Question Answering with Binary Objective

Stanford CS224N Default Project (IID SQuAD Track)

Johnson Jia
Department of Computer Science
Stanford University
johnjia@stanford.edu

Abstract

We show that adding a secondary binary objective of predicting whether a question has an answer or not to the QANet model [1] leads to meaningful improvements on the question answering task SQuAD 2.0 [2].

1 Introduction

The question-answering task is one of the marquee challenges in natural language processing and machine reading comprehension. One well-known public benchmark dataset for the question-answering task is the Stanford Question Answering Dataset (SQuAD) [3]. The specific form of the question-answering task presented in this dataset is given a text passage (also referred to as the *context*) and a question pertaining to the passage (also referred to as the *query*), find a segment of the passage that answers the question. In particular, the answer can be represented by two indices, the first marking the start of the answer segment and the second marking the end. The SQuAD dataset has garnered a lot of attention from the research community and significant progress has been made. Two such noteworthy models that inspired this work are the Bidirectional Attention Flow (BiDAF) [4] and QANet [1]. One key contribution from the BiDAF model is context-query attention layer, which in the words of the authors, "is responsible for linking and fusing information from the context and the query words". Building upon the BiDAF architecture and the groundbreaking findings from [5], QANet replaced all the RNN sequence models with multi-head self attention (and one-dimensional convolutional layers) and significantly improved the convergence rate and model performance. In this paper, we adapt the QANet model to an updated version of SQuAD—which arguably is a more challenging question-answering dataset—SQuAD 2.0 [2].

The main change in SQuAD 2.0 is that now the answers to the questions may not be available in the associated passage. In other words, the model is now expected to not only find the segment of the text that answers the question when there is one, it also needs to determine whether such an answer is present in the text. Empirically, we saw the BiDAF model performed poorer on the SQuAD 2.0 dataset than it did on the SQuAD dataset. The same is true for our implementation of QANet, which was unable to obtained the performance stated in [1] on SQuAD 2.0.[1] Our main contribution is that we show by adding a secondary binary objective of predicting whether the answer is present in the text, we can improve the performance of QANet on SQuAD 2.0. Specifically, using the standard benchmark metrics Exact Match (EM) and F1 score (F1), our multi-task model of QANet with a secondary binary answerability objective (which for simplicity we refer to as **QANet+**) showed an improvement of 1.344 in EM score (from 63.586 to 64.930) and an improvement of 1.094 in the F1 score (from 67.044 to 68.138) over our implementation of QANet on the validation set.[2]

[1] Of course, the comparison is not completely fair and rigorous given we used our own implementation of QANet and the training and evaluation datasets are different, and we didn't have data augmentation, but still the gap in performance is large enough that our intuition suggests the drop in performance is not due to implementation and training differences alone, but rather that the models themselves are less performant on SQuAD 2.0.

[2] Due to the limitation on the number of submissions of the predictions on the test set, we did not evaluate our vanilla QANet on the test set so we don't have the same comparison on the test set.

2 Related Work

There have been lots of interesting and fruitful work in the field of question-answering. For example, there is the Dynamic Coattention Network [6] which adds another attention layer to the context-query attention used in the BiDAF model. Another model sharing a similar theme is the R-Net [7], which adds a self-attention layer besides the context-query attention layer.

In a different direction, [8] made the prediction of the end word of the answer segment directly dependent on the prediction of the start word through a Pointer Network [9]. Another model that focuses on the output prediction is [10], which does not predict the start and end words of the answer segment but instead directly predicts the probability of every possible span being the answer.

More recently, thanks to the groundbreaking work in [11], research on question-answering has focused on finetuning a pretrained self-supervised transformer-based model like BERT. This approach has been incredibly successful, giving us models with colorful names such as RoBERTa [12], ALBERT [13], XLNet [14], T5 [15], and many more, some of which have achieved super-human performance on the SQuAD 2.0 leaderboard. One downside of these models is that they tend to be very large and may not be able to fit on a single GPU, which will make finetuning less accessible. To rectify this, [16] introduced the Reformer, which replaces the dot product in the self-attention with a locality sensitive hashing scheme to reduce the computation cost. It also leverages Reversible Residual Network [17] to allow for processing one encoder-decoder layer in GPU at a time.[3]

3 Approach

Our idea is a simple but an effective one: Add a secondary binary objective of predicting whether the answer is present in the associated passage or not. Our best performing model is a multi-task QANet, which we refer to as QANet+, that does both the start and end word prediction and this secondary binary answerability prediction. We describe the model architecture in detail below. To be consistent with research literature, we will refer to the text passage as the context and the question and the query.

3.1 Input Embedding Layer

We use both word embedding and character-level embedding as in BiDAF [4] and QANet [1]. For word embedding, we use pretrained GloVe word vectors [18]. For character embedding, we first use the 64-dimensional character vectors provided with the project to embed the characters of each word. We then apply a one-dimensional convolutional layer—with kernel size 3 and 64 output channels—over the character vectors in each word. Finally following BiDAF we take max pooling over each word to obtain 64-dimensional character-level embeddings of the words.

We the pass the concatenation of the GloVe and character-level word embeddings through a dropout layer and then a two-layer Highway Network [19]. The resulting word vectors are 128-dimensional, same as that of the word embedding used in QANet. The same input embedding layer is used to encode both the context and query words.

3.2 Embedding Encoder Layer

The embedding encoder layer encodes the interactions between the words within the context and query. For this, we use a combination of depth-wise separable convolution and multi-head self-attention as in QANet. The convolution models local interactions whereas the multi-head self-attention models long-term dependencies between words that are much further apart.

Let us describe the encoder architecture in detail. As shown in Figure 1, it is made up of four blocks. The first is a sinusoidal positional encoding as in [5]. This is to capture the positional information. Then comes multiple depth-wise separable convolutional layers, which are more memory efficient and generalize better than conventional convolutional networks according to [1]. This is then followed up by the third component block, multi-head self-attention, which captures the long-term dependencies in the sequence. This is finally followed up by two feed-forward linear layers with ReLU activation

[3] Initially we planned to use the Reformer architecture to train a large transformer-based model, unfortunately our transformer-based models performed very poorly and we decided to abandon this idea. More details are provided in Appendix B.

in between. More over, each such component is wrapped inside a residual-layer norm block as shown in Figure 2. The residual connection provides a shortcut for the identity to flow through, which is known to improve learning and helps to prevent neurons from dying; the layer norm serves both as a regularization and also helps to smooth the gradients and speed up training [20]. Their usage is standard in transformers.

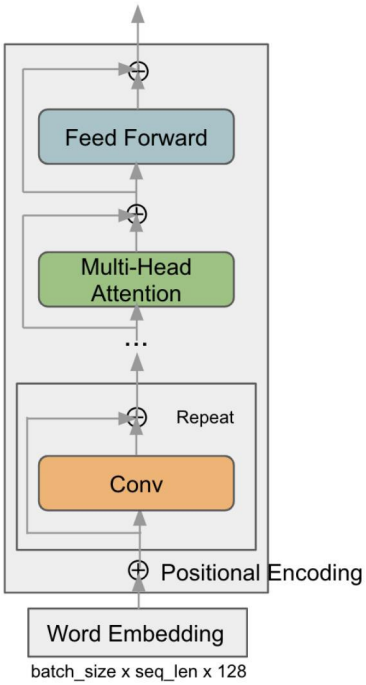


Figure 1: Encoder Architecture

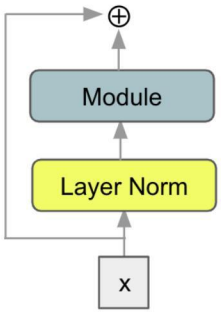


Figure 2: Residual-Layer Norm Connection

For the embedding encoder, we used 4 layers of convolution, each with kernel size 7 and channel size 128. Each convolution layer is followed by a ReLU activation. More importantly, we use stochastic depth (layer drop out) on the convolutional layers as in [1] for regularization and to speed up training. Specifically, during training, we multiply the lth convolutional layer by a Bernoulli variable

$$b \sim \text{Bernoulli}(1 - \frac{l}{L}(1-p))$$

where L is total number of layers—4 in this case—and p is a parameter set to 0.9. For the multi-head self-attention layer, we use 8 heads. The feed-forward linear layers both output 128-dimensional activations. We note that the same embedding encoder is used to encode both the context and query words.

3.3 Context-Query Attention Layer

The context-query attention layer involves computing context-to-query (C2Q) attention and query-to-context (Q2C) attention. To explain this, let $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_N \in \mathbb{R}^{128}$ be the encoded context word vectors, where N is the length of the context; and let $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_M \in \mathbb{R}^{128}$ be the encoded query word vectors, M being the length of the query. Following [1], we first compute a similarity matrix $\mathbf{S} \in \mathbb{R}^{N \times M}$ using the formula

$$S_{ij} = \mathbf{w}_{\text{sim}}^{\top}[\mathbf{c}_i; \mathbf{q}_j; \mathbf{c}_i \circ \mathbf{q}_j]$$

where $\mathbf{w}_{\text{sim}} \in \mathbb{R}^{3 \times 128}$ is a weight vector and $\circ$ denotes element-wise multiplication.[4] As the name entails, S_{ij} is a measure of the similarity between $\mathbf{c}_i$ and $\mathbf{q}_j$. To compute C2Q attention, we take softmax over the rows of $\mathbf{S} = [S_{ij}]$, attending each $\mathbf{c}_i$ to all the $\mathbf{q}_j$'s:

$$\bar{\mathbf{S}}_{i,:} = \text{softmax}(\mathbf{S}_{i,:}) \in \mathbb{R}^M.$$

[4]Note we are using horizontal/row vectors.

Let $\bar{\mathbf{S}} = [\bar{\mathbf{S}}_{i,:}] \in \mathbb{R}^{N \times M}$, the C2Q attention is then computed as

$$\mathbf{A} = \bar{\mathbf{S}}\mathbf{Q} \in \mathbb{R}^{N \times 128}$$

where $\mathbf{Q} = [\mathbf{q}_1^\top; \cdots; \mathbf{q}_M^\top]^\top \in \mathbb{R}^{M \times 128}$ is the matrix whose jth row is $\mathbf{q}_j$.

For the Q2C attention, we first take softmax over the columns of $\mathbf{S}$, which gives us another matrix $\bar{\bar{\mathbf{S}}}$ with

$$\bar{\bar{\mathbf{S}}}_{:,j} = \mathrm{softmax}(\mathbf{S}_{:,j}) \in \mathbb{R}^N.$$

We then compute the Q2C attention as

$$\mathbf{B} = \bar{\mathbf{S}}\bar{\bar{\mathbf{S}}}^\top \mathbf{C} \in \mathbb{R}^{N \times 128}$$

where $\mathbf{C} = [\mathbf{c}_1^\top; \cdots; \mathbf{c}_N^\top]^\top \in \mathbb{R}^{N \times 128}$ is the matrix whose ith row is $\mathbf{c}_i$.

In [1], the final output of the context-query layer is

$$\mathbf{g}_i = [\mathbf{c}_i; \mathbf{A}_{i,:}; \mathbf{c}_i \circ \mathbf{A}_{i,:}; \mathbf{c}_i \circ \mathbf{B}_{i,:}] \in \mathbb{R}^{4 \times 128},$$

where $\mathbf{A}_{i,:}$ and $\mathbf{B}_{i,:}$ are the ith row of C2Q and Q2C attention respectively. In our model, to reduce GPU memory usage, we apply a linear projection to map $\mathbf{g}_i$ down to 128 dimensions.

3.4 Model Encoder Layer

The model encoder layer is made up of three passes of a stacked encoder block. Specifically, we create a encoder block made up of 7 encoders, each of which has the same architecture as in Figure 1. Each of these encoders in the encoder block is composed of 2 layers of convolutions with kernel size 5 and 8-headed self-attention. These are the same choices of hyperparameter as in [1]. We also apply stochastic depth on the convolutional layers as we did for the embedding encoder.

The output of the model encoder layer are three tensors $\mathbf{m}_0, \mathbf{m}_1, \mathbf{m}_2 \in \mathbb{R}^{N \times 128}$. We emphasize that the same encoder block is used to produce each of these tensors; in other words, the encoder block's weights are tied.

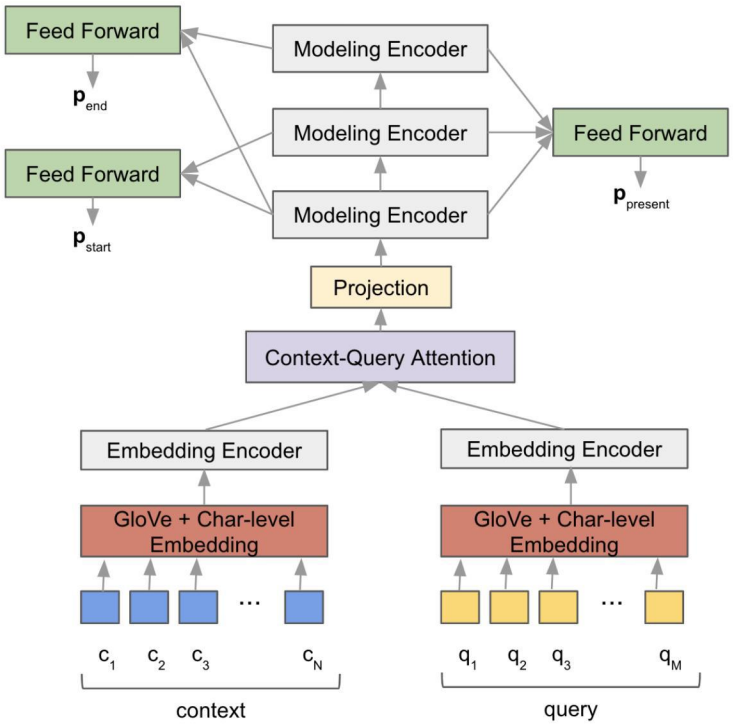


Figure 3: QANet+ Model Architecture

3.5 Output Layer

The output layer consists of two types of outputs: Probabilities of the start ($\mathbf{p}_{\text{start}}$) and end ($\mathbf{p}_{\text{end}}$) words of the answer segment, and the probability of whether the answer is present in the context or not (p_{present}). To predict the start word, we concatenate the output of the first and second pass of the modeling layer, $[\mathbf{m}_0; \mathbf{m}_1]$, and pass it through a single linear feed-forward layer followed by softmax. Predicting the end word is exactly same except we concatenate $\mathbf{m}_0$ and $\mathbf{m}_2$, the output of the third pass of the modeling layer, as the input to a feed-forward layer with softmax activation.

For the binary answerability prediction, we concatenate the 0th word of $\mathbf{m}_0$, $\mathbf{m}_1$, and $\mathbf{m}_2$ and pass $[\mathbf{m}_0(0); \mathbf{m}_1(0); \mathbf{m}_2(0)]$ through a feed-forward layer with sigmoid activation. The reason for selecting the 0the word is that it is a OOV (Out of Vocabulary) token that represents answer not present.

3.6 Loss Function

We use a multi-task negative log likelihood loss function to train the model. Specifically, given a context-query-answer triple (c, q, a), we denote by a_i the starting word position and a_j the ending word position of the answer, then $e = (a_j \neq 0)$ is a boolean value that's 1 if the answer is present and 0 otherwise. With these notation, the loss function is

$$NLL = \sum_{(c,q,a)} \underbrace{-\log(\mathbf{p}_{\text{start}}(a_i)) - \log(\mathbf{p}_{\text{end}}(a_j))}_{\text{start-end prediction loss}} \underbrace{- \alpha \left[e \log(p_{\text{present}}) + (1 - e) \log(1 - p_{\text{present}}) \right]}_{\text{answerability prediction loss}}$$

where α is a configurable hyperparameter.

4 Experiments

4.1 Data

The SQuAD 2.0 dataset we used is split into a training set of 129,941 (context, query, answer) triples, a dev set of 6,078 examples, and a test set of 5,915 examples. We note that the dev and test set are split out from the official dev set with some examples from the actual test set added in.

4.2 Evaluation method

We use the standard evaluation metrics Exact Match (EM) and F1 score (F1) for SQuAD 2.0. One a single example, EM is 1 if the prediction matches the groundtruth word for word and is 0 otherwise; on the other hand, F1 score is the harmonic mean of the precision (ratio of the number of correctly predicted words over the total number of words predicted) and recall (ratio of the number of correctly predicted words over the total number of words in the groundtruth). For SQuAD 2.0, there are three human labeled answers (when the answer is present in the context), and in this case we take the maximum of the EM and F1 of the prediction. Finally we take an average of the EM and F1 over all examples in the dev and test set to get the final EM and F1 score.

4.3 Experimental details

For training our QANet+, we used a dropout probability of 0.1 across all layers where dropout was used. The optimizer we used is ADAM [21] with $\beta_1 = 0.8$, $\beta_2 = 0.999$, and $\epsilon = 10^{-7}$. Following [1], we used a learning rate scheduler that increase the learning rate from 0 to 0.001 in the first 1000 steps using the function $(1 - e^{1-s}) \times 0.001$, and then maintain a constant learning rate of 0.001 onwards. We also added a L2 weight decay factor of 3×10^{-7}. We did not truncate gradient norms. The weight on the answerability binary objective is 0.1.

The results shown below are from models trained on a single Nvidia GTX 1080 Ti GPU with 11GB of VRAM. For QANet+, we used a batch size of 18 in order to make it fit in the GPU memory. Each epoch took around 30 minutes of training time. The results shown in Table 1 are the best checkpoint model on the dev set after training for 30 epochs.

We also performed some manual hyperparameter search with the dropout probability (0.15 instead of 0.1) and the loss weight α on the binary answerability objective (0.75 and 0.125 instead of 0.1). In

Model	EM	F1
BiDAF	57.284	60.912
BiDAF w char-level encoding	60.108	63.723
QANet	62.074	65.460
QANet dropout probability 0.15	62.074	65.441
QANet stochastic depth and tied embedding encoder weights	63.586	67.044
QANet+	**64.930**	**68.138**

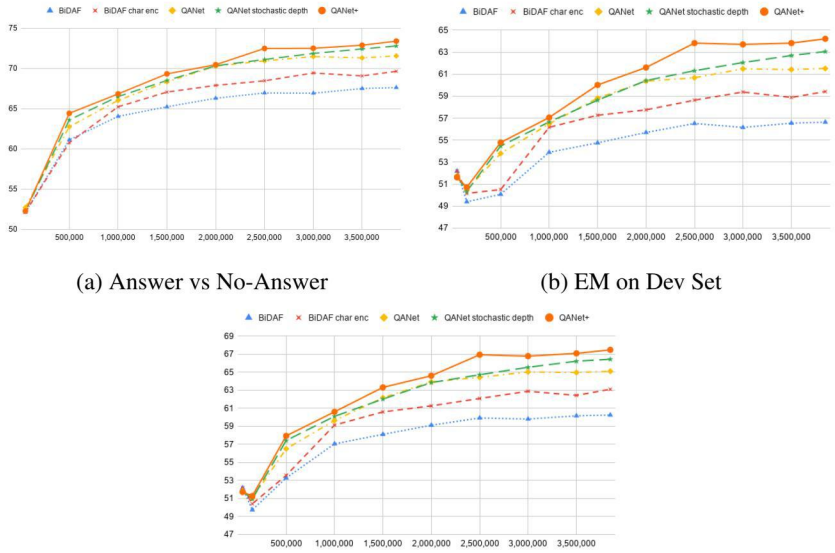
Table 1: Comparison of Model Performance on the Dev Set

both cases the model performed worse. We also tried joint training the word embeddings instead of use the fixed GloVe embeddings, but again the performance deteriorated.

4.4 Results

As shown in Table 1, QANet+ has the best performance, beating our implementation of QANet by 1.344 on EM and 1.094 on F1 on the dev set. The improvement is even more significant when compared against the baseline BiDAF model, with a gain of 7.646 in EM and 7.226 in F1.

For the test leaderboard (non-PCE), we trained QANet+ for 50 epochs using the same configuration. This model obtained an EM score of 61.657 and a F1 score of 65.265.



(a) Answer vs No-Answer

(b) EM on Dev Set

(c) F1 on Dev Set

Figure 4: Training Progress Comparisons

The model performance improvements of QANet+ over QANet and BiDAF is clearly observed during training as well. We see in Figure 4 that QANet+ consistently outperforms QANet and BiDAF (and their variants) during training on the EM and F1 metrics, as well as the classification accuracy of answer or no answer (AvNA).

It is also worth noting that our experiments were able to reproduce the metric gains from known improvements over the baseline BiDAF provided for the project. Adding character-level encoding to the baseline resulted in around 3 points gain in EM and F1. Replacing BiDAF with QANet (a variant with no stochastic depth and using separate embedding encoder for context and query) resulted in another gain of around 2 points. Enabling stochastic depth and using the same embedding encoder for context and query resulted in yet another 1.5 points of improvement. Finally our QANet+ provided another point of improvement.

5 Analysis

We note that the probabilities of the start and end words can already capture the presence of an answer in the context. In particular, the first word of the context is an OOV (Out of Vocabulary) token that can be used to indicate the lack of an answer in the context. So if $\mathbf{p}_{start}(0) \cdot \mathbf{p}_{end}(0)$ is larger than any predicted answer span, then the prediction is no answer. This however, is less direct than predicting the presence of an answer. Given roughly half of the questions in the training dataset cannot be answered from the context provided, it goes to reason that adding a secondary binary answerability objective will improve the model performance. Our work shows this indeed the case. We believe this improvement has to do with the inductive bias the model learned from this binary answerability prediction. Moreover, as the model gets better at predicting answerability, the start and end probabilities it assigns to the no-answer token at the 0th position are also more accurate, and this indirectly helps the prediction of the start and end words as well.

6 Conclusion

Multitask learning has proven to be effective in improving model performance in machine reading comprehension (e.g. both BERT [11] and GPT-2 [22] area multitask models and arguably this helped their performance). Our model, QANet+, builds on QANet [1] by adding a secondary binary objective of predicting answerability. It achieved meaningful improvements over QANet in both EM and F1 metrics on the SQuAD 2.0 dataset. We believe there are still other tasks (such as predicting the length of the answer) that can help to further improve the performance of QANet and this will be work for the future.

References

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

[2] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018.

[3] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016.

[4] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[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] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *CoRR*, abs/1611.01604, 2016.

[7] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 189–198, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[8] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905, 2016.

[9] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[10] Yang Yu, Wei Zhang, Kazi Saidul Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-end reading comprehension with dynamic answer chunk ranking. *CoRR*, abs/1610.09996, 2016.

[11] 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.

[12] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[13] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942, 2019.

[14] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019.

[15] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[16] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer, 2020.

[17] Aidan N. Gomez, Mengye Ren, Raquel Urtasun, and Roger B. Grosse. The reversible residual network: Backpropagation without storing activations. *CoRR*, abs/1707.04585, 2017.

[18] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[19] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

[20] Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization. *CoRR*, abs/1911.07013, 2019.

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[22] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

A Tensorboard Plots

Figure 5: Legend

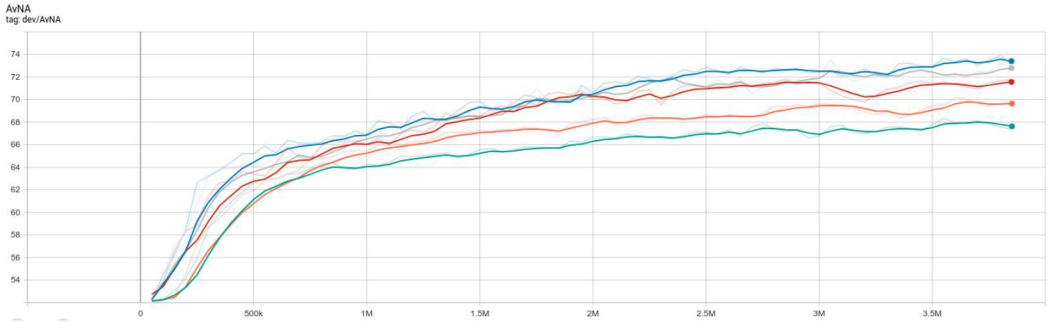


Figure 6: Dev Answer vs No-Answer

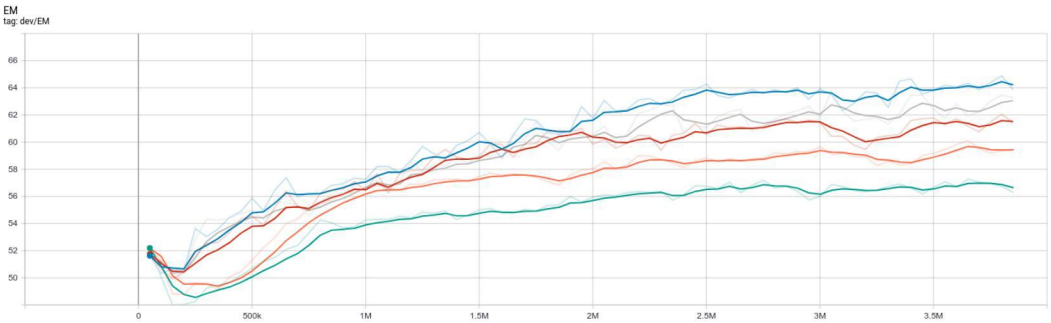


Figure 7: Dev EM

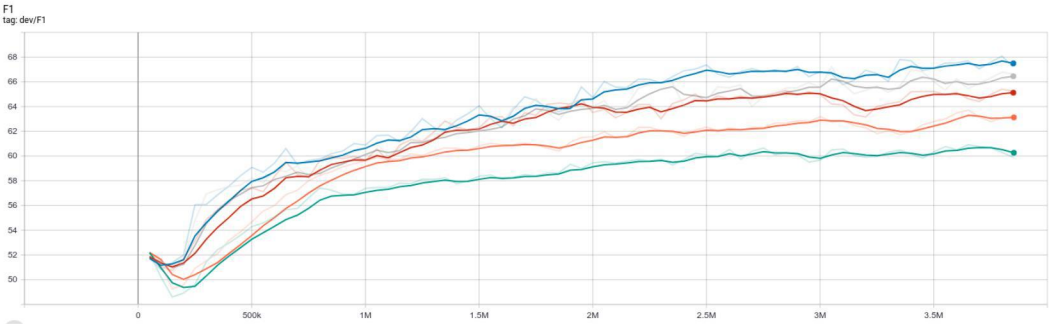


Figure 8: Dev F1

B Unsuccessful Experiments

My initial plan was to train a transformer-like model. In particular, I wanted to use a stacked context encoder layer to encode the context and pass its output to a stacked query encoder + context-query attention layer to predict positions of the start and end words. One such design involved using 6 layers of the transformer encoder from [5] to encode the context and query, and compute context-query attentions with the encoded tensors at each layer. Then I added a few linear layers to predict the start and end position from the 6 sets of context-query attentions. Unfortunately this model didn't work. I tried a few variations but was unsuccessful.