, it needs to encode some form of positional information; this concept was also inspired by the Transformer paper. These positional embeddings are more suited than learned positional embeddings because they can extrapolate to longer lengths. These concepts like bidirectional attention and encoders with self attention are core components on my QANet implementation.

3 Approach

Before diving into the details, it is useful to define some important concepts and terminology. A QA problem can be formulated as following: Given a context / passage of n words, C = c1, c2..., cn and a question / query sentence of m words, Q = q1, q2..., qm, we want to output a span S = ci, ci+1, ...ci+j from the original paragraph C if the question is answerable. QANet uses fixed size contexts and questions. In the baseline QANet implementation, only the first 400 tokens in a given paragraph are used as context and 50 tokens in a given question are used as query. For each word, the first 16 characters are only used to compute the character embeddings. If the context/query is shorter than the length threshold, zero-padding is applied. QANet uses pre-trained GloVe [3] vectors as word and character embeddings for representing both query and context.

Below is the high-level architecture of my QANet Model:



The QANet model also consists of 5 key modules:

- Embedding layer: combines the word and character embeddings into a singular representation for each word. This is done by (1) Looking up the 300-dimensional pre-trained GloVe embeddings, (2) Learning a 200-dimensional embedding for each character in the word using convolution, and (3) Concatenating the two embeddings into a single vector. Also, a two-layer highway network is used to avoid the vanishing gradient problem.

- Encoder layer: The context and the question are passed through this encoding layer. It uses a single encoder block with 4 convolutions. The output of this layer is a 128 dimensions vector.

- Context-Query Attention layer: The output from the Encoder layer is fed to the Context-Query attention layer which combines context and query and outputs a representation for each word in context. The purpose of this layer is to calculate two key Matrices: (1) Context-to-query attention matrix, and (2) Query-to-context attention matrix.

- Model encoder layer: This layer consists of 3 stacked encoder blocks that share weights and the number of encoder blocks within each stack is 7. The number of convolutions within each block is 2.
- Output layer: The key layer for prediction is the output layer. The optimization function is formulated as follows: let P1 = softmax(W1[M0; M1]) and P2 = softmax(W2[M0; M2]) be the probabilities of each position in the context being the start and end of the answer span, respectively. W1 and W2 are trainable weights and M0, M1, M2 are outputs from the three model encoders. The objective function is defined as:

$$Log(\theta) = -1/N \sum_{i}^{N} [log(P^1_{y^1_i}) + (P^2_{y^2_i})]$$

Where y1 and y2 are start and end positions of the ith training instance from the ground truth. The model outputs probabilities for all pairs and ultimately these probabilities are turned into a span consisting of a start index and an end index, indicating the location of the answer in the context paragraphs.

4 Experiments

4.1 SQuAD Data Set

The Stanford Question Answering Dataset (SQuAD 1.1) data set consists of questions posed by crowdworkers on a set of Wikipedia articles. SQuAD consists of over 100,000 rows of data in the form of a question, an associated Wikipedia context paragraph containing the answer to the question, and the answer. The ground-truth answer labels are represented in the form of two indices, a start index and an end index, which represent words in the context paragraph. The following is a training example from the SQuAD dataset, consisting of a question, context paragraph, and answer span (in green)

Question: The New York Giants and the New York Jets play at which stadium in NYC ?

Context: The city is represented in the National Football League by the New York Giants and the New York Jets , although both teams play their home games at MetLife Stadium in nearby East Rutherford , New Jersey , which hosted Super Bowl XLVIII in 2014 .

(Training example 29,883)

4.2 Evaluation Method

The model performance is evaluated using Exact Match (EM) and F1 scores.

Exact-Match (EM) Score

$$\text{EM} = \frac{\text{exactly matching answers}}{\text{total evaluated questions}} \times 100$$

F1 score

$$P = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

$$R = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$\text{F1} = 2\frac{PR}{P+R}$$

4.3 Experimental details

Most of my training parameters were based off the QANet paper. The ADAM optimizer was used with $\beta 1 = 0.8$, $\beta 2 = 0.999$, $\epsilon = 10^{-7}$.

All experiments were run on a Standard NC6 Promo Azure Ubuntu VM with 56 GiB of RAM, 6 vCPUs, and 1 NVIDIA Tesla GPU. I run the following experiments in my hyper-parameter search:

4.3.1 Training batch size

I experimented with different mini-batch sizes to find the optimum batch size that balances speed and performance. Namely I tried batch sizes: 50, 100, 200, 300, 500, and 1000

4.3.2 Number of Epochs

I experimented with different number of Epochs to find the optimum number that achieves the best model performance. Namely, I tried the following number of Epochs: 3, 10, 20, and 30

4.3.3 Learning rates

I experimented with different learning rates to find the optimum learning rate that achieves the best model performance. Namely, I tried the following learning rates: 10e-3, 10e-4, 10e-5, 10e-6, and 10e-7.

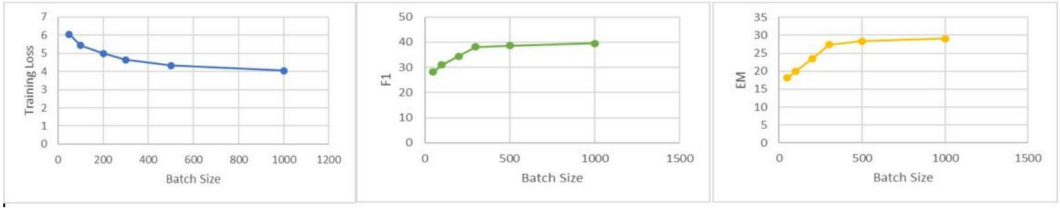
4.4 Results

Overall, my model achieved OK results (i.e. 38 for EM and 52 for F1) on the validation set. Based on the results obtained from the original paper, these results are definitely worse than expected. However, the EM and F1 upward trend during my experiments strongly suggests that training the model for longer epochs would have further improved its performance as I had to stop at 30 epochs due to the VM credit limit and the very long model training time (30+ hours).

Bellow are the main hyper-parameter search findings:

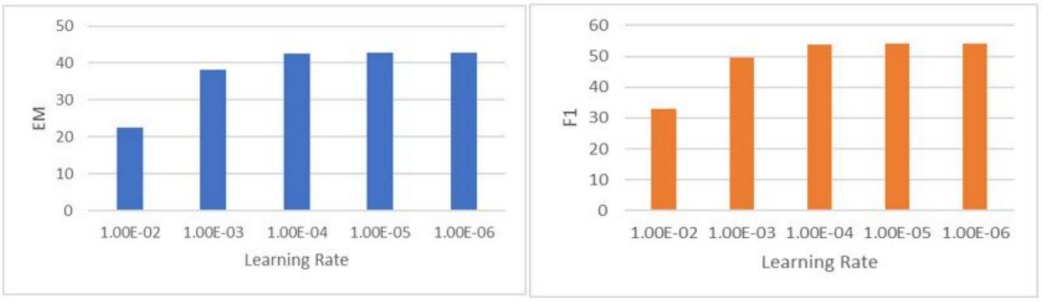
4.4.1 Batch size hyper-parameter search

I run one epoch of the baseline model (Learning rate = 10e-3) at different mini batch sizes. As the batch size increased the training loss dropped and started flattening around 500 batch size. Also the F1 and EM scores increased progressively and started flattening around 500 batch size as well. The Training run time was virtually the same across all experiments as expected. As such, I determined that batch size 500 is the optimum number.



4.4.2 Learning hyper-parameter search

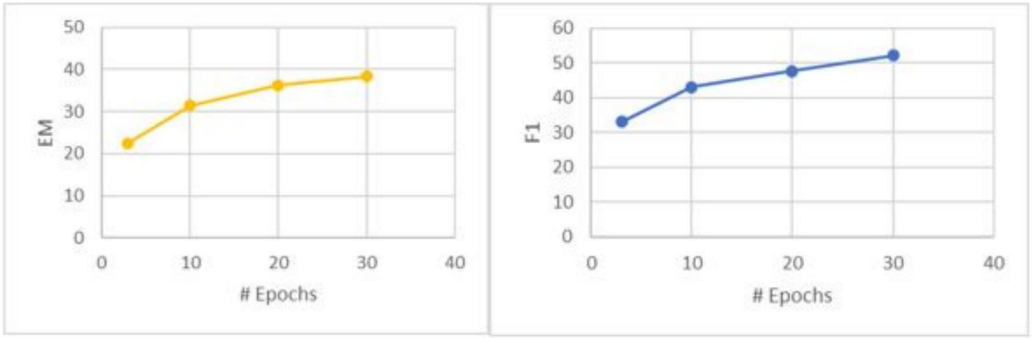
I run 5 epochs of the baseline model at different ADAM optimizer learning rates: 10e-3, 10e-4, 10e-5, 10e-6, and 10e-7. As the learning rate decreased the F1 and EM performance of the model improved to settle around 42 for EM and 53 for F1 score. As such, I determined that the optimum learning rate for model training is 10e-5.



4.4.3 Epochs parameter search

I run the baseline QANet model at different number of Epochs (3, 10, 20, and 30), with a batch size of 500 and learning rate of 10e-05.

The F1 and EM performance of the model kept improving gradually as the number of epochs increased to reach 38 for EM and 52 for F1. Unfortunately, I could not train the model beyond 30 epochs as it started to take too long (the 30 epochs model training took more than 30 hours to complete) and I almost exhausted my VM credits.



5 Analysis

I sampled 100 random questions/answers to perform error analysis. I classified the errors into a number of categories and also classified the errors by skill deficiencies as described in [5] when the model gives a "no answer" or predicts a completely incorrect answer.

Throughout my experiments, I noticed that the errors attributed to partial span mismatch were reduced with longer model training (20 epochs and up). That is surprising as I would expect such improvement to only occur with a larger model architecture (increased number of attention heads, bigger hidden size etc.)

My error distributions show that my QANet model makes errors that are common to most question answering deep learning models in general. It still struggles with inference, bridging, analogy, as well as logical analogy. In general, I observed that in the presence of multiple prolific entries, the model is likely to discard more relevant context information, and focus on the phrases that are most similar in structure. Also, the model tends to fail when there seem to be more than one viable answer to questions.

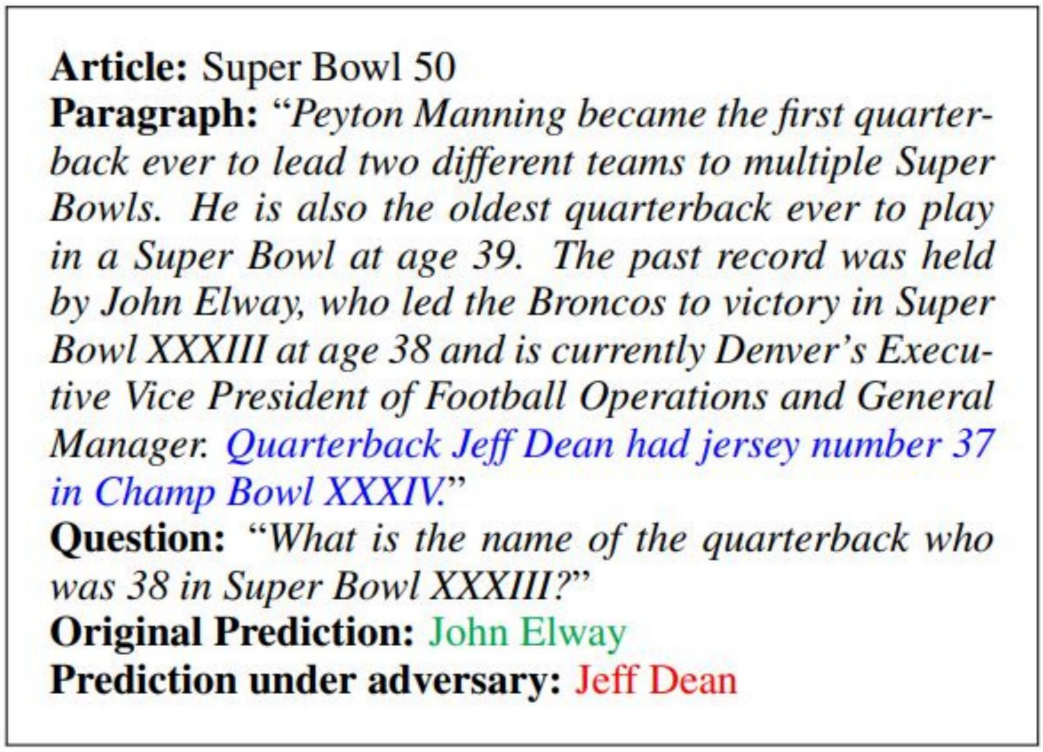
One possible way to alleviate these issues could be to add more self-attention heads to my QANet architecture and decrease the number of convolutions (i.e. encoder blocks). That could potentially help the model better learn global dependencies and could also improve the model's ability to perform logical reasoning. Moreover, using pre-trained contextual embeddings, or adding other features to the input vectors (e.g. named entity types) might improve the model's performance.

6 Conclusion

In this project I implemented the QANet model for machine reading comprehension from scratch. I evaluated the model against the SQuAD 1.1 dataset and analyzed the results. QANet is a complex model with more than 130 layers. Properly implementing the architecture, along with the data pre-processing logic, in a short time and based on the brief descriptions in the paper was a significant challenge. My main motivation behind choosing this project was to carefully study the key ideas (such as attention mechanism, highway networks etc.) from the recent innovations of neural architectures that the QANet is built upon. I believe I accomplished a great deal of hands-on learning throughout this project. However, due to time constraints, I could not further explore the following ideas as I originally hoped:

- Adapting and evaluating the QANet architecture for the SQuAD 2.0 dataset.
- Evaluating different model optimization techniques (Quantization, pruning, etc.) for IoT/Embedded device implementations.
- Finetuning the model to further improve training speed, accuracy, and inference time.

Also, I would have liked to investigate how susceptible models trained on the SQuAD data set are to adversarial attacks. Since the QANet model relies on finding the correct sentence from the context paragraph, it could be vulnerable to adversarial sentences inserted into the paragraph which resembles the question but designed to fool the model [2]. Here is an example:



The sentence highlighted in blue is the adversarial example inserted in order to trick the model. To human readers, it doesn't change the answer to the question "What is the name of the quarterback who was 38 in Super Bowl XXXIII?" as the adversarial sentence is talking about Champ Bowl XXXIV. However, to the model, the adversarial sentence aligns better with the question than the ground truth sentence.

References

[1] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. *arXiv:1710.10723*, 2017.

[2] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *arxiv:1707.07328*, 2017.

[3] Jeffrey Pennington, Richard Socher, , and Christopher Manning. Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014.

[4] Minjoon Seo, Aniruddha Kembhavi, and Ali Farhadi. Bidirectional attention flow for machine comprehension. *arXiv:1611.01603*, 2016.

[5] Saku Sugawara, Yusuke Kido, Hikaru Yokono, and Akiko Aizawa. Evaluation metrics for machine reading comprehension: Prerequisite skills and readability. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017.

[6] Stanford University. The stanford question answering dataset. *https://rajpurkar.github.io/SQuAD-explorer/*, 2018.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, and Lukasz Kaiser. Attention is all you need. *arXiv:1706.03762*, 2017.

[8] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.0790*, 2016.

[9] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, , and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017.

[10] 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. *arXiv:1804.09541*, 2018.