QANet Analysis: CS 224N Milestone Default Project (non-PCE)

Timothy Le
Department of Computer Science
Stanford University
Stanford, CA
tle7@stanford.edu

Abstract

The SquAD dataset is an exciting problem for NLP deep learning models. Given a context passage and a question, a model must predict the start and end indices in the context as an answer, or the model can predict no answer. Based on the QANet architecture, I have implemented the components described in this paper. The most significant portion of the described architecture in this paper comes from a new self-attention encoder block that has has multiple convolution networks that output to a self-attention mechanism. In the process of developing this model, I have observed how particular components, such as multi-head attention, are vital to the structure and performance of the model. I describe the positive and negative effects of both simplifying and directly utilizing components of the QANet architecture.

1 Introduction

The SQuAD dataset presents the challenge of predicting start and end indices of an answer based on a passage query. These start and end indices must be sub-spans of the passage/context. SQuAD 2.0 also allows for some correct answers to have "No Answer," adding an additional nuance to the problem. The existing baseline for this model uses sequential LSTM's throughout the model to predict the start and end indices for answers. The QANet architecture offers an approach that uses self-attention and parallelizable depth-wise separable convolutions, allowing the model to learn local interactions through the convolution layers and global interactions through the self-attention mechanism.

This report describes the effects of using the components in the QANet architecture and simplified versions of its core components. This analysis highlights why complexity in certain portions of the model, such as using multi-head over single-head attention, is vital when working with this model's architecture. The QANet model also utilizes layer normalization and dropout, which appear to also impact results. Overall, the analysis in this paper demonstrates how the QANet paper core design choices are vital in its successful results.

2 Related Work

The structure of my model comes from the QANet paper.[1] At the time of its releast, this model had a higher F1 score than the best published F1 score. QANet uses depthwise separable convolutions and self-attention. The multi-head self-attention mechanism comes from the "Attention is All You Need"/Transformer paper.[2] This mechanism measures the similarity between a query and a key for

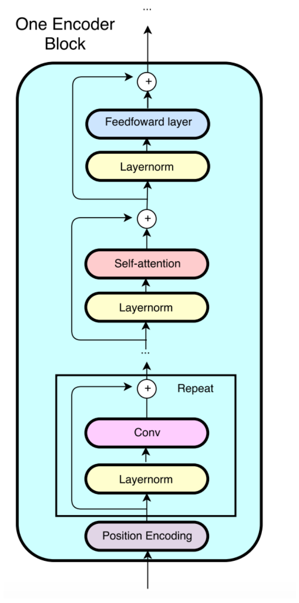
[1]Yu, Adams Wei, et al. "Qanet: Combining local convolution with global self-attention for reading comprehension." arXiv preprint arXiv:1804.09541 (2018).

[2]Vaswani, Ashish, et al. "Attention is all you need." Advances in Neural Information Processing Systems. 2017.

each of the positions in the input. Because the goal of this project is predict the correct start and end indices for an answer to a passage query, this mechanism is useful in understanding the relationships between different positions in a passage or query. This mechanism optimizes for speed by utilizing dot product attention. QANet additionally optimizes for speed by not using any sequential recurrent neural networks, but it uses parallelizable depth-wise separable convolutions.

3 Approach

The baseline model can be described in the default project handout.[3] My milestone model builds upon this model by changing the encoder and model blocks, replacing them with the transformer encoder block specified in the QANet paper. At a high level, the encoder layer uses a single transformer encoder block while the model layer utilizes stacked transformer encoder blocks. Within each block, there is a self-attention mechanism. According to the "Attention is All You Need"/Transformer Paper, self-attention is a mechanism for relating different the positions of a sequence together, forming a representation of the given sequence. We will now describe the following parts of the encoder block and how we used them, and we conclude with a description of the output layer, which is specific to the QANet architecture. Below, I have included an image of the encoder block to visualize this important component of the architecture:



3.1 Position Encoding Layer

Based on the Transformer paper, the self-attention mechanism has no positional information on its own, so there must be a mechanism to propogate positional information. The exact calculations are in the paper. For the position encoding layerm I used the implementation from the jadore801120 transformer repository.[4] At a high level, this code forms a table of position encoding information and adds it to the embedding matrix.

3.2 Convolution Layers w/ Linear transformation

In order to use the self-attention mechanism from the Transformer paper, we must form our query, key, and value matrices. We do this, based on the QANet paper, by running the embeddings from the previous layer through four depthwise separable convolutions. I modified a public depthwise separable 2D convolution layer to become a 1D convolution layer.[5] This layer was important in

[3]http://web.stanford.edu/class/cs224n/project/default-final-project-handout.pdf
[4]https://github.com/jadore801120/attention-is-all-you-need-pytorch/blob/master/transformer/
[5]https://github.com/tstandley/Xception-PyTorch/blob/master/xception.py

maintaining the dimensions of the original embedding matrix passed in, allowing this layer to function well with the existing BiDAF attention layer. I also timed this depthwise separable convolution, and I found it to be slightly faster than a normal convolution by an average of 0.015 seconds for each call for the forward function.

I pass in the embeddings from the previous layer with positional encoding information to this convolution layer. I then pipeline the convolution layers together to learn from the embeddings and positional information. I then add position encoding information to this output to preserve position encoding information. I then apply a linear layer to the output of the convolutions 3 times to form my query, key, and value for the self-attention multi-head mechanism.

3.3 Self-attention mechanism: multi-head and single-head

Based on the QANet paper, I am applying a self-attention mechanism to the output of the above convolution layer. Please refer to the Transformer paper for details for exact calculations. I implemented both single-head and multi-head attention to compare performance. For multi-head attention, I used 4 heads, based on default project recommendations. Multi-head attention is supposed to be more effective on complex data. Rather than a single head operating on the whole hidden size dimension, each head operates on a smaller portion of the hidden size dimension. After the self-attention mechanism, I have a residual connection that adds the input of the self-attention mechanism to its output, preserving information that could have been lost in the attention mechanism operations.

I wrote the wrote both single-head and multi-head functions myself based on the Transformer paper while referring to the multi-head attention implementation in the HuggingFace Transformer repository.[6] The output of this mechanism is passed into the feed forward layer of the encoder block.

3.4 Feed Forward layer

The last component of the self-attention encoder block is a feed forward layer. According to the Transformer paper, the purpose of this layer is to be a fully connected layer that applies two linear transformations to each position. The output of this information is added to the original input, forming a residual connection. I based my implementation on the jadore801120 referenced above and on the Transformer paper.

3.5 Model Layer - Stacked encoder blocks

The model layer utilizes the transformer encoder blocks. The modelling layer uses three stacked blocks that share weights, as described in the QANet paper. For my experiments, each stacked block has 4 to 6 encoder blocks. I was not able to make the stacked block have 7 encoder blocks as is done in the QANet paper. As I increased the number of encoder blocks in a stack, the machine began to run out of memory, leading me to lower the batch size. I implemented this layer based on the descriptions in the QANet paper and without referring to an existing implementation. Each of these stacked blocks forms an output that gets passed to the output layer.

Note: the "Experiments and Analysis" section describes how the initial model layer was a single encoder block early on in development of the model, and the final model features the approach described in this section.

3.6 QANet Output Layer

The QANet output layer is specific to the QANet architecture. Unlike the BiDAF architecture, it does not use an RNNEncoder. It also takes in the 3 matrices outputted from the model layer: $M0$, $M1$, and $M2$, each corresponding to one of the three encoder blocks.

The layer has two linear layers. The first linear layer is applied on the concatenation of $M0$ and $M1$, and the second linear layer is applied on the concatenation of $M0$ and $M2$. A masked softmax is applied on the output of each linear layer, forming the output of this layer so that the loss can be calculated.

[6] github.com/huggingface/pytorch-openai-transformer-lm/blob/master/

4 Experiments and Analysis for non-PCE division

The data used is from the SQuAD dataset described in the default project handout cited above. The evaluation metrics are the F1 and EM scores. I built this model by making the model as simple as possible, described below, and then incrementally making the model more complex. This approach allowed me to observe how certain components of the model have a vital role in the QANet architecture. Below, I will describe the results of making the model more complex. Because the most significant portion of the QANet architecture is the encoder block used in the encoder and modelling layers, most of the results will be related to making changes in the encoder block. Note that I have combined the "Experiments" and "Analysis" sections so that the reader does not have to repeatedly refer back to the results when reading a separate "Analysis" section.

The following table is a summary of the results described below. These results represent the scores from the last epoch of training the models, which approximate how well the model is doing overall. Ideally, these results would be the best scores from the entire run, but I collected the data from the last epoch because it was more accessible than the best scores.

Name/ Metric	Simple Model	Resid Conn.	QANet Output Layer	Stacked Model Layer (4)	Multi-Head Attn	Full QA	Char Embed	Baseline
Train F1 Score	52.19	50.92	46.68	38.21	48.21	52.19	61.57	61.53
Train EM Score	52.19	49.22	47.27	37.71	48.13	52.19	58.12	58.24
Train Dev NLL	8.05	4.21	5.18	7.88	7.74	6.53	3.10	3.05

4.1 Simple Model

My initial model featured an encoder block that had 2 convolution layers and only single-head attention. My model layer was also a single encoder block, rather than having stacked encoder blocks. The Simple Model also only used the output layer of the BiDAF baseline model. Each subsection below describes advancing the model from this simple model.

This model achieved a training Dev NLL of 8.05, an F1 score of 52.19, and an EM score of 52.19. Qualitatively, this model appeared not to learn because the loss started at and stayed at around a Dev NLL of 8 for 30 epochs. With such a high loss that did not decline, I chose to preserve cloud CPU time and not run the test script.

4.2 Residual connections in the Encoder Block

The QANet architecture has a residual connection between every layer in the self-attention encoder block. The purpose of each residual connection is to ensure that as transformations, such as convolutions, are applied to the data, crucial information such as position encodings propogate forward and do not get lost. One design error of the Simple Model was that it only had a residual connection in the feed forward layer and not in the convolution layer or the in self-attention mechanism.

Once I added these residual connections to the convolution and self-attention layers, the training Dev NLL started at a similar loss as in the Simple Model, and it began to decrease until it reached a training Dev NLL of 4.21, with F1 and EM scores of 50.92 and 49.22, respectively. On the testing script, the model now achieves a Dev NLL of 5.40, and F1 and EM scores of 52.14. While the F1 and EM scores on the testing data were both low with the added residual connections, this result demonstrates that without the residual connections for each layer in the encoder block, the loss cannot go down.

One crucial piece of information that likely gets lost in the Simple Model is the positional encoding information that is vital in predicting start and end indices. With these modifications, this positional encoding information now gets added to the output of the convolution layer to form the input to the self-attention mechanism. Because the single-head self-attention mechanism also has a residual connection, the positional encoding information that was in the input of the self-attention mechanism also gets propogated to the feed forward layer. As mentioned in Ashish Vaswani's guest lecture, residuals are vital in a model, and this analysis supports this concept.[7]

4.3 Adding QANet Output Layer

With a lower training Dev NLL, I now changed the BiDAF output layer to the QANet output layer. While the previous BiDAF output layer only needed a single matrix, the QANet output layer needed three input matrices. Therefore, I had three single encoder blocks that shared weights. On the last epoch of the training, the Dev NLL dropped to 5.18, but the F1 score dropped to 47.27, and the EM score also dropped to 46.68. However, on the test script, the Dev NLL was 5.66, and both the F1 and EM scores were 52.19, same as before.

At first glance, based on the drops from the training data, it seemed as if there may be an error in the output layer. While this is possible, an important note is that each of the three output matrices of the model layer was formed by a single encoder block, rather than a stacked encoder block. I hypothesized that because the output layer was designed specifically for the QANet architecture, the output layer may have been expecting a more complex output from the model layer in order to perform its computation. The next phase was then to make the model layer more complex.

4.4 Stacked Model Encoder Blocks in Model Layer

With the hypothesis that a more complex model layer was needed in order for the output layer to perform proper computations, I changed the model layer from three single encoder blocks to having three stacked encoder blocks containing 4 encoder blocks. On the training data, the Dev NLL rose to 7.88, with respective F1 and EM scores of 38.21 and 37.71. On the testing data, the Dev NLL was 7.74, with respective F1 and EM scores of 39.99 and 39.54.

The model results have declined, which could indicate that the problem is having the stacked encoder blocks. At this point, I also modified for the context and query encoders to share weights, as indicated in the paper, and I used the Adam optimizer according to the QANet paper specifications. The stacked encoder blocks, shared weights in the encoders, and Adam optimizer are all potential sources of the drop in the scores. However, I recognized that the model encoder is performing a complex set of operations on its input tensors.

Therefore, I hypothesized that the problem may be that model layer with stacked encoder blocks may be performing complex operations on inputs that were too simple since it pipelines multiple encoder blocks based on the input. The encoder layers form outputs that, after the attention layer, become inputs to the model layer. This hypothesis of the model layer problematically performing complex operations on simple inputs seemed plausible because the encoder blocks in the encoding and model layers were only using single-head attention. I therefore tested this hypothesis by incorporating multi-head attention as my next phase.

4.5 Multi-Head Attention in Encoder Blocks for Encoder and Model Layers

Because the output and model layers are more complex, my hypothesis was that having a more complex encoder block would potentially improve the results. I therefore implemented multi-head attention in the encoder blocks, replacing the previous single-head implementation. This directly affected the encoder and model layers, which both used the encoder blocks. On the training data, the Dev NLL remains high at 7.74, but the F1 and EM scores have both increased to 48.21 and 48.13, respectively. On the test data, the Dev NLL is 7.75, and the F1 and EM scores are 49.07 and 49.00 respectively.

This addition of multi-head attention shows a training improvement of roughly 10 F1 and EM points (38.21 to 48.21 for F1 and 37.71 to 48.13 for EM). One sees the significance of multi-head attention

[7]http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture14-transformers.pdf

from this improvement in F1 and EM scores although the NLL is still high in both training and testing. It appears that it is difficult to expect the model to improve by only making one component of it complex; other parts must also increase in complexity.

4.6 Full QANet: Dropout, Layer Norm, and 6 Stacked Encoder Blocks

With the loss still being high at 7.74 with the addition of multi-head attention, I added layer normalization, dropout with a rate of 0.1 throughout the model, and increased the model layer to have 6 stacked encoder blocks. This implementation represents the closest implementation to the paper I was able to form.

For training, the Dev NLL decreases to 6.53, and the F1 and EM scores is both 52.19. On testing data, the Dev NLL is 6.54, and the F1 and EM scores are also 52.19, always predicting "No Answer". The loss has decreased from the multi-head component. I noticed that applying dropout and layer normalization alone, without a deeper model layer, made the training Dev NLL to stop decreasing at around 6.55, at which point the training F1 and EM scores were both 52.19. Deepening the model layer's stacked blocks therefore did not improve the model. A future step would be to increase the dropout rate to see if the loss would go down significantly. However, such a high NLL, compared with that of the baseline, and low F1 and EM scores must indicate a bug that would require more time to solve.

4.7 Character Embeddings with Baseline

Because the fullQA results were not above baseline, I decided to implement character embeddings on top of the baseline BiDAF model. This helped to improve baseline results because the character embeddings allowed the model to learn the morphology of words, especially out of vocabulary words. On the training script, on the last epoch, the dev NLL is 3.10, with an F1 score of 61.57 and an EM score of 58.12; these values are comparable to those of the baseline's last epoch (F1 of 61.53 and EM of 58.24). On the testing script, this model's dev NLL is 2.94, with an F1 score of 62.29 and an EM score of 59.10. These scores on the testing script are above the baseline's scores (F1 of 61.27 and EM of 58.46). Because this implementation yielded the highest scores, I submitted this model under the name "QANet Analysis7" to the non-PCE test leaderboard.

5 Conclusion

The QANet paper makes seeks to "justify the usefulness of each component." While my final QANet model does not perform as well as intended, I chose to title this report "QANet Analysis" because the incremental development approach in this report allowed me to analyze the efficacy of core components of the QANet architecture. The analysis in this paper has demonstrated the importance of having residual connections throughout the encoder block. Without proper residual connections, the model's loss did not decrease. The above results also support the notion that one cannot just make part of the model complex without also adding complexity to related portions of the model. Simply making one portion complex may lead the model's performance to decline. In the context of this paper, an output layer that expected complex input required a complex (stacked) model layer, and this complex model layer required a more complex multi-head self-attention mechanism. Compared with the Simple Model, the full QANet model, which represents the closest model I could build to that of the QANet paper, achieves the same F1 and EM score, but has a lower training NLL (from 8.05 to 6.53) that is likely do to layer normalization and/or dropout.

Going forward, I would investigate ways to first make the training loss decrease to at least that of the baseline model (training Dev NLL ≈ 3). Debugging would certainly be the priority to investigate if there is a logic error. Another way of debugging could be analyzing if the model is complex enough; for example, I could increase the number of attention heads for the multi-head attention mechanism and observe the effects. I would continue to incrementally make the model more complex while debugging layers. Based on my above analysis, this incremental approach has allowed me to understand the usefulness of core QANet architecture components.

References

[1] Adams Wei Yu et al. "Qanet: Combining local convolution with global self-attention for reading comprehension." In: *arXiv preprint arXiv:1804.09541* (2018).

[2] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing System* (2017).

[3] *attention-is-all-you-need-pytorch*. URL: `https://github.com/jadore801120/attention-is-all-you-need-pytorch/blob/master/transformer/SubLayers.py`.

[4] *pytorch-openai-transformer-lm*. URL: `github.com/huggingface/pytorch-openai-transformer-lm/blob/master/`.

[5] Ashish Vaswani and Anna Huang. *Self-Attention for Generative Models*. Feb. 2019.

[6] *Xception-PyTorch*. URL: `https://github.com/tstandley/Xception-PyTorch/blob/master/xception.py`.

[2] [1] [4] [3] [6] [5]