QANet-XL: Applying Transformer-XL to QANet

Ryan Cole
Department of Computer Science
Stanford University
`rcole34@stanford.edu`

Abstract

Question Answering has become a classic NLP task, especially over the past few years with the rise of transformers and pre-trained contextual embeddings. On the SQuAD 1.1 dataset, the QANet [10] architecture performed extremely well and produced the majority of the top scores by incorporating aspects of transformers [9]. However, PCE models have overwhelmingly dominated the SQuAD 2.0 leaderboard. My goal in this project is to try to apply new ideas about non-fixed-length context from Transformer-XL [3] to the QANet model in order to improve upon scores. Despite running into some problems in implementing models, I did end up seeing an improvement in performance.

1 Introduction

Question Answering is a task that is fairly straightforward for humans. From a young age, the format is familiar and can be found on countless reading comprehension tests – given this passage, answer the following questions. Naturally, it became a point of emphasis in the NLP community, especially when the SQuAD1.0 dataset was released in 2016 [7]. Initially, the tried-and-true long short-term memory (LSTM) models [5] led the way, but soon other techniques such as bidirectional attention flow (BiDAF) [8] took over the top of the leaderboard. However, with the introduction of Transformers [9] in 2017, the Transformer-based QANet model [10] quickly achieved state-of-the-art performance. With the rise of pre-trained embeddings in 2018, though, BERT [4] managed to even surpass human performance on the SQuAD1.0 dataset.

Clearly the problem needed to be made a little harder. SQuAD2.0 [6] introduced a large number of unanswerable questions to the dataset, and suddenly the models that had performed so well on the old dataset struggled to identify when there was no answer. BERT emerged from the pack as the best at being able to perform well on this dataset too, and as of early March Google AI has produced a BERT-based ensemble model that performs only a couple of tenths worse than human performance.

But what happened to the QANet models that once sat atop the leaderboard? Were vanilla transformers inadequate to accurately identify which questions had no answers, or were they just out-performed by pre-trained models, so no one bothered trying them on SQuAD2.0? My goal for this project was to recreate a QANet model to test its performance on SQuAD2.0, and then try to extend it to improve performance. The most intriguing improvement that I sought to make was to apply the ideas of Transformer-XL [3] get rid of fixed-length context and introduce a recursive structure to allow the Transformer blocks to draw connections between farther apart sections of text. Similar to the improvement that LSTMs and GRUs made over vanilla RNNs, I hoped that this would also improve the performance of a QANet model. In the end, however, as I will describe more later, it seemed that the fact that the context in the SQuAD dataset is of a fixed, fairly short length, limiting the improvements of Transformer-XL.

2 Related Work

There has been a lot of recent work on Question Answering in the NLP community. One of the earlier papers that laid framework and introduced concepts that other SQuAD models have adapted was "Bidirectional Attention Flow for Machine Comprehension" [8]. The BiDAF model introduced the idea of not only having the query attend to the context paragraph, but also performing attention calculations in the opposite direction – with the context attending to the query. This proved to be a useful technique, and BiDAF models are commonly used as a baseline for other SQuAD models.

The main papers that inspired my work were "Attention Is All You Need" [9], "QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension" [10], and "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context" [3]. Vaswani and his team changed the way the NLP community thought about solving problems when he introduced the Transformer and the idea that attention is the key aspect of many models and high levels of performance can be achieved simply by stacking many transformer blocks on top of one another. Yu and others at Carnegie Mellon and Google Brain took this idea and used stacks of transformer-like blocks along with context-query attention to find the most likely start and end positions of an answer. The work of Dai et al. in coming up with Transformer-XL expanded on Transformers by introducing a recursive structure to them and freeing them from the binds of fixed-length context, inspiring my changes to the base QANet model.

More recent work with pre-trained contextual embeddings, such as BERT [4] have further pushed our notion of what is possible to achieve with our models. Recent work has shown that using these pre-trained embeddings, can offer huge improvements in performance, allowing new state-of-the-art benchmarks to be reached across Question Answering and other tasks.

3 Approach

The first step of of approach was implementing a QANet [10] model before attempting to make changes to it to reduce dependencies on fixed-length content using the ideas of Transformer-XL [3]. The QANet architecture is best described through the diagram below from the QANet paper [10].

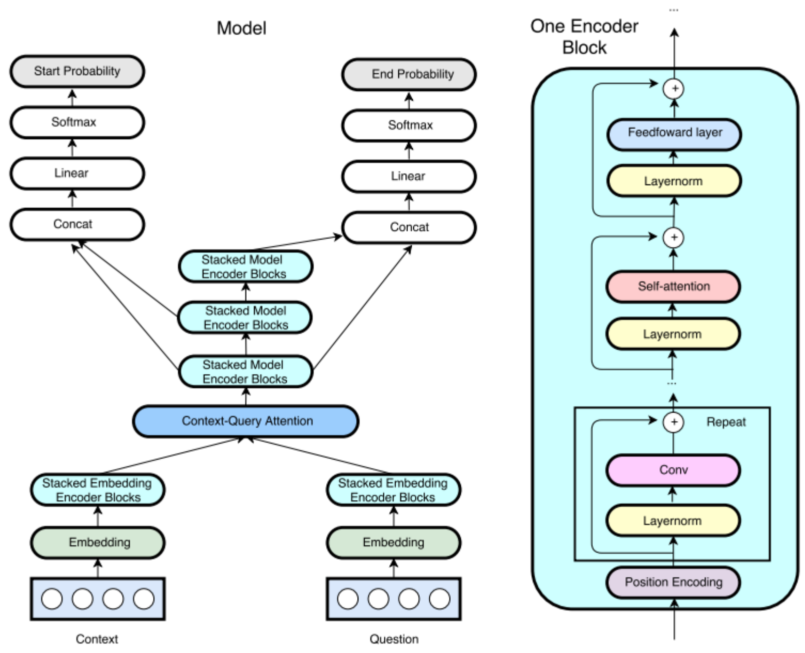


Figure 1: QANet architecture.

The blue encoder block shown enlarged on the right draws from the ideas of Transformers [9] and is repeated at various points in the architecture. Within the block, we get positional encodings, apply a depthwise-separable convolution layer [2], apply self-attention, and finish with a feedforward layer, normalizing with Layernorm [1] at each step. This block is first used to encode the embeddings of both the context and the query. For the context-query attention, I used the attention function supplied in the BiDAF starter code. 3 stacks of encoder blocks are then applied sequentially, with outputs concatenated and put through linear and softmax layers to arrive at start and end probabilities for the location of the answer. I adhered to most of the parameters and specifications in the QANet paper, except for using 4 heads for multi-headed attention rather than 8, 3 encoder blocks per stack instead of 7, `d_model` = 96 instead of 128, and a batch size of 32 instead of 64. All of these changes were so that the model could run on an NV12 virtual machine with 2 GPUs. I spent the better part of 3 days implementing these layers using PyTorch in code that I wrote initially. However, after experiencing slow and poor performance despite double- and triple-checking against the specifications in the QANet report, I ended up referring to and incorporating parts from this QANet implementation in PyTorch from Github. I opted to do this because the main point of my project was to see if Transformer-XL would improve the base model, even though this pushed my model into the PCE Leaderboard category based on the instructor definition of things. Running that version of QANet also had fairly low performance, however, as I later saw on Piazza was the case for most students implementing QANet, so although I was inclined to believe that the model that I had painstakingly implemented myself was close to being correct, I decided to move forward with the other implementation and just try to improve on that model's performance with Transformer-XL features.

Although I knew going in that the typical transformer's fixed context length was still longer than the context of SQuAD passages, I was interested to see if applying the recurrent structure of Transformer-XL would be able to improve performance. There were only a couple of key changes to make to the structure of my QANet encoder blocks to adjust according to the changes Transformer-XL makes to vanilla Transformers. First, I needed to apply the following equations for self-attention to add extended context through recursion:

$$\begin{aligned}
\widetilde{\mathbf{h}}_{\tau+1}^{n-1} &= \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right], && \text{(extended context)}\\
\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} &= \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top}, && \text{(query, key, value vectors)}\\
\mathbf{h}_{\tau+1}^{n} &= \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right). && \text{(self-attention + feed-forward)}
\end{aligned}$$

Figure 2: Transformer-XL self-attention equations.

Then, importantly, we would also need positional encodings to be relative instead of absolute when we no longer have fixed context length. In our attention equations, we now need to replace absolute embeddings U_j with relative embeddings R_{i-j}, where R is a sinusoid encoding matrix. The original absolute attention equation ($A_{i,j}^{abs}$) and the new relative attention equation ($A_{i,j}^{rel}$) are shown below as they appear in the Transformer-XL paper [3].

$$\begin{aligned}
\mathbf{A}_{i,j}^{\text{abs}} = q_i^{\top} k_j &= \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{E}_{x_j}}_{} + \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{U}_j}_{} + \underbrace{\mathbf{U}_i^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{E}_{x_j}}_{} + \underbrace{\mathbf{U}_i^{\top}\mathbf{W}_q^{\top}\mathbf{W}_k\mathbf{U}_j}_{}.\\
\mathbf{A}_{i,j}^{\text{rel}} &= \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_{k,E}\mathbf{E}_{x_j}}_{} + \underbrace{\mathbf{E}_{x_i}^{\top}\mathbf{W}_q^{\top}\mathbf{W}_{k,R}\mathbf{R}_{i-j}}_{} + \underbrace{u^{\top}\mathbf{W}_{k,E}\mathbf{E}_{x_j}}_{} + \underbrace{v^{\top}\mathbf{W}_{k,R}\mathbf{R}_{i-j}}_{}.
\end{aligned}$$

Figure 3: Absolute and relative attention equations.

The paper also presents the option of using a positional feedforward layer. More details on all of these changes can be found in the Transformer-XL paper [3].

While this may be assumed as I did the default project, for completeness I will also mention here that I ran the BiDAF [8] model from the provided starter code as a baseline. More information on that model can be found at the paper cited above or in the project handout.

4 Experiments

As I was doing the default final project, I was working with the SQuAD 2.0 dataset, which has roughly 150k questions along with corresponding passages. Roughly half of the questions have answers, but the rest do not. Answerable questions have their answer drawn directly from the context passage. For our projects, we are working with a subset of this SQuAD 2.0 data because we do not have access to the official test set data.

The task is for the model to predict the answer to the question given the passage, or predict “N/A" if there is no answer to the given question in the passage. We measure success using the classic Exact Match (EM) and F1 scores as commonly used for SQuAD and other Question Answering datasets.

I first ran the provided baseline, which uses a Bidirectional Attention Flow [8] model without concatenating character CNN embeddings onto the GloVe word embeddings. Using the classic SQuAD metrics of EM and F1 to gauge performance, the baseline had an F1 score of 60.5 and an EM score of 57.39.

I spent many hours going over the QANet [10] and Transformers [9] papers and attempting to create my own implementation in PyTorch, building off the provided baseline codebase, leaning as needed on cited papers for strategies used such as Layer Normalization [1] and Depthwise Separable Convolutions [2]. After fine-tuning layers and making sure matrix shapes were aligned and small examples seemed to be functioning properly, I attempted to run the model on an Azure NV6. I quickly ran into CUDA memory issues. After troubleshooting this and shrinking my batch size, it managed to run, but F1 and EM metrics did not improve much above the low 50s. Frustrated and having already spent more time than I had initially budgeted just to get the base QANet working without Transformer-XL, I decided to use an existing QANet PyTorch implementation, and make changes to that for Transformer-XL. I realized that using this existing codebase would push me into the submission category with PCE, but I was more just interested in seeing how I could perform with Transformer-XL. However, after copying over some of the layers from that codebase and making some minor tweaks to ensure compatibility with the provided framework, I saw improvements in the AvNA metric and in NLL, but EM and F1 did not improve. At that point I realized that I had forgotten to update the learning rate and optimizer to match the description in the QANet paper, so I changed it to an Adam optimizer with warm-up to a 0.001 learning rate over the first 1000 batches. A graph of this learning rate warm-up is shown below.

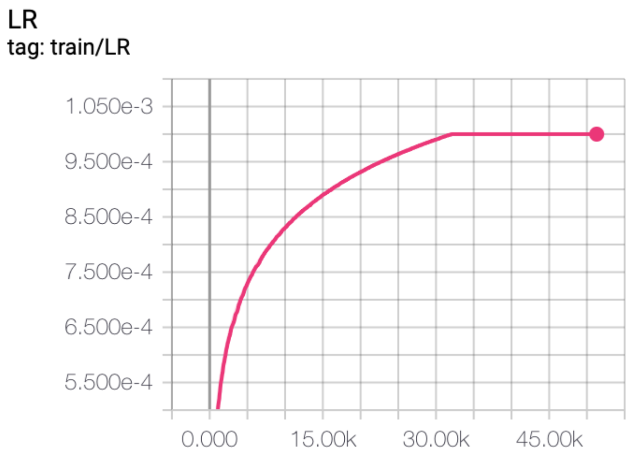


Figure 4: Learning rate warm-up.

I ended up using an Azure NV12 machine with 2 GPUs to train the base QANet architecture. Each epoch took about 40 minutes. Result graphs from dev set validation during the training process are shown at the top of the next page.

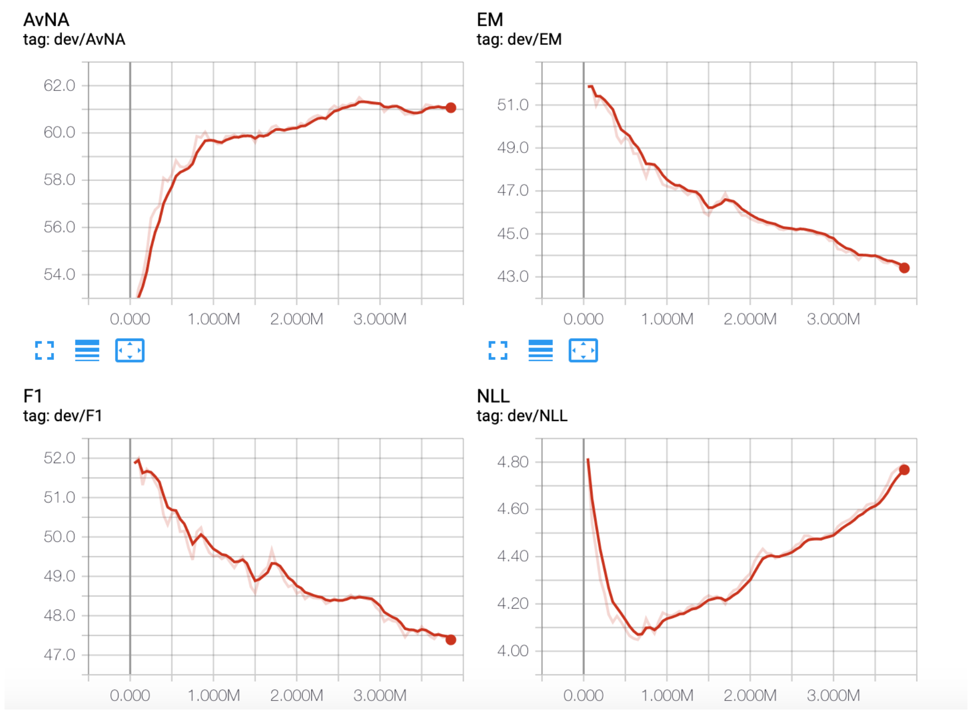
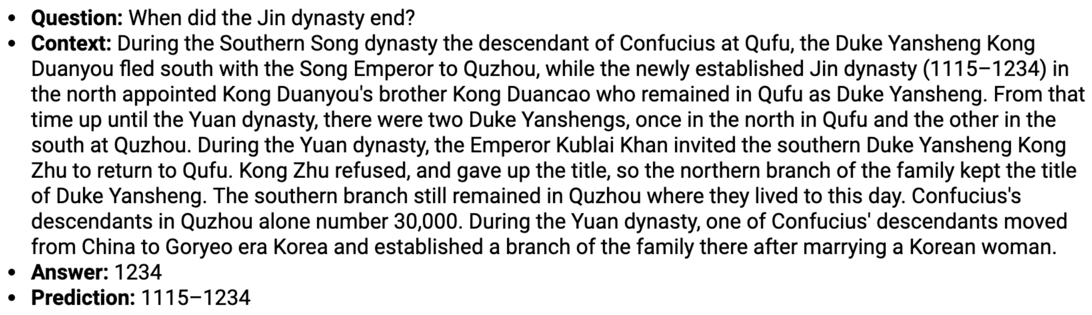


Figure 5: Training metrics for base QANet model.

There was still clearly something wrong. NLL declined sharply, but then began to overfit the training data after around 60000 iterations. While AvNA increased sharply at first and steadily throughout, F1 and EM scores declined steadily throughout training. I found it interesting that the model seemed to be learning which questions had answers, just not getting the correct answers. An example that I found from looking through the selected samples in Tensorboard seemed to indicate that some of the answers were close to being right, but just not quite capturing the correct set of words that the gold-standard source answer had.



- **Question:** When did the Jin dynasty end?
- **Context:** During the Southern Song dynasty the descendant of Confucius at Qufu, the Duke Yansheng Kong Duanyou fled south with the Song Emperor to Quzhou, while the newly established Jin dynasty (1115–1234) in the north appointed Kong Duanyou's brother Kong Duancao who remained in Qufu as Duke Yansheng. From that time up until the Yuan dynasty, there were two Duke Yanshengs, once in the north in Qufu and the other in the south at Quzhou. During the Yuan dynasty, the Emperor Kublai Khan invited the southern Duke Yansheng Kong Zhu to return to Qufu. Kong Zhu refused, and gave up the title, so the northern branch of the family kept the title of Duke Yansheng. The southern branch still remained in Quzhou where they lived to this day. Confucius's descendants in Quzhou alone number 30,000. During the Yuan dynasty, one of Confucius' descendants moved from China to Goryeo era Korea and established a branch of the family there after marrying a Korean woman.
- **Answer:** 1234
- **Prediction:** 1115–1234

Figure 6: Sample dev set result from training base QANet model.

The model correctly identified that we were looking for a year, but incorrectly took the whole range of years instead of just the end of the range. Many other errors mirrored this one, but I could not figure out why this might be the case.

Without a clear idea of what to do to improve this problem, I decided to move on to implementing aspects of Transformer-XL to see if it would improve anything. Because the Transformer-XL paper [3] was a little vague on the implementation of some of its changes, because I had already spent many hours more than I had budgeted for this project on implementing QANet and needed to spare some time for other classes, and because I had already used some external code, I drew on this Github implementation of Transformer-XL to help make some of the changes to represent aspects of Transformer-XL. While performance was similar overall, there was definitely better initial performance before the model started to overfit after about 60000 iterations.

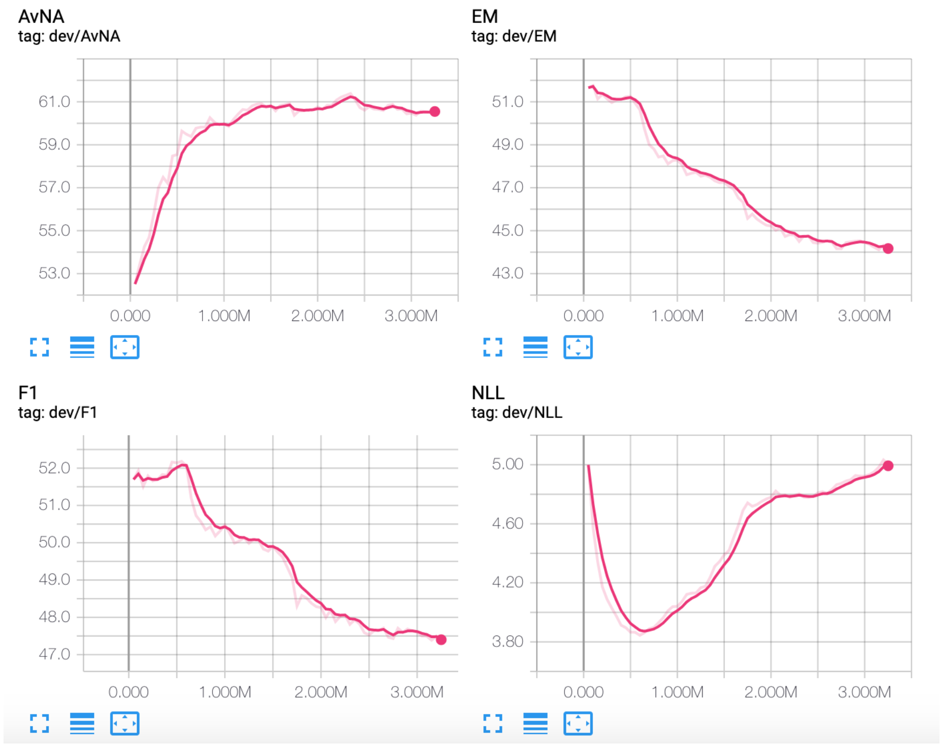


Figure 7: Training metrics for Transformer-XL QANet model.

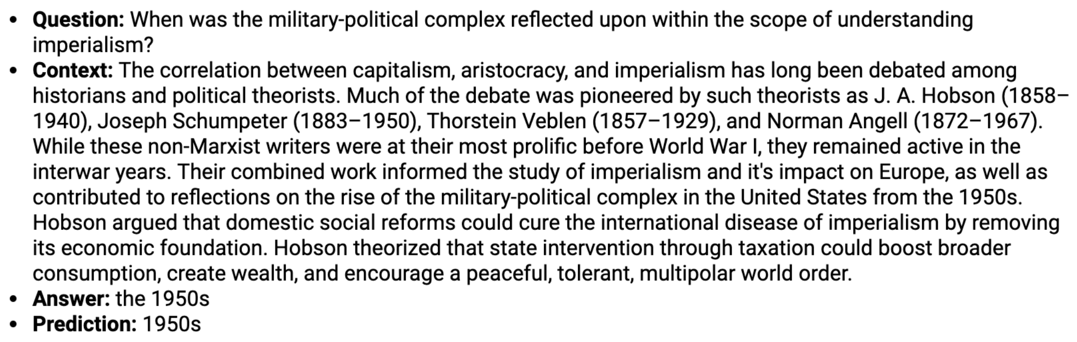
I also ran the default BiDAF code with my own added character embeddings to get a slight improvement over the baseline performance in case we get points for improving on the baseline. I submitted results from the BiDAF with character embeddings model to the non-PCE leaderboard and results from the QANet-XL model to the PCE leaderboard because I used some external code in that. A summary of these results are shown below.

Model	Dev EM	Dev F1
Baseline BiDAF	57.39	60.50
BiDAF with char embeddings	57.50	60.88
Base QANet	51.99	51.87
QANet-XL	52.18	50.93

Although the results of my QANet models were certainly lower than I hoped and expected, they still gave some insight into the possibility for Transformer-XL's usefulness when used in place of vanilla Transformers.

5 Analysis

As I highlighted in the previous section, it seemed like some dev set examples were being answered almost correctly and this inability to match the exact wording was hurting performance scores. That being said, there were certainly still some examples where the model predicted incorrectly, and even mis-identified whether or not there was an answer to the question. From an informal survey of the selected examples in Tensorboard, it appears that, while the AvNA metric improves, we still tend to predict N/A fairly often, even when there is an answer. This bias toward choosing no answer might indicate that, given more time, it would be good to raising the threshold of probability needed to say that there is no answer, or otherwise influence our model to choose an answer. There are a few occasions on which we predict an answer when there is no answer, but overall we would have a much higher performance level if we only looked at the subset of data points without an answer. This fact combined with that already-low EM and F1 scores, however, indicates that the model does rather poorly at correctly predicting an answer, even if we account for cases where just one word is different or missing as in the example below.



- **Question:** When was the military-political complex reflected upon within the scope of understanding imperialism?
- **Context:** The correlation between capitalism, aristocracy, and imperialism has long been debated among historians and political theorists. Much of the debate was pioneered by such theorists as J. A. Hobson (1858–1940), Joseph Schumpeter (1883–1950), Thorstein Veblen (1857–1929), and Norman Angell (1872–1967). While these non-Marxist writers were at their most prolific before World War I, they remained active in the interwar years. Their combined work informed the study of imperialism and it's impact on Europe, as well as contributed to reflections on the rise of the military-political complex in the United States from the 1950s. Hobson argued that domestic social reforms could cure the international disease of imperialism by removing its economic foundation. Hobson theorized that state intervention through taxation could boost broader consumption, create wealth, and encourage a peaceful, tolerant, multipolar world order.
- **Answer:** the 1950s
- **Prediction:** 1950s

Figure 8: Sample dev set result from training Transformer-XL QANet model.

One of the biggest conclusions that we can draw from examining the results of the base QANet model and the Transformer-XL QANet model is that it does seem as if adding features of Transformer-XL to the encoder blocks of QANet helps to improve performance, as we saw a higher F1 score, which was what we were maximizing in training. As claimed in the Transformer-XL paper [3], even when the context length is short enough for vanilla Transformers to capture the whole thing, the recurrent structure of Transformer-XL can allow QANet to pick up on more information from the context and query. This further supports the notion that this could be a good improvement to vanilla Transformers, and it would be interesting to see what other tasks could have their performance improved with these additions. Both models, however, began to overfit to the training data around only 60000 iterations, which could be related to the problem of why performance was so poor, but also could indicate that a shorter training time would be adequate, allowing for more rapid iteration to get improvements.

6 Conclusion

Despite many obvious shortcomings to this project, I was intrigued to find the improvement in performance when I added Transformer-XL features to QANet. While it was a rather small improvement, many architectural changes are commonly implemented to achieve small performance boosts. If I had more time to work full-time on this project and try more techniques for debugging the neural net to determine the cause of the poor performance, it would be interesting to see whether it was still just a small increase with Transformer-XL, or a larger one (or no increase at all). I also only had time to implement some of the aspects of Transformer-XL, so I would love to be able to implement it more fully and see if the increase grew in magnitude. I learned a lot about the intricacies of NLP neural nets, and the power of these models, but also the struggles of debugging them when things don't work out quite as planned. Nonetheless, I am proud of the time that I put into this project and the progress that I made without having a partner to help me out. If Transformer-XL can truly offer an improvement over Transformers, we could see a whole new wave of state-of-the-art models coming out in the near future!

References

[1] Jimmy Lei Ba, Jamie Ryan Kirosa, and Geoffrey E. Hinton. Layer normalization. http://arxiv.org/abs/1607.06450, 2016.

[2] François Chollet. Xception: Deep learning with depthwise separable convolutions. http://arxiv.org/abs/1610.02357, 2016.

[3] 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. http://arxiv.org/abs/1901.02860, 2019.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding, 2018.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory, 1997.

[6] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad, 2018.

[7] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text, 2016.

[8] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. http://arxiv.org/abs/1611.01603, 2016.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. http://arxiv.org/abs/1706.03762, 2017.

[10] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. Qanet: Combining local convolution with global self-attention for reading comprehension. http://arxiv.org/abs/1804.09541, 2018.