Positioning Self in SQuAD

Ankit Baghel
abaghel@stanford.edu

Neel Yerneni
nyerneni@stanford.edu

Abstract

Current state of the art NLP models for reading comprehension tasks entail extremely large and complex neural architectures to achieve optimal performance. Many of these models capitalize on the sequential nature of language by employing RNNs to encode and capture meaning in the input space. However, these models have been shown to be relatively slow, limiting both training and performance speeds. More specifically, much of contemporary research has focused on abandoning recurrent models in favor of ones that scale more efficiently as the training data or input size grows [6]. Our paper focuses on building a question answering system that is both accurate and efficient so that its knowledge can grow unimpeded by computational costs. We build on the already proofed BiDAF architecture while building towards swapping out the usage of RNN-based encoding layers with other types of encoding blocks. We take inspiration from QANET and BERT that primarily make use of convolutional and attention based frameworks. Our aim is to significantly improve both the efficiency and accuracy of the baseline BiDAF model. We found that replacing encoding blocks with GRU's instead of LSTM's not only improved performance speed but also improved accuracy. Incorporating self-attention into the BiDAF framework in addition to positional encodings significantly improved F1 and EM scores on the dev and test sets as well. We conclude by analyzing both the quantitative results and the qualitative implications with respect to our final model architecture.

1 Introduction

Our paper focuses on the NLP task commonly referred to as either question answering or reading comprehension. Given two pieces of text, one representing the query and the other the context paragraph, a model must identify the location of the answer to the query within the context paragraph. Traditionally, the model learns to output the start and end indices of the section of words corresponding to the answer in the context paragraph. There are many potential sources of complexity in this problem with respect to the large degree of variation possible in the query and context. This makes it particularly difficult to build a model that generalizes well. As mentioned, state of the art models entail extremely complex and deep architectures to properly learn the relationships necessary to perform the task well. Newer methods have placed an increased focus on scalability- turning away from slower-to-train recurrent architectures. We will describe some of the techniques used and how we build our models off of them.

2 Related Work

Our baseline consists of the BiDAF model as introduced in [3]. In particular, this paper makes strong use of a mechanism commonly used for NLP tasks known as attention whereby models are trained to learn to focus on more relevant portions of one input with respect to another input. Specifically in this paper, the authors focus on producing attention weighted representations of both the context and the question with respect to one another allowing them to obtain very strong results. Following the development in the use of attention, we also look at R-Net, which introduces the idea of self-attention- to better and more richly capture the notion that certain parts of a context passage shed light on other parts with respect to a question. They proceeded under the idea that an encoding, as done in [3], for example, may not effectively capture longer dependencies between portions of the context passage.

The models from these papers make heavy use of recurrent neural networks to produce encodings of inputs at various steps in the model. QANet was a recently introduced model that experienced a significant speed increase in performance by forsaking RNNs in exchange for CNNs and self-attention. QANET uses repeated "encoder blocks" throughout its model. These encoder blocks contain convolutional layers, self-attention layers, and feed-forward layers which are repeated, stacked, and connected through residual connections [6]. Google's BERT model takes a similar approach in moving away from recurrent encodings of a passage in developing a transformer. [1] This type of transformer has been shown to be extremely promising for reading comprehension tasks. It is made up of a multi-headed self-attention layer followed by a fully connected feed-forward layer and grouted with layer normalizations and residual connections.

3 Approach

We set out to extend the baseline BiDAF model for SQuAD using ideas such as, but not limited to, self-attention, convolutional layers, and positional encodings to improve not only F1 and EM scores but also the speed of training and prediction. We do this by approaching our implementations from two fronts. The first front focuses on building up the accuracy of our SQuAD answering model. The second focuses on building tools such as QANET derived encoding "blocks" to replace RNNs and other computationally expensive aspects of these implementations to speed up training while preserving accuracy.

3.1 Character Level Embeddings

The baseline model uses only pretrained GloVe word-level embeddings. Our first step, therefore, was to add character level embeddings. Fortunately, pretrained character-level embeddings are provided as a json file. Similar to how the baseline uses the provide word-level embeddings file, we load in the pretrained character embedding layer without freezing it, run the context and question character indices through the embedding layer to create a tensor of shape (batch_size, seq_len, char_embed_size), and then use 1D convolutions to convert the character level embeddings to shape (batch_size, seq_len, word_embed_size). The word-level and character-level embeddings are then concatenated along the last axis to create a final embedding of shape (batch_size, seq_len, 2 $\times$ word_embed_size). The following linear projection and highway encoder layers then convert this tensor to a final embedding of shape (batch_size, seq_len, hidden_size).

3.2 Self-Attention

Our next goal was to add the mechanism of self-attention to our model as originally described in the R-NET paper to better and more richly capture the notion that certain parts of a context passage shed light on others with respect to a question.[2] We chose to implement additive attention whereby the attention weight matrix a is computed by summing two products each consisting of the input and a learned weight matrix which is then followed by another multiplication with the input and then softmax over the last dimension. We then multiply this matrix with the original input to obtain an attention weighted representation of said input.

$$s = p^T(W_1 p + W_2 p)$$
$$a = SoftMax(s)$$
$$q = ap$$

We finally concatenate the original input to the output and gate it with a learned gate g.

$$\begin{aligned} o &= [p, q] \\ g &= \sigma(W_g o) \\ o &= g * o \end{aligned}$$

In later models, we add positional encodings as described in [5]. The following is added to the inputs of the self-attention layer before applying the vanilla self-attention

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{\frac{2i}{d_{model}}}})$$

where pos is the position of the word in the sequence, d_{model} is the hidden size of the model, and i is the relative position in the embedding dimension (which has length hidden size).

3.3 QANET Encoder Block

Our "EncoderBlock" starts by adding a fixed positional encoding to the input as described in Vaswani et. al..[5] The input is then passed through stacked layer norm plus depth-wise separable convolutional layers. Depth-wise separable convolutional layers were used to reduce computational costs [6]. The number of stacked layer norm and convolutional layers varies depending on whether the EncoderBlock is applied before or after BiDAF attention. Four are stacked if before and two are stacked if after to allow more information to pass through the bidirectional attention layer. Dropout is also applied after every other stacked layer norm and convolutional layer. Residual connections feed-forward the positional-encoding-added input to the output of the stacked layer norm and convolutional layers. Layer dropout as described in [6] is also used in every residual block whereby the output of some residual block has a probability of being ignored proportional to the number of layers in the model.

The output of the stacks are then put through a self-attention layer that uses multi-headed self-attention [5] using one or four heads. Lastly, two feed-forward layers are applied followed by one last layer norm. Similar to the stacked convolutions, the self attention and feed-forward components each respectively have residual connections from their input to their output along with the associated layer dropout. The EncoderBlock ultimately produces output with the same shape as the initial input.

3.4 QANET Model Encoder and Output

Seven stacked EncoderBlocks with two convolutions per block is run over the output of the context2query attention layer (we used the baseline BiDAF attention layer) three times using the same shared weights each time. The outputs of each model encoding are M1, M2, and M3 respectively. In the output layer, M1 and M2 are concatenated along the hidden size dimension, are run through a 1D convolutional layer that changes the last dimension to size one, and are then run through a log_softmax to produce the prediction for the start of the answer span. Similarly, M1 and M3 are concatenated and are used to produce the prediction for the end of the answer span.

3.5 Integration of QANET into BiDAF

Our goal was to test to see if including encoder blocks in place of some RNNs can preserve accuracy while speeding up training and performance. Encoder RNNs before the baseline BiDAF attention layer would each be replaced with a single EncoderBlock for both context and question. The output of these two EncoderBlocks would then feed into the BiDAF attention layer. The RNNEncoder immediately after the BiDAF attention layer would then be replaced with a one-dimensional convolutional layer to change dimensions of the output followed by seven stacked EncoderBlocks (QANET Model Encoding Block). The output from these stacked EncoderBlocks is then run through the BiDAFOutput layer. We developed the model using this integration scheme but were ultimately unable to test it due to difficulties verifying correct implementation of the encoder block.

3.6 Other layers

Other layers such as the BiDAF attention, RNN_Encoder, and BiDAF Output are left unaltered with the exception of swapping any LSTMs in the RNN_Encoders and BiDAF Output layers with GRUs.

It should also be noted that all additions to the provided baseline BiDAF model have been implemented by us using descriptions of self-attention, transformers, and QANET's convolutional transformer variant in their respective cited literature. The only exception to this is the use of positional encodings and multi-headed attention, which we implemented while referencing the Transformer repo (https://github.com/jadore801120/attention-is-all-you-need-pytorch) that was written for the WMT 2014 English-to-German translation task and not for SQuAD. As such, significant time was required to debug and test new layers before adding them to our models.

4 Experiments

4.1 Data

As described in the default project handout, the dataset we are using is roughly 150,000 questions associated with paragraphs from Wikipedia. About half of the questions are unanswerable if using only their associated paragraphs. Our SQuAD models are meant to predict whether the answer to the question is within the paragraph and if it is, point to the span of text containing the answer. [4]

4.2 Models

The "Baseline" model is a vanilla BiDAF model without character-level embeddings as described in [4]. The "Baseline + Char Level Embeddings" model is this same baseline except with character level embeddings concatenated onto the word-level embeddings before being run through the highway encoder. This model is meant to resemeble the "BiDAF-No-Answer (single model)" on the Squad 2.0 leaderboard.

Next, we augmented this model by incorporating self attention as described in the R-Net paper. [2] This was done by adding a self-attention layer followed by an RNNEncoder layer in one of two locations within the BiDAF model. In the "Self_Attention_Before_BiDAF" model, this addition is inserted right before the BiDAF attention layer. In the "Self_Attention_After_BiDAF" model, the addition is inserted after the Modeling layer of the vanilla BiDAF. Since both of these models add an RNN to the vanilla BiDAF to process the output of self-attention, we decided to convert all LSTMs within the model to GRUs to limit the increase in computational costs. In our results, we discuss the relative performance of using LSTMs or GRU and find that GRUs outperform LSTMs. We also add positional encodings to the self-attention layer in the second phase of experiments.

We were also interested in adding QANET encoder blocks to our BiDAF model to determine if the switch from RNN to convolutions would speed up training. However, in order to test our implementation of the encoder blocks, we decided to implement the vanilla QANET as described in [6] and see if it performs as expected.

4.3 Experiment Details and Evaluation Methods

Training experiments generally ran for thirty epochs except in later experiments where early stopping was utilized when metrics have converged. An Azure NV6 was used to run all experiments. BiDAF was trained with a learning rate of 0.5, a hidden dimension of 100, and a dropout probability of 0.2. All other hyperparameters are as described in [4].

Our QANET model was run with the hyperparameters described in [6] with the exception of hidden size 96, number of heads for multi-headed attention being either one or four, and a learning rate of 0.001 or 0.0005 (default learning rate = 0.001).

We evaluated the model using F1, EM, and AvNA scores. The F1 score computes a harmonic mean between the precision and recall scores using the overlap of words between the prediction and ground truth; the EM score measures the percentage of exact matches to the ground truth data. Given that the data contains queries with no answer, the AvNA score computes classification accuracy only with respect to whether the existence of an answer was correctly predicted. As efficiency of our model is another axis we hope to optimize, we also want to log training time per epoch. We use and track the sum of negative log-likelihood loss (NLL) of the predicted start and end points of answer spans during training.

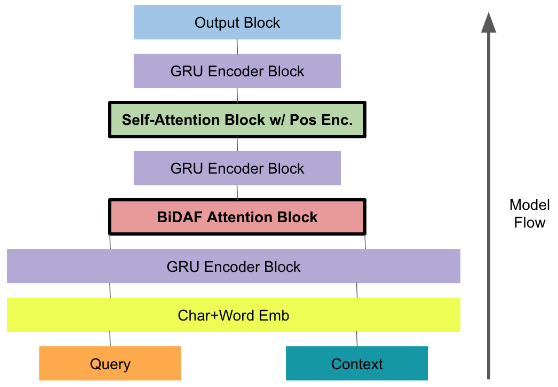


Figure 1: Final BiDAF Model

4.4 Results

In the first phase of our project, we experimented with the addition of character-level embeddings and self-attention. We observed the following results on the Dev Non PCE Leaderboard:

Model	Epoch Training Time	Dev F1	Dev EM	Dev AvNA	Dev NLL
Baseline	18 min	61.29	57.79	68.38	3.1
Baseline + Char Level Embeddings	24 min	63.78	60.19	70.69	3.13
Self_Attention_Before_BiDAF	49 min	63.62	59.74	70.43	3.01
Self_Attention_After_BiDAF	41 min	65.07	61.75	71.4	2.79

Table 1: Phase 1 Results for Dev Set. (Submitted to the Dev Non-PCE Leaderboard)

As expected, all three implemented models were able to outperform the baseline on F1, EM, and AvNA scores. However, since these improvements required increasing the number of trainable parameters, the training time per epochs increased significantly across all models. The biggest increase in training time came with the addition of self-attention, which is expected since adding self-attention requires the addition of a relatively large bidirectional RNN. Self-attention itself adds a few linear layers to the model as well, but the parameters from this addition is small compared to the parameters added by the RNN.

An interesting finding was that while both self-attention layers had large training times, "Self_Attention_Before_BiDAF" took an extra 8 minutes per epoch to train compared to "Self_Attention_After_BiDAF". This is unexpected since the self-attention and subsequent RNNEncoder layer added to both models were initialized identically. The only difference was their placement within the overall model.

Observing our experiments using Tensorboard (please see Appendix for figures), we can note that our three implemented models surpass the baseline model by the 1.000M mark across all metrics. While a significant gain in performance can be attributed to the inclusion of character-level embeddings, we observe that the inclusion of self-attention reduces the number of steps needed to reach improved scores regardless of placement before or after the BiDAF attention layer. This is a promising result since it indicates that self-attention models may need fewer overall epochs to reach their best checkpoint. This idea is further supported by the fact that the "Self_Attention_After_BiDAF" model reaches its best check point at around the 2.000M mark and declines slightly in performance thereafter. Because of this, we used early stopping in later improvements of this model.

In the second phase of our project, we explored ideas related to transformers [5]. We first implemented positional encodings as described in [5]. These were then added to the self-attention layer. We then tested two versions of the Self_Attention_Pos_Encoding_BiDAF model: one using GRUs and another using LSTMs. We found that using GRUs instead of LSTMs resulted not only in slightly faster training times but also had better F1 scores. Because of this result, we elected to use GRU as the only type of RNN in later models.

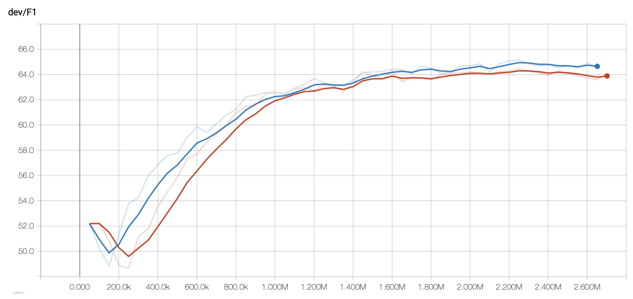


Figure 2: F1 Scores between Self_Attention_Pos_Encoding_BiDAF models: blue is the GRU model and red is the LSTM model.

Our implementation of QANET [6] was plagued by many iterative rounds of testing and debugging. During this process, we were able to experiment with some of the hyperparameters and their effect on training performance. Consider the following three versions of QANET

Model	Learning Rate	Num Heads
QA_1	0.001	1
QA_2	0.001	4
QA_3	0.0005	4

Table 2: Hyperparameter tuning of QANET

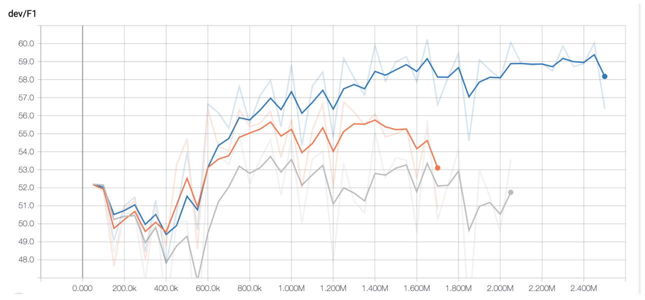


Figure 3: F1 Scores between QA models: grey is the QA_1, orange is QA_2, and blue is QA_3.

We observe an increase in F1 score when increasing the number of attention heads from one to four. This is as expected since increasing the number of attention heads allows the model to attend to information from even more subspaces of representation.

We also observe a significant increase in F1 score when dividing the vanilla QANET learning rate by two. The reason for this increase is likely due to a caught bug by which parameters were normalized and layers were dropped too often. This results in a smaller learning rate being required in order to stabilize learning and avoid the spikes seen in QA_1 and QA_2.

4.5 Final BiDAF Model Results

Model	Epoch Training Time	Dev F1/EM	Test F1/EM
Self_Attention_Pos_Encoding_BiDAF	41 min	65.528/62.040	64.074/60.575

Table 3: Final Model Results on Non-PCE Leaderboards

We find that the addition of positional encodings increases metrics without adding any significant training time to the model. This makes sense as positonal encodings for all training examples is precomputed during initialization and stored, indicating that taking inspiration from transformers to improve BiDAF was a good approach.

5 Analysis of Final BiDAF Model

5.1 Question and Context Lengths

We first explored the effect question length and context length had on dev set scores. We found that there was no clear correlation between length and metrics for either length. Please see the Appendix for figures and discussion.

5.2 Answer Length

Next, we explored the effect true answer length had on our metrics.

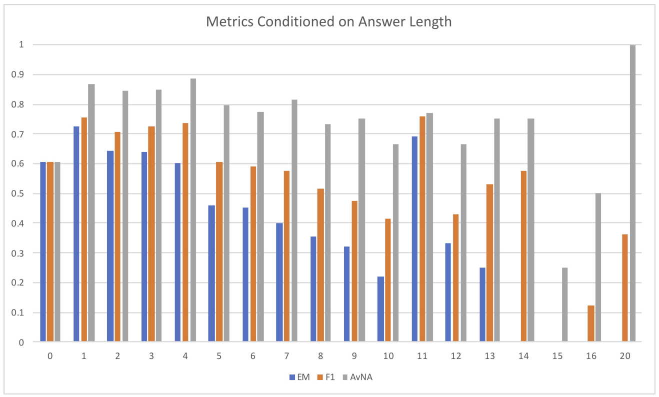
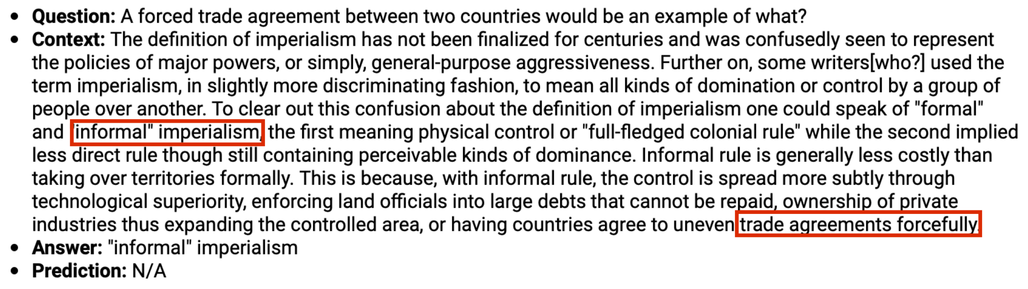


Figure 4: Average scores subsetted on answer length. EM is in blue. F1 is in red. AvNA is in grey.

Here, we observe a downward trend in all metrics as the length of the true answer increases. This makes sense due to how RNNs and even self/multi-headed attention mechanisms have a limited window in which pertinent information can flow in a sequence. If an answer span is relatively long, then it is difficult for attention mechanisms to place the correct amount of attention on far away words from the current center word. It is likely that increasing the number of self attention layers or increasing to eight heads of attention would help this issue in future models.

5.3 Error Analysis

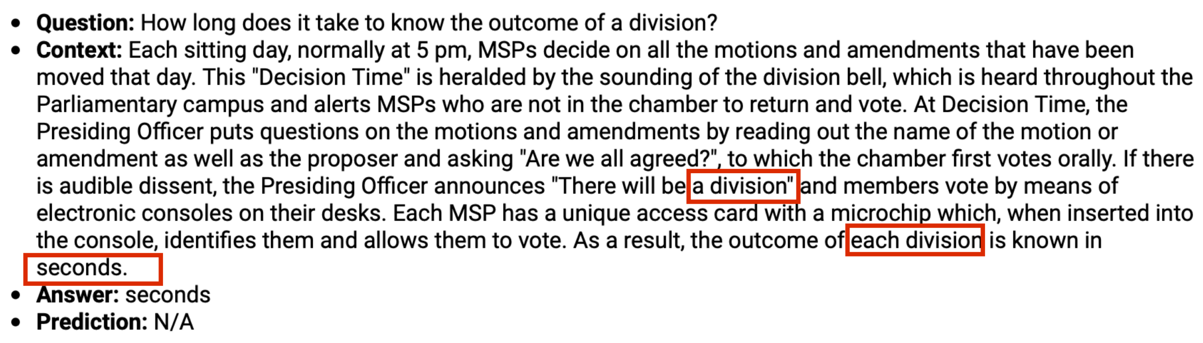
Lastly, we analyze questions that our final BiDAF model gets wrong as well as questions its outperforms on compared to the baseline in order to better understand how to improve models in the future. First consider the following question that our final BiDAF model gets wrong.



- **Question:** A forced trade agreement between two countries would be an example of what?
- **Context:** The definition of imperialism has not been finalized for centuries and was confusedly seen to represent the policies of major powers, or simply, general-purpose aggressiveness. Further on, some writers[who?] used the term imperialism, in slightly more discriminating fashion, to mean all kinds of domination or control by a group of people over another. To clear out this confusion about the definition of imperialism one could speak of "formal" and "informal" imperialism, the first meaning physical control or "full-fledged colonial rule" while the second implied less direct rule though still containing perceivable kinds of dominance. Informal rule is generally less costly than taking over territories formally. This is because, with informal rule, the control is spread more subtly through technological superiority, enforcing land officials into large debts that cannot be repaid, ownership of private industries thus expanding the controlled area, or having countries agree to uneven trade agreements forcefully.
- **Answer:** "informal" imperialism
- **Prediction:** N/A

In this question, the correct answer is "informal imperialism", but our model predicts no answer. The reason for this is that the phrases "informal imperialism" and "uneven trade agreements forcefully" are located far apart within the sequence. While RNNs can capture sequential information, they can only capture a finite length away from the query word. The addition of self-attention likely improves upon this issue, but further models would certain benefit from multi-headed attention, which provides a more "global" version of self attention that provides more rich sequential information than RNN and self attention alone.

Now consider the following question:



- **Question:** How long does it take to know the outcome of a division?
- **Context:** Each sitting day, normally at 5 pm, MSPs decide on all the motions and amendments that have been moved that day. This "Decision Time" is heralded by the sounding of the division bell, which is heard throughout the Parliamentary campus and alerts MSPs who are not in the chamber to return and vote. At Decision Time, the Presiding Officer puts questions on the motions and amendments by reading out the name of the motion or amendment as well as the proposer and asking "Are we all agreed?", to which the chamber first votes orally. If there is audible dissent, the Presiding Officer announces "There will be a division" and members vote by means of electronic consoles on their desks. Each MSP has a unique access card with a microchip which, when inserted into the console, identifies them and allows them to vote. As a result, the outcome of each division is known in seconds.
- **Answer:** seconds
- **Prediction:** N/A

In this question, the baseline could not find the correct answer, so it predicted no answer. Our final BiDAF model, however, was able to predict the correct answer. The reason the baseline fails is likely due to the phrasing of the two instances the word "division" appears in the context. In the first, it is "a division" and in the second, it is "each division." Because the question states "a division", the baseline model likely focuses in on "a division" as the center word to look at when in reality, the answer is located near "each division." Our BiDAF model correctly predicts the answer because it implements self-attention, which allows it to better evaluate the relative importance of words in the context. Therefore, it can detect that while "each division" does not match the question stem, it is located right next to the answer.

6 Conclusion

In this study, we were able to improve upon the vanilla BiDAF model by experimenting with self attention and positional encodings. We were also able to improve performance while minimizing the increase in computational costs by using GRUs instead of LSTMs within our model. We were also able to make a decent improvement in performance using positional encodings with virtually no increase in training time.

While it is unfortunate that we were unable to fully implement the vanilla QANET, we were able to borrow ideas from it such as positional encodings to improve our self-attention BiDAF models. We were also able to make incremental improvements on our QANET implementation through debugging and through hyperparameter tuning. As of writing this paper, we have implemented a version of QANET whose score metrics grow more smoothly. (Please see appendix for F1 score graph) It still does not beat baseline BiDAF, but debugging it this far has been a worthwhile exercise in implementing machine learning algorithms on PyTorch and in experimenting with multi-headed attention and layer dropout hyperparameters.

A limitation of this study is that it improves performance at the cost of speed. Once we fix our QANET implementation, we will be able to explore how convolutional stacks can improve the speed of our BiDAF model. We also spent some time implementing the layers needed for the traditional Transformer in [5] but put it aside in order to focus on the QANET debugging. We plan on returning to the tradtional Transformer so we can use it to evaluate the speed increases provided by QANET integration into BiDAF. Ultimately, the addition of "traditional" transformers or EncoderBlocks will provide us with the opportunity to thoroughly explore the trade-off between accuracy and efficiency within the BiDAF architecture.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[2] Microsoft Research Asia Natural Language Computing Group. R-net: Machine reading comprehension with self-matching networks. May 2017.

[3] Minjoon Sero, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidrectional attention flow for machine comprehension. *ICLR 2017*, 2017.

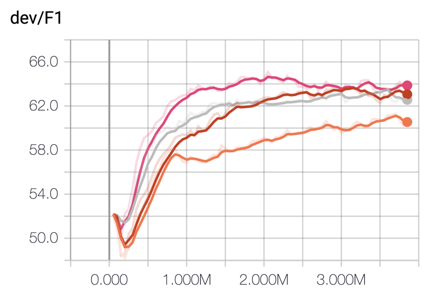
[4] CS224n Teaching Staff. "cs 224n default final project: Question answering on squad 2.0", 2019.

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

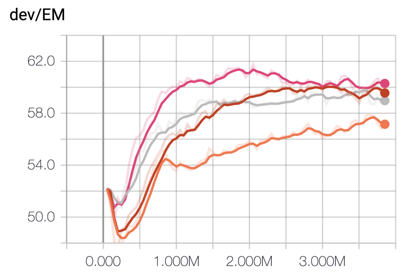
[6] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension, 2018.

7 Appendix

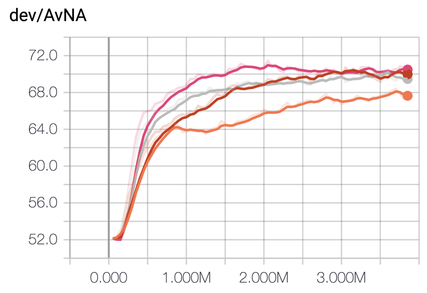
7.1 Phase 1 Models Training Metrics



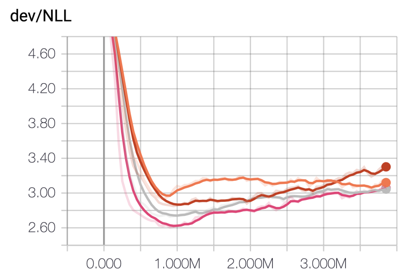
(a) Dev F1



(b) Dev EM



(c) Dev AvNA



(d) Dev NLL

Figure 5: Phase 1 Training Metrics: The orange line represents the Baseline model. The red line represents the Baseline + Char Level Embeddings Model. The grey line represent the Self_Attention_Before_BiDAF model. The magenta line represent the Self_Attention_Before_BiDAF model.

7.2 Question Length Metrics

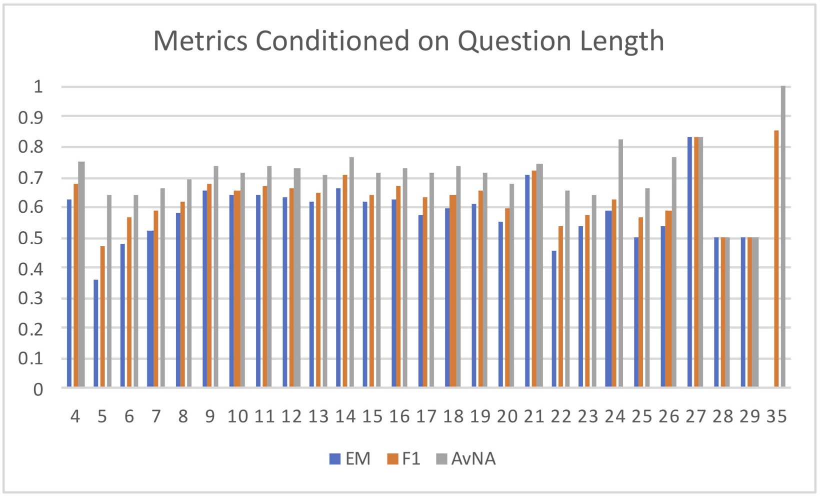


Figure 6: Average scores conditioned on question length. EM is blue. F1 is red. AvNA is grey

We observed that there is no significant trends any of the metrics measured. There is a slight improvement in EM and F1 when considering shorter length questions however. While it appears that question length 35 is associated with a AvNA score of 1.0, there was only one question in the dev set that was this long. This brings up a limitation of this figure since the length of questions is sparse in the dev set. A future analysis we would like to do would be to generate this figure from the training set, which has significantly more examples.

7.3 Context Length Metrics

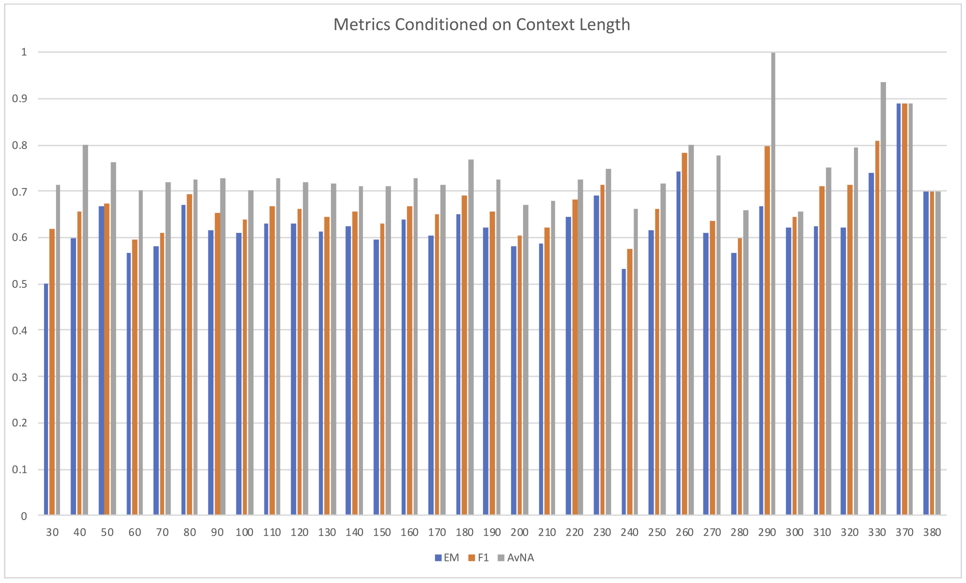


Figure 7: Average scores conditioned on context length. Lengths are rounded down to the tens place. EM is blue. F1 is red. AvNA is grey

Similar to question length, context length appears to have no clear correlation with any of our metrics. There appears to be high scores for the longest contexts, but this is mostly due to sparsity of the dev set context lengths. Our model appears to perform equally well on all context lengths.

7.4 Final QA Model

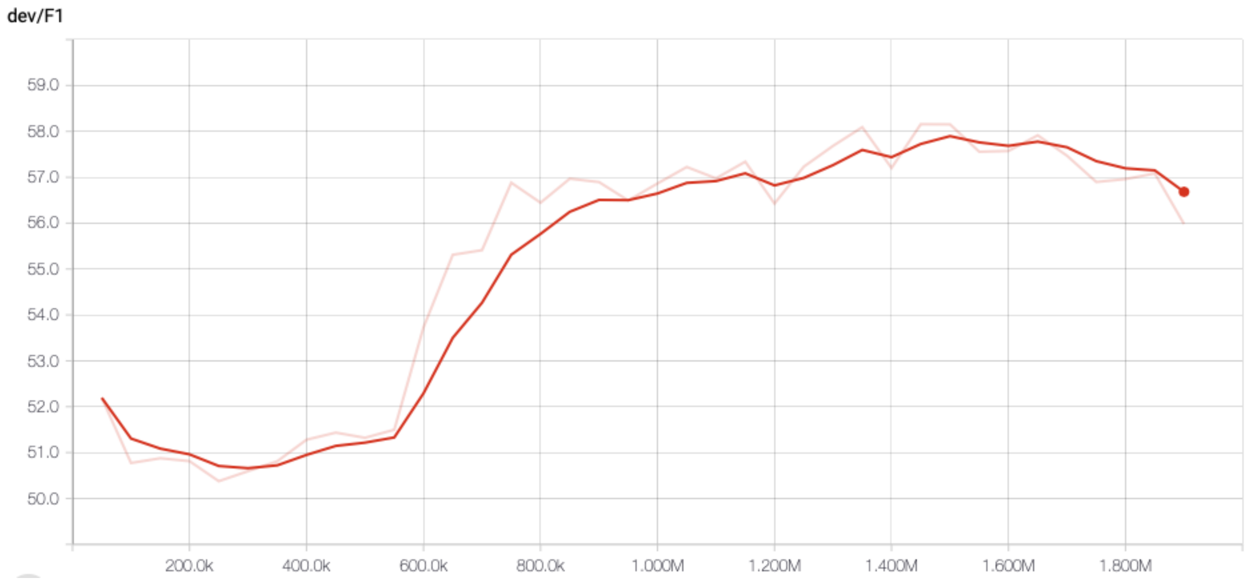


Figure 8: The F1 score of the final QANET implementation during training. While the performance does not coverge to or above the baseline's, it does have a much more smooth learning curve compared to previous iterations of the model, indicating that future work on this model is promising.