Transformer-XL on QAnet

Stanford CS224N Default Project

James Chao
Department of Computer Science
Stanford University
jbchao@stanford.edu

Abstract

When it was published in 2018, QAnet [1] revolutionized question answering models by replacing traditional LSTM-based models with transformers. Transformers don't suffer from long-term dependancy information loss like RNNs do, but since they process entire sentences at a time, they can only learn from context within their fixed input window. My aim is to fix this issue by reintroducing an element of recursion into the transformer-based model as described in Transformer-XL [2] and update QAnet to be competitive on SQuAD v2.0. To accomplish this, I 1) redesigned and re-implemented the QAnet structure to include word-character embedding convolutional layers to increase contextual information learned and 2) applied transformer-XL to each encoder block by storing previous hidden states in memory, reusing them in calculations of the next hidden state, and using relative positional embeddings. I found that the word-character convolution worked exceptionally well, and when applied to the baseline BiDAF model were able to achieve **EM/F1/AvNA = 59.5/63/69**. The base QAnet was able to perform at similar levels to the baseline, at **EM/F1/AvNA = 53.44/56.15/64.31**, but the transformer-XL improvements did not help and in fact made the model worse, clocking in at **EM/F1/AvNA = 59.5/63/69**.

1 Key Information to include

- Mentor: Kaili Huang
- External Collaborators (if you have any):
- Sharing project:

2 Introduction

Question Answering (QA) has long been a major area of focus in natural language processing. The field had long been dominated by recurrent neural networks (RNNs) before the advent of the transformer [3], a powerful new architecture that has become the industry standard in QA and many other fields. In this project, I seek to combine the best of both worlds and introduce recurrent elements into transformer-based QA models so that they can recapture some of the unique benefits that their predecessors offered.

Question answering involves taking a query and scanning a relevant context paragraph for the span that represents the answer. Previously, this was done with RNNs by sequentially processing each step of the context/query and using information from each previous word to inform the encoding of the next word. This allows RNNs to build contextual relationships, but it suffers from memory loss over long sequences (values in matrices that are multiplied so many times they lose any sort of meaning) and are slow to train. By instead processing entire sequences at once and using self-attention to determine which words are more relevant to other words, this problem is avoided entirely. This is exactly what transformers do, and QAnet [1] leveraged this to acheive record-breaking scores on SQuAD v1.1.

In this project, I re-implement my own version of QAnet from scratch as the backbone of my model. While QAnet is able to acheive remarkable results on SQuAD v1.1, it may struggle with the many "no answer" questions in SQuAD v2.0, as indicated by the AvNA score. I hypothesize that part of this may be due to encoding layers not being able to retrieve contextual information from sentences outside of the fixed input window due to the fixed input window size of transformers. As such, I expand upon QAnet by implementing Transformer-XL [2] on top of it, namely segment-level state reuse and relative positional encodings. In segment-level state reuse, during multi-head self-attention computation, hidden states from each iteration of the model (i.e. each batch or sequence of words/characters computed) are stored into memory and recalled during the next stage to retain contextual information. To incorporate past information, we simply concatenate with the new data. In order to retain information about from the positional embeddings, which would no longer make sense when concatenating the old and new data, we use fixed positional embeddings with learnable transformations so that the values can be trained over time as useful data is concatenated [2].

Additionally, I add in a convolutional embedding layer on combined word and character embeddings, which I call the contextual layer. The purpose of this layer is to allow words and characters to learn from each other before being fed into the self-attention layer of the embedding encoder so that the self-attention can relate certain important characters to relevant respective words and vice versa.

3 Related Work

As mentioned before, the field of QA used to be dominated by RNN-based models, specifically ones that used LSTMs to bidirectionally generate encodings like BiDAF [4]. BiDAF introduced the concept of bidirectional (C2Q and Q2C) attention, where both context and query can learn from each other to build question-aware context representations and context-aware question representations. BiDAF models were able to obtain EM scores of around 65 on SQuAD v1.1

Transformers [3] revolutionized the field by delivering much faster and more accurate results through leveraging the self-attention mechanism. QAnet [1] adapted the transformer to the QA task, and was able to achieve much higher scores (EM = 82.2) than any RNN/LSTM model on SQuAD v1.1.

Currently, cutting edge QA based models that are able to attain the highest scores on SQuAD v2.0 and other datasets are built upon pre-trained transformers, which start with embeddings obtained by training over very large corpora of text. Models using pretrained transformers like BERT [5] and ALBERT [6] are able to acheive remarkable EM scores on SQuAD v2.0 of about 90.

4 Approach

My main approach was to first implement a functional version of the QAnet architecture and then implement Transformer-XL features on top of that. Since the baseline BiDAF model has the same overall structure as the QAnet model, I was able to incrementally develop my reimplementation of QAnet by switching out layers of the baseline model one-by-one. Below are the steps I took in developing my model.

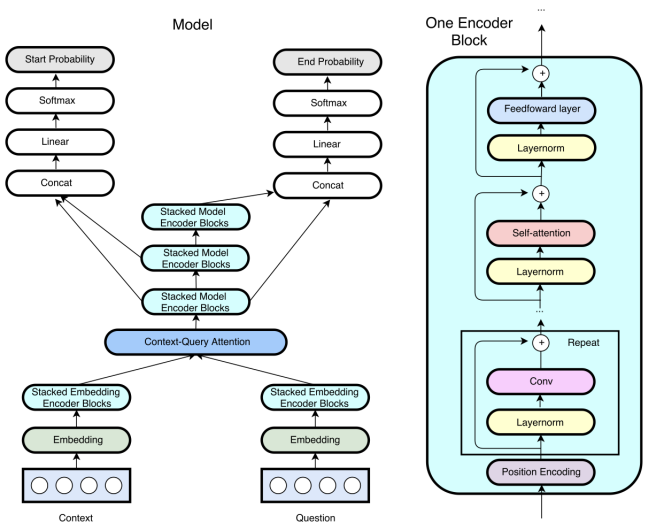

Figure 1. An overview of the architecture of QAnet, which is the basic skeleton this model is built around [1].

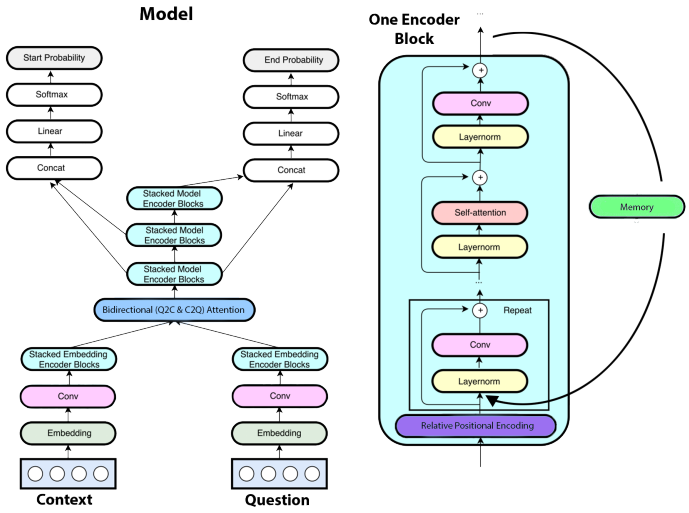

Figure 2. My final model implementation, with the convolutional context layer included.

4.1 Baseline

My baseline for this project is the word-embedding-only BiDAF model provided in the default project. It includes a word embedding layer, an LSTM bidirectional encoder layer, a bidirectional attention layer, two LSTM bidirectional modeling encoder layers, and an output layer. Further implementation details can be obtained from the original BiDAF paper [4].

4.2 Convolutional Word and Character Embedding Layer

I expanded upon the baseline by adding in 200-dimensional character level embeddings, which are initialized randomly and fed through a 2D convolutional layer to learn from each other. I then concatenate these character embeddings to the word embeddings obtained in the baseline and feed both through a convolutional layer to change their dimensionality to the hidden size (h = 128) specified

in the QAnet paper [1]. The joint embeddings are then fed through a 2-layer highway network [7] before being passed to the encoder. This singular change, with the rest of the baseline model intact, was enough to motivate a significant 5-point increase in scores across the board (AvNA, F1, and EM), which will be discussed further in the analysis section.

4.3 Context Layer

I experimented with adding an additional convolutional layer to add additional contextual information between words and characters, but found that this layer caused the model's learning to plateau before going up slowly, the implications of which will be discussed further in the analysis section. While I still believe there is potential in this layer, since it decreased performance, I ended up removing it from my final implementation.

4.4 Encoder Blocks

This is the bread and butter of the QAnet [1] architecture. Each block consists of a positional encoder, several depthwise separable convolutional layers, a self-attention layer, and a feed-forward layer. The base structure of QAnet is presented on the right below.

The depthwise convolutional layer implementation was obtained from [8].

The positional encoder math was obtained from [9].

I make three modifications to the encoder block in my implementation:

4.4.1 Convolutional feed-forward layer

The original QAnet paper [1] mentions it uses linear layers for some of its feed-forward layers, but I use only convolutional layers in my encoder blocks. The goal of this change is to allow different parts of the attention to learn from each other. This convolutional layer has the same architecture as the convolutional layers at the beginning of the block.

4.4.2 State reuse

At the end of each encoder block, the hidden states of each of its layers are stored within long-term memory which persists over all the iterations of training. At the next iteration over the next sequence of inputs, these memories are used in the calculations of the new state and fed together into the convolutional and self-attention layers. This allows the encoder to build relationships over context spans larger than the input window size. Given, the value of the hidden states for the $s-1$th segment of the $n-1$th layer, he calculation of the nth layer's hidden states for the sth segment is given by the following equations:

$$
\begin{gathered}
h_s'^{n-1} = [SG(h_{s-1}^{n-1}); h_s^{n-1}] \\
q_s^n, k_s^n, v_s^n = h_s'^{n-1} W_q^\top, h_s'^{n-1} W_k^\top, h_s'^{n-1} W_v^\top \\
h_s^n = AttentionLayer(q_s^n, k_s^n, v_s^n)
\end{gathered}
\tag{2}
$$

where SG stands for the stop gradient, which lets gradients flow through forward but not backpropagate. This is done so that the gradient for the current function is not backpropagating through to the previous hidden states. I saw no reason to include long-term state memory for any layer other than the self-attention layer, so I will only be saving states for those [2].

4.4.3 Relative Positional Encoding

One problem that state reuse introduces is that concatenation results in positional encodings becoming meaningless. Thus I will only employ relative positional encodings, calculated each iteration, to represent locational information. Relative positional encoding modifies the state reuse mechanic when calculating attention by calculating attention for each query, key, value with respect to a relative encoding matrix R and putting attention values through a positionwise-feed-forward layer to calcualate the next layer's state. These modifications are shown below:

$$h'^{n-1}_{s-1} = [SG(h^{n-1}_{s-1}); h^{n-1}_{s-1}]$$
$$q^n_{s-1}, k^n_{s-1}, v^n_{s-1} = h'^{n-1}_{s-1} W^\top_q, h'^{n-1}_{s-1} W^\top_k, h'^{n-1}_{s-1} W^\top_v$$
$$A^n_{s-1,i,j} = q^n_{s-1,i} k^n_{s-1,j} + q^n_{s-1,i}{}^\top W^n_{k,R} R_{i-j} + u^\top k_{s-1,j} + v^\top W^n_{k,R} R_{i-j}$$
$$a^n_{s-1} = MaskedSoftmax(A^n_{s-1,i,j}) v^n_{s-1}$$
$$o^n_{s-1} = Linear(a^n_{s-1})$$
$$o^n_{s-1} = Layernorm(o^n_{s-1} + h^{n-1}_{s-1})$$
$$h^n_s = PositionwiseFeedForward(o^n_s)$$
[2]

4.5 Bidirectional Attention, Modeling, and Output Layers

While QAnet only uses C2Q and not Q2C for its attention layer, I thought it would be beneficial to still have query representations learn from context representations. Therefore, I use the bidirectional attention layer used in the baseline BiDAF model [4]. .

The modeling layer is the same as in the QAnet paper, and consists of three groups of seven encoder blocks in a row. The parameters for these convolutional layers follow the paper's guidelines [1]. .

The output layer is the same as in the QAnet paper, and combines inputs from between each group of modeling layers to produce trainable representations for the start and end pointers [1]. The probabilities of the starting and ending positions of the answer are modeled as follows, where M0 is the output after the first modeling group, M1 is the output after the second group, and M2 after the third:

$$p^1 = MaskedSoftmax(W_1[M_0; M_1]), p^2 = MaskedSoftmax(W2[M0; M2])$$ [1]

.

5 Experiments

This section contains the following.

5.1 Data

The model will be trained and evaluated on the provided SQuAD 2.0 dev and test sets obtained by splitting the official dev set in half. The training dataset is the official SQuAD training set and consists of 129,941 examples, whereas the dev set has 6078 examples. The datasets consist of question, context, and answer triplets for which the model has to perform the following task: given a query, determine the start and end pointers marking the boundaries of the correct answer in the context paragraph. There also exist many questions without an answer, for which the model should return no answer.

5.2 Evaluation method

The model is evaluated on three metrics: exact match (EM), F1, and answer vs. no answer (AvNA). They are defined as follows:

EM: Whether the model got the exact start and end of the answer, or successfully predicted that there was no answer.
F1: A harmonic mean of precision and recall. Precision refers to how much of the predicted answer is in the actual answer, and recall is the opposite.
AvNA: A special metric designed by the course staff to evaluate how well the model differentiates between question/context pairs that have answers and ones that do not.

5.3 Experimental details

I mostly used the hyperparameters specified in the original QAnet paper [1]. They are as follows:

L2 weight decay: $\lambda = 3x10^{-7}$
Dropout probability: 0.1

Hidden size: 128
Character embedding size: 200
Encoder layer convolution number: 4
Encoder layer convolution number: 2
Adam beta 1: 0.8
Adam beta 2: 0.999
EMA Decay: 0.999

Similar to the paper, I used a learning rate that warmed up with an inverse exponential increase from 0.0 to 0.001 over the first 1,000 steps, and then remained constant from there. The kernel sizes for my convolutional layers were set to 1, since I had trouble with higher kernel levels and the sizes of the embeddings. I trained the baseline and BiDAF + Convolutional Embedding iterations for 10 epochs each, since I noticed the slopes were mostly flat after that, which took about two hours each. Surprisingly, despite being known for their speed, the transformer based models (base QAnet and Transformer-XL) took much, much longer to train. I trained both of these for about 30 epochs, which took between 6 to 8 hours each.

5.4 Results

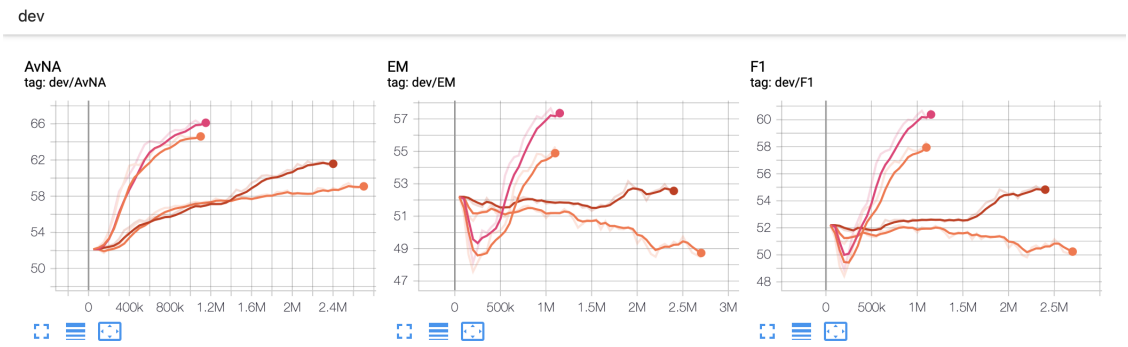

Figure 3. My final training graphs. The upper orange line is the baseline, the magenta line is the convolutional embeddings, the red line is the base QAnet, and the lower orange line is the Transformer-XL.

Model	Baseline	Conv. Embedding	QAnet	Transformer-XL
AvNA	64.88	**69**	64.31	59.2
EM	55.28	**59.5**	53.44	48.46
F1	58.35	**63**	56.15	50
NLL	3.04	**2.78**	3.34	4.07
Num Iterations	1.1M	2.5M	4M	2.7M

Figure 3. My final training scores.

QAnet Test EM/F1 = 51.023/54.052

My models did not perform as well as I expected them to. I exptected QAnet to perform better than BiDAF across the board, and for Transformer-XL to at least perform as well as QAnet. The two transformer models' training curves are not in the desired shape, and take forever to start learning or, in the case of Transformer-XL, progressively get worse. This suggests some flaw with my implementation of QAnet since the baseline scores for QAnet are much higher than what I was able to achieve and I'm utilizing the same hyperparameters as specified in the paper.

The convolutional embedding layer, when paired with the BiDAF model, worked surprisingly well, raising scores across the board by 4-5 points. With this early success, I continued my approach of adapting each layer one by one, adding in additional improvements like more convolutions or state reuse as I went along. My finished model ended up gluing together different parts of both models and linking together potentially incompatible layers like the BiDAF attention layer or highway encoder

with the transformer encoder layer. Further discussion of results and suspected problem areas is included in the subsequent section.

6 Analysis

6.0.1 Training Graphs

Compared to the ideal-looking training curves of the baseline and convolutional embedding models, the transformer-based models' curves clearly indicate something inhibiting training, or perhaps training of the wrong things. What's interesting is that throughout almost all their iterations, both transformer models do steadily improve in AvNA, even when the other two scores were not improving at all. This could indicate that AvNA is easier to train for, or that my own modifications caused the model to fixate on this distinction. Regardless, neither base QAnet nor Transformer-XL managed to learn much in terms of EM and F1. Since my learning rate and other related hyperparameters are all the same as in the QAnet paper, this must be because information is not being passed forward through the layers in a meaningful way, or that information is not being allowed to backpropagate through the layers to the areas that need to be impacted the most. Below are some possible reasons why this may be the case.

6.0.2 Potential Issues

1. **Incorrect formation of Q, K, V tensors during self-attention calculation.**
   In the self-attention layer, to separate out Q, K, and V, I used a linear projection layer to map the input tensor of shape (batch size, sequence length, hidden size) to a tensor of shape (batch size, sequence length, 3 * hidden size) so that I could have three different tensors for my Q, K, and V, each of size hidden size. This implementation makes logical sense, but I don't know if it's exactly what torch's $nn.MultiheadAttention$ is looking for since its implementation is behind the scenes. Implementing my own version of multihead self-attention was a goal of mine, but I had many other issues that I felt should be prioritized due to time constraints.

2. **Too many convolutional layers**
   When I first successfully implemented my convolutional embedding layer, I was shocked to find out the extent to which an extra convolutional layer improved my model by so much. I reasoned that letting different parts of words and sentences learn from each other as much as possible would lead to the greatest success. As such, I replaced the linear feed-forward layers in the encoder block with convolutional ones and added an additional contextual encoding convolutional layer. I found the latter actually prevented training, while the former didn't have much of an impact. Perhaps convolving over some of the many abstract representations of data in the model don't help it build representations at all but instead jumble up information that should remain distinct. This includes the convolutional embedding layer that I built the rest of the model around: just because it worked with the BiDAF model doesn't mean it's well-suited for the QAnet architecture.

3. **Transformer-XL**
   Perhaps the Transformer-XL architecture is not well-suited for QAnet or SQuAD. If the sequence lengths aren't too long for a singular input window to process, then there's no need for state reuse and I could just be adding on extraneous information that confuses the model. Indeed, each context paragraph of BiDAF is only about 300 words at most, and I found that I could simply increase the maximum sequence length in my positional encoder to be larger than that. Whether or not this had unintended consequences or adequately solved the problem requires further experimentation. Additionally, I very well could have misunderstood the convoluted math in the paper and implemented the memory reuse and relative positional embeddings incorrectly.

4. **x.transpose(1,2)**
   At many points in the model, one layer, such as a layer norm, required its input to be in the shape (batch size, sequence length, hidden size), while the next, say a 1D convolutional

layer, required its input to be in the shape (batch size, hidden size, sequence length). This happened at many points in many forms throughout the model, and I almost seemed to be transposing the two dimensions back and forth at random. Perhaps there is a way to avoid so many transformations so information doesn't get confused as it's fed through each layer of the model.

5. **Meaningless output**
   At the end of a run, the model outputs two probabilities: one for the starting index of the answer, and one for the ending index. This is the same output as the BiDAF model, so after extracting outputs from each of the three groups of modeling encoder blocks, I fed the logits through the masked softmax function used by the BiDAF model and hoped for the best. However, the information could have arrived in the output layer in a very different state than for the BiDAF model, and so using the same output function on incompatible data might have produced misleading results.

7 Conclusion

In my work building this model, which in essence is a hybrid of three different models (BiDAF, QAnet, Transformer-XL), I found that combining different aspects of different models, even if they're all beneficial on their own, does not always produce better results. Transformer-XL in particular does not seem well-suited to this task, as there doesn't seem to be much use for state reuse without extremely long context paragraphs, and relative positional encodings would thus only subtract valuable information from the absolute ones in the base model. Nevertheless, I did successfully expand upon the BiDAF model with my own convolutional embedding layer with respectable results. I implemented a functional version of QAnet, which stagnated at first but did end up learning. Given more epochs and Azure credits, it is very likely its scores would have improved further. Finally, while my Transformer-XL implementation was unable to perform to standards, I was able to learn about what unique benefits it offers and when it may or may not be useful. Future work will go into remedying the limitations of my model; namely, why its learning stagnates for so long and why it is unable to learn at a sufficient rate.

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. *ArXiv*, abs/1804.09541, 2018.

[2] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *ArXiv*, abs/1901.02860, 2019.

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

[4] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *ArXiv*, abs/1611.01603, 2017.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

[6] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *ArXiv*, abs/1909.11942, 2020.

[7] Oyebade Kayode Oyedotun, Abd El Rahman Shabayek, Djamila Aouada, and Björn E. Ottersten. Highway network block with gates constraints for training very deep networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1739–173909, 2018.

[8] BangLiu. Bangliu/qanet-pytorch at 0ce11ca1494c6c30d61c0bf2b78907fe27369962.

[9] Samuel Lynn-Evans. How to code the transformer in pytorch, Oct 2018.