QANet: Convolutions and Attention are All you Need

Anirudh Joshi
ajoshi80@stanford.edu

Stanford University

Abstract

The last year has seen attention based Natural Language Processing models start out performing their recurrent based equivalents[1,2]. With recurrence being one of the major speed bottle necks in NLP models for both training and inference, it is important to be able to build models that are easily parallelizable to take advantage of modern computing like GPUs. Transformers introduced a network architecture that was fully attention based however it has also been shown that convolutions can provide better local context understanding and hierchical language comprehension [2]. QANet proposes to use both convolutions and self attentions to form a model architecture that is faster than recurrent based approaches. From my experiments, I found that augmentations to recurrent based models like BiDAF can produce higher performance than the custom implementation QANet on SQuAD 2.0. It's important to note that while QANet was one of the top SQuAD 1.1 performers there is no SQuAD 2.0 baseline to compare a custom implementation to.

1 Introduction

Reading comprehension and question answering are critical natural language tasks that many modern NLP models benchmark against [1]. The applications of a good reading comprehension system can extend from digital assistants and chat bots to even medical applications where patients want to know what conditions their symptoms could map to.

Early reading comprehension models were recurrent based because recurrent models were able to naturally learn sequential information in a sentence [2]. While the sequential nature provided representational benefits, the network's training and inference speed would significantly increase depending on the length of the input sentence. Recently there has been shift to make use of attention layers to improve on the performance of these models. Previous reading comprehension models have made use of bidirectional attention flows between the context and query as well as encoded representations of the context and query [6]. While these systems perform well, they are slow and do not take advantage of recent work in Transformers and other models that exclusively use convolutions or self attention. Given that most of the current state of the art natural language models are solely attention based, it seems logical to extend this to reading comprehension.

QANet attempts to bring these innovations in neural architecture for a question answering model. QANet proposes to exclusively make use of convolutions and self attentions in the encoders to produce context and query representations. The motivation here is that convolutions capture local structure while the attention captures global structure in the sentence. On top of these encoders, there is a context to query attention layer which implements the bidirectional attention that allows both the context and the query to attend to each other. The reason why such an attention layer works is that it produces representations of the context that are query aware. By computing the attention at every time step and allowing attention weights to propogate to modeling layers, the architecture reduces the risk of information loss caused by summarization. Finally output probabilities are computed to determine the start and end indices of an answer in the context.

While the hypothesis was that such a combination of attention and convolution would improve F1 and EM performance on SQuAD 2.0, I found that using character embeddings with BiDAF produced a better F1 score (63) than using my implementation of QANet (52). This possibly could be an implementation issue and the results section and analysis will walk through the step by step process as well as ablation tests.

The key advantage of using convolutions and attention is to improve speed in inference and training. With the increased focus on privacy and running machine learning on the edge, faster models would be preferable. However from my experiments I saw that while BiDAF took 30 minutes to train per epoch, QANet takes close to 55 minutes per epoch. This could be due to the increase in number of parameters in the model.

2 Related Work

QANet took inspiration from previous reading comprehension work like BiDAF [6]. 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. Further to reduce information loss caused by intermediate representations, BiDAF's attention is not used to summarize the context into a fixed length vector but is rather computed at every time step and the attention vector along with previous layer representations are allowed to propogate to further modeling layers.

As described above QANet departs from BiDAF in that the encoder blocks have no recurrence but rather just convolutions and multihead self attention. This is very similar to the Transformer paper with the addition of convolutional layers [5]. Transformers first introduced the idea of using just self attention and feed forward blocks to model natural language phenomenon. The authors introduce the concept of multihead attention which is also used in QANet. Instead of performing a single attention function the authors found that by computing the attention 'h' times the model could jointly attend to information from different representational subspaces at different positions.

Since QANet has no concept of recurrence it needs to encode some form of positional information. This concept is also inspired by the Transformer paper where they use sin and cosine functions of different frequencies. The positional embeddings have the same hidden model dimension so they can be summed. 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 implementation of QANet.

3 Approach

The model is heavily based off the QANet architecture (Figure 11 in Appendix) so my approach was similar to what the authors took in QANet. I implemented the code from scratch for most of the QANet layers and the model itself. The only externally sourced code was an open source implementation for Multihead attention. The goal was to produce an implementation of QANet that matches the non data augmentation results in the original paper.

1. Embedding Layer: Pretrained 300 dimensional GloVe embeddings were used to initialize the word embeddings. All out of vocabulary tokens are mapped to <UNK>. The character embeddings are obtained by initially using GloVe embeddings for the characters but then passing through a convolutional layer. The embeddings for the words and output of the convolutional layer are concatenated then passed through a two layer Highway network to obtain a 128 dimensional embedding. This part is unique to my model as the original QANet outputs an embedding size of word embeddings + character embeddings. I also run experiments where I follow QANet's version of the embedding shape. When this is done, I run the concatenated word and character embeddings through a Depthwise Separable Convolutional layer to map the dimensions of the embedding output to the hidden size of 128. When taking this approach I tried methods: 1. Concatenating the 300 dimensional word embeddings with the 64 dimensional character embeddings and then resized to the hidden dimension of the model (128). 2. Projecting the 64 dimensional character embeddings to a 200 dimensional embedding, then concatenating with the word embeddings and resizing.

2. Encoder Block: The input to the encoder block is summed with a positional encoding consisting of sin and cos functions as described in Vaswani et al [5]. This is passed into a block of 4 Depthwise Separable Convolutional Layers. As described in QANet, Depthwise convolutional layers are preferred due to memory efficiency and speed. The reduction in the number of parameters and transformations in Depthwise Separable Convolutional layers makes it more likely to generalize. The kernel size of these convolutions is 7 with 128 filters and there are residual connections between each iteration of the convolutional layers in the block of 4 layers. The output of this is passed into a Self Attention layer which is modeled after Multihead Attention in Vaswani et al [5]. There are 8 heads used in the QANet implementation [2]. There are residual connections between the input and output of the self attention layer. The output of the self attention layer is passed to a feedforward layer and the output of the encoder block has dimensionality of 128. Layer normalization is used prior to each layer for regularization [2]. Further Stochastic Layer Depth is used on the convolutional blocks within the encoders. This involves bypassing layers with a probability proportional to the depth of the layers. Doing this reduces the average depth of the network during training and provides regularization [7].

3. Context to Query Attention Layer: There is a variant of this layer in many reading comprehension models. I made use of the BiDAF Attention Layer provided with the baseline for this. A similarity matrix is computed by concatenating the context, query along with the dot product between the two. The output of this layer is 4* the hidden dimension (*d = 128*) so it needs to be resized prior to passing into the next encoder blocks [2].

4. Stacked Encoder Blocks: There are 3 repetitions of 7 encoder layers called Stacked Encoder Blocks. These encoder blocks are similar to the one used in step 2 however there are 2 convolutional layers with kernel size of 5. At the end of each repetition the output of the block is taken and concatenated to feed to the output layer. The weights of the encoder blocks are shared between the three repetitions [2]. To fit the model in memory, I experimented with reducing the number of blocks needed in the stacked encoders.

5. Output: The output layer from QANet [2] computes two softmaxes for two different concatenations of encoder outputs. The model predicts the probability of each token in the context being the start or end of an answer. M_0, M_1, M_2 are the intermediate outputs of the Stacked Encoder Block repetition. The loss function is the same used in BiDAF and is the negative sum of log probabilities of the predicted distributions, averaged over all the training examples.

$$p^1 = softmax(W_1[M_0; M_1]), p^2 = softmax(W_2[M_0; M_2]) \tag{1}$$

$$L(\theta) = -\frac{1}{N}\sum_{1}^{N}[log(p^1_{y^1_i}) + log(p^2_{y^2_i})] \tag{2}$$

Except step 3, I coded all the other layers including the Depthwise Convolutional Layer as well as the overall model. The baseline model for the default final project is the BiDAF model without character embeddings. The model has an EM of 55 and F1 of 58 on the dev set.

4 Experiments

4.1 Data and Evaluation Metrics

The dataset used for this is the SQuAD reading comprehension dataset. The data is a paragraph, a question about the paragraph and the goal is to answer the question. The task is measured by Exact Match (EM) and F1 scores. The Exact Match is whether the output of the model exactly matches the answer. The F1 score is 2 * precision * recall / (precision + recall). Apart from those metrics we also measure the negative log likelihood loss as well as AvNA which provides a measure of classification accuracy for answer vs. no-answer predictions.

4.2 Training Procedure

Most of the training parameters were based off QANet. The ADAM optimizer was used with $\beta_1 = 0.8, \beta_2 = 0.999, \epsilon = 10^{-7}$. A constant learning rate of 0.001 was used and an exponential moving average is applied with decay rate 0.9999.

The BiDAF and smaller QANet experiments were run on NC6 while the larger QANet models were run on NC12 as the models were significantly larger in memory. A batch size of 16 was used for all experiments. I found that the QANet training runs took 55 minutes per epoch.

4.3 Results

Some ablation analysis tests are done as part of the results to debug potential issues with the model.

BiDAF with Character Embeddings The first experiment was an adapation of the existing BiDAF model to include trainable character embeddings that are initialized with the GloVe embeddings (64 dimensions). The cumulative word and character embedding dimension is (300 + 64 = 364). I also switched the optimizer to Adam with a constant learning rate of 0.001.

Making those adjustments produces dev set results that beat the Stanford BiDAF No Answer (single model) on SQuAD 2.0 (EM 60 and F1 63).

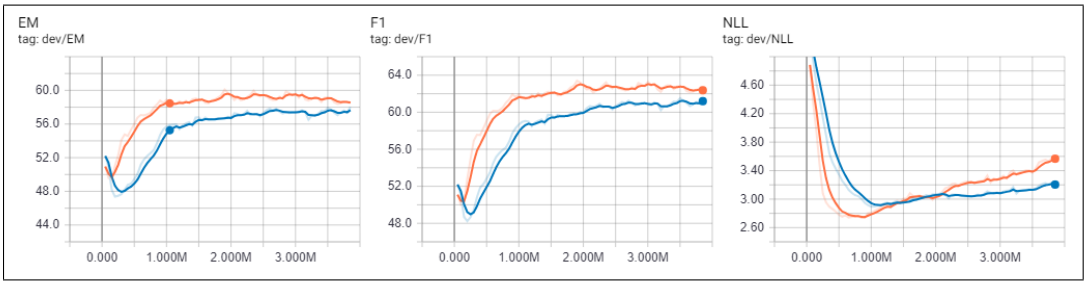

Figure 1: BiDAF with Character Embeddings: From the right, the graphs are produced on the dev set and represent EM, F1 and the loss. The orange curve represents BiDAF with word + character embeddings and the blue curve represents the baseline BiDAF.

QANet with Single Encoder Layer I ran a smaller version of QANet with a single Encoder Layer in the Stacked Encoder Block above the Context to Query Attention Layer. The loss on both the train and dev set reduce for the first million steps however afterwards the dev loss increases wile the train loss continues to decrease. The model's EM and F1 decreased slightly over the course of the run to 46 and 49 respectively. While the upward turn in the dev loss indicated overfitting, the training loss clearly wasn't low enough and the downward trend of EM and F1 indicated that the capacity of the model wasn't sufficient to accurately capture the task. I also used a smaller word + character embedding space (128) so that may also have led to poor performance.

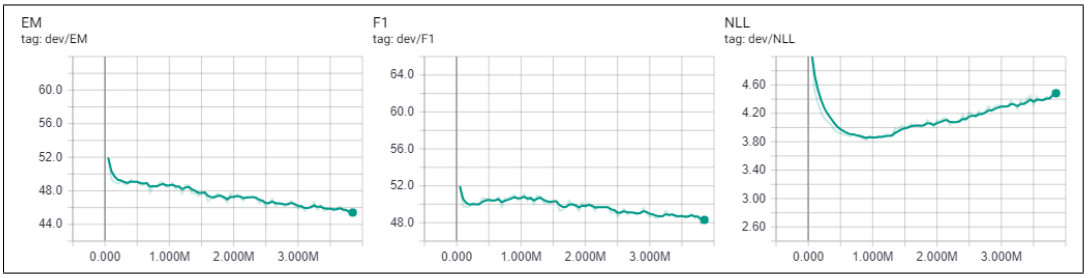

Figure 2: QANet with Single Encoder Layer. From right the plots represent EM, F1 and the negative log likelihood loss. The model starts overfitting after 1 million steps.

QANet with 7 Encoder Layers To increase the capacity of the model, I introduced 7 encoder layers in the Stacked Encoder Block. I continued to use the reduced embedding dimension (128). The dev and train loss reduce in sync down to 3.8 after 1 million steps. The EM and F1 reduce initially but later on F1 increases upto 51. The model clearly doesn't overfit since the dev loss almost perfectly matches the train loss. However the lack of 'learning' as measured by EM and F1 leads me to believe that there is a bug in the code or that the embeddings are not representative of the data in SQuAD.

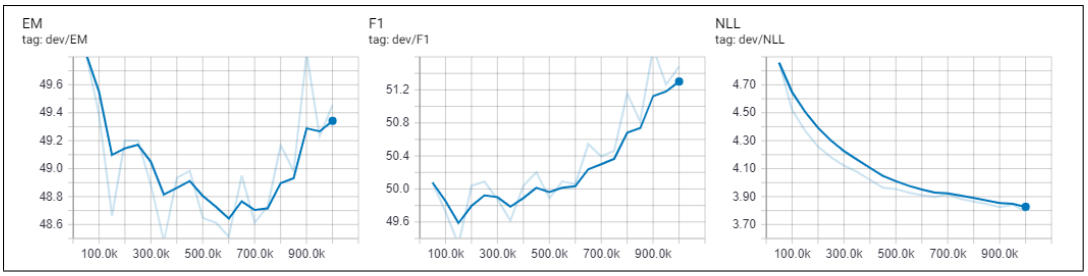

Figure 3: QANet with 7 Encoder Layers: The EM and F1 increase after an initial period of decreasing. However the overall scale is still fairly small so it is likely to be noise.

QANet with 7 Encoder Layers and Larger Embeddings This experiment showed very similar results to the previous one. Introducing larger embeddings showed the similar EM and F1 (49.5, 50 respectively) as the experiment with smaller embeddings. The train and dev loss decrease consistently but that might be due to the ability to perform well on N/A answers. This experiment was run for 400 k iterations but was halted due to lack of measurable gain.

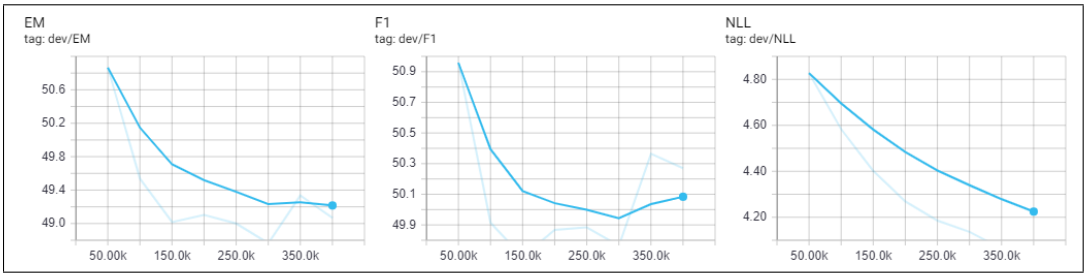

Figure 4: QANet with 7 Encoder Layers and Larger Embeddings. The larger embeddings didn't have much of an impact on model performance in EM and F1. The loss follows a similar trend.

QANet with no Encoder Layers To understand why the network is not able to learn the task, a network with just embeddings, context to query attention and the output is used. The model clearly performs worse than the QANet with all the encoder layers but not as much worse as expected. The dev loss starts overfitting at 350k steps but EM and F1 scores fall very rapidly to 47 and 48 respectively. The training loss keeps dropping even after 500 k steps clearly indicating that more regularization may be needed. This experiment also indicated that the issue was the encoder layer.

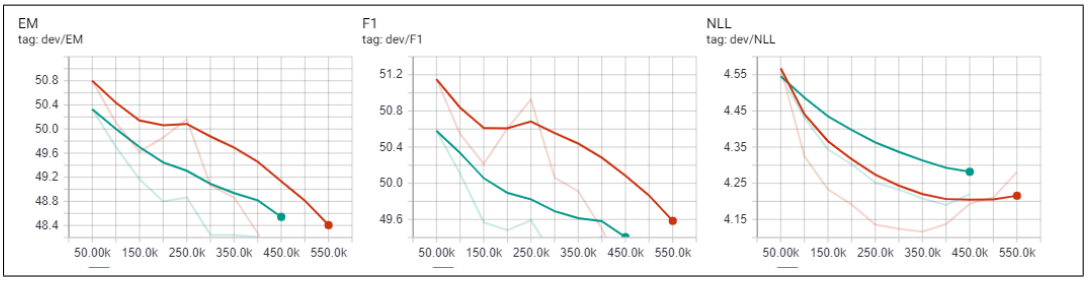

Figure 5: QANet with no Encoder Layers (Red). Removing encoders only produces a minor drop in performance compared to the model with all 7 encoder layers (Green)

QANet Encoders without Positional Encodings For this experiment, a single encoder block was added prior to the context to query attention layer and no encoder blocks in the stacked encoder blocks. Digging into the Encoder Block, the first experiment was to remove the positional encoder and compare it to the performance without it. The positional encodings are summed with the input to the Encoder layer. The addition of the positional encoder does seem to improve the performance slightly but not as much as expected. F1/EM scores start higher (51/51) with the positional encoders

but start at 49.5/49.5 without them. Debugging the positional encoder more closely found that the encodings were accurate and were not the root cause of the poor F1 score.

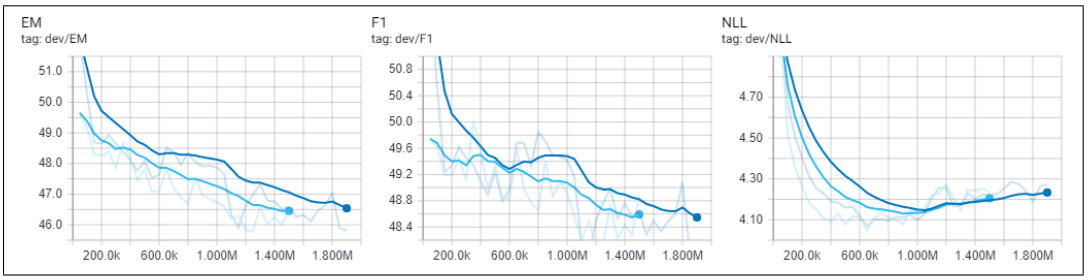

Figure 6: QANet Encoders with (Dark Blue) and without (Light Blue) Positional Encodings. The lack of positional encodings drops performance slightly.

QANet Encoders without Multihead Attention The other major component in the encoder layer is multihead attention. The positional encodings were added back and an experiment was run without multihead attention. Without multihead attention the model starts to overfit very quickly and F1 and EM fall to 48 and 47 within 450 k steps which is much faster than with the multihead attention. This concluded that the multihead attention layer wasn't the issue.

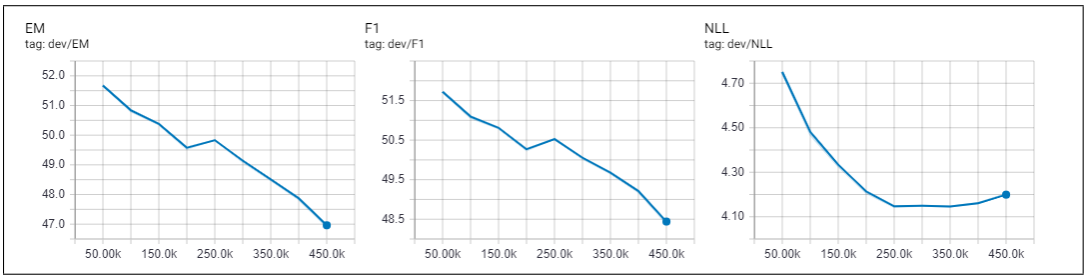

Figure 7: QANet Encoders without Multihead Attention. F1 and EM scores fall faster than with Multihead Attention.

QANet Encoders with Character Embeddings Dimensions from Original QANet Paper At this point most of the components of the network were debugged and to resolve the issue, I tried to get the model more closely inline with the implementation in the original QANet paper [2]. I used pretrained GloVE embeddings for the character embeddings which have a dimension of 64. After passing through a convolutional layer this gets appended to the 300 dimensional word embedding to form a final embedding size of 364. The paper uses a 200 dimensional character embedding which is appended to the 300 dimensional word embeddings and reshaped to the hidden dimension of the model. In both approaches the embeddings are resized to the hidden dimension (128) of the model. Using the paper's approach I ran an experiment and discovered that the model drastically overfits after 500 k steps. The large size of the character embeddings may prevent generalization from occurring on words that produce <UNK> on the word embeddings.

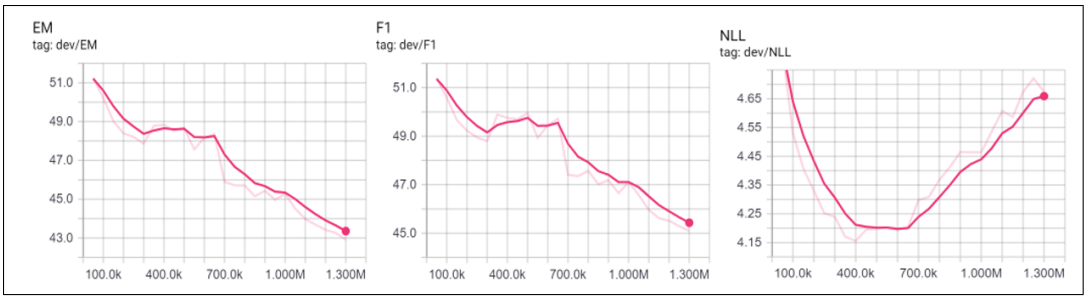

Figure 8: QANet Encoders with Larger Character Embeddings. This approach is more inline with the original paper.

QANet Encoders with Altered Feed Forward Layers and Stochastic Depth Dropout A second addition that was made was to change the feed forward layers to have two linear layers separated by a ReLU in the encoder block. Further an alteration was made to the stochastic depth dropout to dropout convolutional layers instead of dropping out activations. The resulting F1 and EM scores however matched earlier runs where they start off at 50 and drop to 45. Once again validation loss starts showing signs of overfitting once the loss hits 4.10.

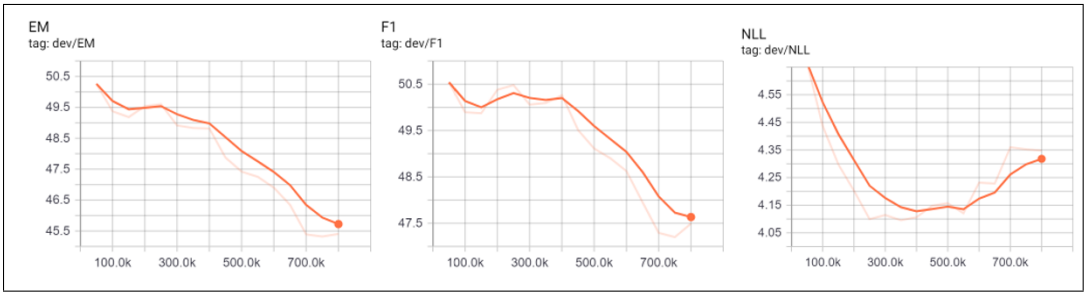

Figure 9: QANet Encoders with Altered Feed Forward Layers and Stochastic Depth Dropout

The below table summarizes the best result from each model.

Models	F1	EM
BiDAF (baseline)	58	55
BiDAF (with character embeddings)	63	60
QANet (No Encoders)	50.5	50.3
QANet (Single Encoder with Positional Encodings and Multihead Attention)	51.2	50.8
QANet (Single Encoder without Positional Encodings)	49.8	49.5
QANet(Single Encoder without Multihead Attention)	51.5	51.5
QANet(All Encoders)	52	51

Based on the results obtained from the original paper, the results from the above experiments are definitely worse than expected. It is interesting to see that character embeddings can provide a good boost regardless of the architecture by providing a source of trainable embeddings. Based on the ablation tests above it is likely that the model produced worse due to the encoder. Given the "lack of learning" in terms of F1 and EM, it appears as though the dropout or objective function could be causing this.

5 Analysis

Typically when a model is not performing regardless of architectural improvements, addition of more layers and various learning rate schemes, it means that something could be wrong with the masking or the objective it is trying to optimize. To figure out if masking is an issue, I visualized the heat maps for the self attention in the encoder block for both the context and the query. These maps are

taken from the first encoder blocks prior to context to query attention. The computed heat maps show that the words in the sequence are not attending to the padding characters, which is a positive sign. It is also interesting to see the difference between the heat maps from the first batch to one after 5 epochs. Initially for each word, the model seems to attend on all the other words equally but later on the attention maps looks more sparse and only one or two words are attended to. Further looking at the context attention heat map, we see that words in the middle of the context were attended to more closely for all words. This might be because in the training data set most answers may be found in the middle of a context so the model is trained to focus its context representation on the middle words.

This is the example on which the heat map was produced: *Context: It is conjectured that a progressive decline in hormone levels with age is partially responsible for weakened immune responses in aging individuals. Conversely, some hormones are regulated by the immune system, notably thyroid hormone activity. The age-related decline in immune function is also related to decreasing vitamin D levels in the elderly. As people age, two things happen that negatively affect their vitamin D levels. First, they stay indoors more due to decreased activity levels. This means that they get less sun and therefore produce less cholecalciferol via UVB radiation. Second, as a person ages the skin becomes less adept at producing vitamin D. Question: What type of immune cells help to destroy abnormal cells in tumors?*

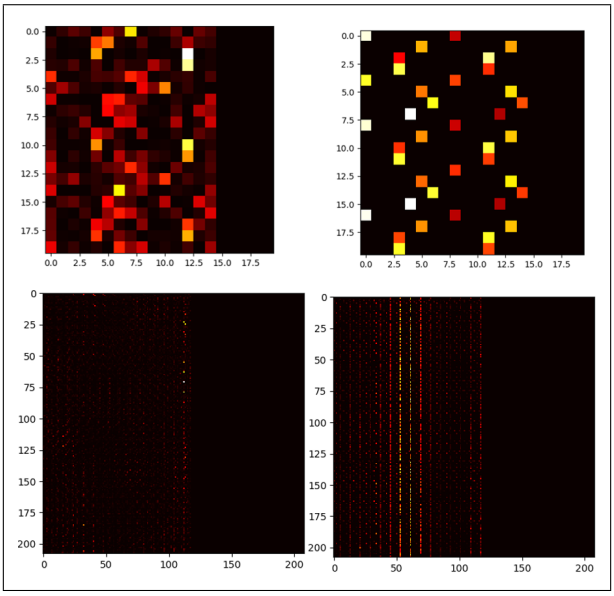

Figure 10: (Top Left) Attention weights from query encoder in first batch. (Top Right) query encoder after 5 epochs. (Bottom Left) context encoder in first batch. (Bottom Right) context encoder after 5 epochs

To look at whether the objective is an issue, I looked into some sample answers that the model returns for a given question and context. Based on the loss function, if the model does really well at predicting the start of an answer but not the end, the loss would still go down but F1 and EM would be low. I saw several examples like the one below where the model predicts answers which start correctly but then incorporates too many tail words. While I do think the loss function could be tweaked, it is unlikely to be the root of the issue since BiDAF performs well with the same loss function.

True Answer: economic; Model Prediction: economic stagnation

6 Conclusion

In this paper, I walked through a custom implementation of QANet as a way to speed up training and inference of reading comprehension models. The core differentiator of the model is that is drops all recurrent layers in favor of convolutions and self attention layers. Some of the key findings is that character embeddings in conjunction with word embeddings helps improve model performance regardless of architecture. It appears that in this implementation since the model is capable of overfitting the data, more regularization might be needed in order to allow the loss on the validation set to continue to drop. The use of multihead attention gives a model a clear path to interpretability on examples, as it is easy to see which part of a context or query the model is focused on. However, the model produced by this implementation does not match the performance of the original QANet. It

is clear that there could be potential changes to QANet in order to improve performance. Investigating alternative loss functions which doesn't allow for guessing one end of the answer index but not the other is a good first step. Further some of the improvements talked about in Transformer XL [4] could really help the baseline QANet achieve higher scores on longer contexts. Using a byte pair encoded vocabulary instead of word and character based vocabularies could allow the model to generalize better. Further, training on synthetic data as described in the paper may help give the model a boost in performance.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee: "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", 2018; [http://arxiv.org/abs/1810.04805 arXiv:1810.04805]

[2] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi: "QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension", 2018; [http://arxiv.org/abs/1804.09541 arXiv:1804.09541].

[3] Yann N. Dauphin, Angela Fan, Michael Auli: "Language Modeling with Gated Convolutional Networks", 2016; [http://arxiv.org/abs/1612.08083 arXiv:1612.08083]

[4]Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le: "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context", 2019; [http://arxiv.org/abs/1901.02860 arXiv:1901.02860]

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser: "Attention Is All You Need", 2017; [http://arxiv.org/abs/1706.03762 arXiv:1706.03762]

[6] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi: "Bidirectional Attention Flow for Machine Comprehension", 2016; [http://arxiv.org/abs/1611.01603 arXiv:1611.01603]

[7] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra: "Deep Networks with Stochastic Depth", 2016; [http://arxiv.org/abs/1603.09382 arXiv:1603.09382]

Appendix

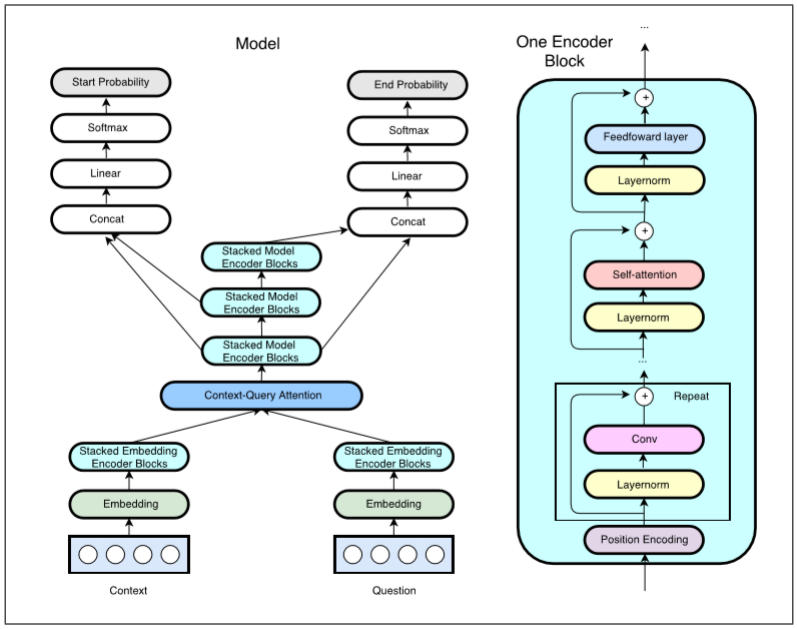

Figure 11: (Left) Full Network architecture of QANet. (Right) One of the Encoder Blocks in the network