QANet and Performance on SQuAD v2.0

Stanford CS224N Default Project

Alex Fan, Yixuan Wu
Department of Statistics
Stanford University
`alexfan@stanford.edu` `sherryyw@stanford.edu`

TA Mentor: Sarthak Kanodia

Abstract

Question answering, and more generally machine reading comprehension, has always been a staple in NLP benchmark testing because of its representation of machine's ability to understand linguistic elements. Modern question answering systems, ranging from the RNN-based systems to the large-scale transformer-based models, have become remarkably complex within the last few years. The goal of this project is to build upon the BiDAF method [1] by adding character-embeddings, and build from scratch the transformer-based QANet model adapted from Yu et al. [2] Additionally, we tuned the three parameters, learning rate, dropout probability, and number of stacked encoder blocks in relationship with batch size. We achieved an F1 score of 69.11 and an EM score of 65.85 on the hidden test set.

1 Introduction

Reading comprehension is a rapidly developing and exciting field in natural language processing. Early research focused mostly on cloze-style datasets, in which a model aims to fill in missing words in text. However, with the publication of Stanford's SQuAD dataset and advancement of modelling techniques, question-answering (QA) systems are gradually equipped with logical reasoning and content comprehension skills to answer questions posed by humans in natural languages. With such ability, QA systems are seen to improve search engines functions, answering general user questions in dialogue systems, extracting key information from long passages, etc.

QA systems fulfill these tasks by extracting the right span of words to answer a given question when provided a context paragraph. There are several challenges posed by this NLP task. First, the models may fail to find answers that are far away from the question, especially when phrases similar to that of the answer are much closer. Secondly, it can be extremely challenging for the model to correctly predict that some questions are unanswerable. This requires both reading comprehension of the paragraph and critical judgement that none of the facts answer the question. Otherwise, providing wrong and especially seemingly right answers can be extremely misleading to the readers who don't know the ground truths.

Currently, most of the best performing models, according to SQuAD 2.0's leaderboard, rely on pre-trained contextual embeddings. Examples includes "SA-Net on ALBERT", "ALBERT + DAAF + Verifier", and "Retro-Reader on ALBERT", which are all currently ranked top 10 on the leaderboard [3]. While pre-trained language models provide promising results, the process of pre-training is generally extremely time- and resource-demanding. Therefore, we would like to focus on non-pretrained language models, which would allow us dive into the basics of model-buildings and thus better understand model architecture.

Out of the non-pretrained models, the transformer-based QANet which was originally introduced by Yu et al. in 2018 exhibits great performance on the previous version of SQuAD. Therefore, with the addition of unanswerable questions in the SQuAD v2.0, we are interested in examining if QANet can more effectively handle unanswerable questions.

For this project, we were given the baseline Bi-Directional Attention Flow(BiDAF) model adapted from Seo et al. [1] Thus, our work includes:

- Implementing the character-level embeddings on the baseline BiDAF model.
- Adapting the QANet model from scratch.
- Exploring variations of the model, which includes fine-tuning hyperparameters and ensembling the model to improve performance.
- Performing both qualitative and quantitative analyses of our final model.

2 Related Work

Our work is based on the BiDAF model introduced by Seo et al. [1]. In short, this model relies on a bi-directional attention that ensures the attention to flow both from context to question and from question to context. The output of this attention layer is then fed into a LSTM-based encoder layer, of which the output probabilities are calculated based on.

Instead of using the LSTM-based encoder layer, QANet, on the other hand, uses stacked encoder blocks that are constructed with self-attention and convolution layers. This feed-forward architecture allows for parallel computations and thus speeds up model training. Yet such parallel structures demands much higher GPU memory, which is an issue we faced throughout our model training.

We also closely studied hierarchical attention with fusion networks presented by Wang et al. (2018)[4] and adapt them into the existing QANet architecture. However, due to the time constraint and structural difference between the two techniques, we were not able to create a viable model that could compete with QANet.

3 Approach

We approach the question answering task through two distinct architectures. First, following Seo et al. (2017), we implement character-level embeddings on the baseline BiDAF network given to us[1]. Second, adapting from Yu et al. (2018) and their QANet model, we implement a transformer-based model that improves upon the original BiDAF model.

3.1 BiDAF

Baseline Model For the sake of conciseness, we refer the reader to the description of the baseline BiDAF model[5] (without character embeddings) for details on the encoding and output layers.

Embedding Layer for Full BiDAF Context/query words and characters are first fed into embedding layer. For the words, we leverage pre-trained 300-dimensional GLoVe vectors[6]. The learn-able character embeddings, which are randomly initialized, are additionally fed through a convolution network and max pooled across the word length such that there is one embedding for a given word. These embeddings are then concatenated, projected back down, and fed into a two-layer Highway Network[7], resulting in each word of the sequence being in $\mathbb{R}^h$.

3.2 QANet

The QANet architecture, proposed by Yu et al. (2018), is almost a one-to-one match with the original BiDAF model in terms of structure. In fact, both the embeddings layer and the context-query attention layer remain exactly the same, and the main difference lies in the removal of LSTMs in favor of convolution-based transformer blocks[2]

QANet Encoder Block The encoder block is a modular component that can be stacked on top of itself. It contains three main components: the (repeated) convolution network, the multi-headed self-attention, and the feed forward network(s). Given an input matrix $\mathbf{X}$, we first add a sinusoidal positional encoding[1] to the matrix in order to capture relative positions of tokens [8]. The matrix is

[1]Our implementation of the positional encoding borrows from the open-source pytorch example with some minor modifications: https://github.com/pytorch/examples/blob/main/word_language_model/model.py

then passed into the convolution network, which is implemented as a depthwise-separable convolution network with additional ReLU and dropout layers at the end. Yu et al. (2018) notes that the depthwise-separable convolution has empirically shown to be more memory efficient and have better generalization in capturing local contexts. The output of the convolution network is then fed through a multi-headed self-attention network[2], which is a staple of transformer-based models [8]. The feed-forward network are implemented as two linear layers with a ReLU for nonlinearity between the two. For each of these 3 components, separate layer norms and residual layers are added at the beginning and end respectively.

Contextual Embedding Block Layer For the contextual embedding encoder, there is only one block with the convolution network repeated four times. It takes as input dimension $d = 128$, which is the hidden state parameter of the model, from the output of the character/word embedding layer, and it outputs a matrix in dimension $d = 128$.

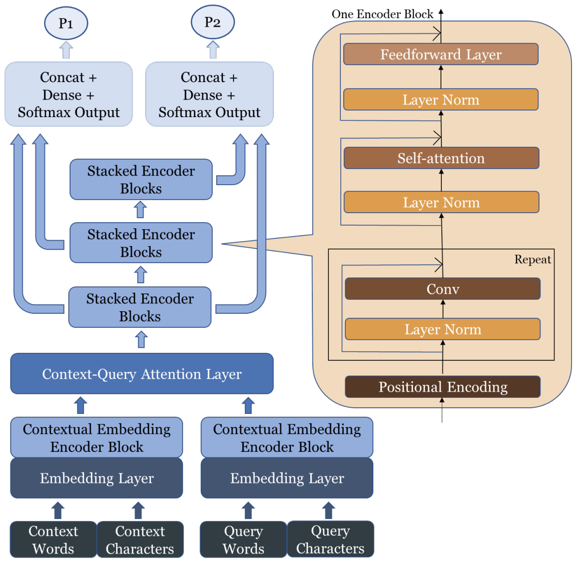


Figure 1: QANet Diagram

Context-Query Attention Layer The outputs of the contextual embedding layer are next used in the context-query attention layer, which comes from Seo et al. (2017)'s BiDAF. This layer begins by calculating a similarity matrix between the context and the query given by the following equation:

$$\mathbf{S}_{tj} = \alpha(\mathbf{C}_{:t}; \mathbf{Q}_{:t}) \in \mathbb{R}$$

where α is a trainable function defined as $\alpha(\mathbf{c}, \mathbf{q}) = \mathbf{w}_{\mathbf{S}}^{\top}[\mathbf{h}; \mathbf{u}; \mathbf{h} \circ \mathbf{u}]$. This similarity matrix is used to calculate two sets of attention: context-to-query (C2Q) and query-to-context (Q2C). C2Q's attention weights are calculated as $\mathbf{a}_t = \mathtt{softmax}(\mathbf{S}_{t:}) \in \mathbb{R}^J$, and the resulting attended query vector is $\tilde{\mathbf{Q}}_{:t} = \sum_j \mathbf{a}_{tj}\mathbf{Q}_{:j}$. In similar fashion, Q2C's attention weights on the context words are calculated as $\mathbf{b}_t = \mathtt{softmax}(\mathtt{max}_{col}(\mathbf{S}_{t:}))$ and the resulting attended context vector is $\tilde{\mathbf{C}}_{:t} = \sum_j \mathbf{b}_{tj}\mathbf{C}_{:j}$. Finally, the output of the context-query attention layer is the concatenated matrix $\mathbf{G} = [\mathbf{c}; \tilde{\mathbf{q}}; \mathbf{c} \circ \tilde{\mathbf{q}}; \mathbf{c} \circ \tilde{\mathbf{c}}] \in \mathbb{R}^{4d}$

Stacked Modeling Encoder Blocks For the stacked encoder blocks, we use a total of seven blocks, each with the inner convolution network repeated twice. We pass a resized version[3] of the context-query attention, through the stacked encoder blocks three times (sharing weights each time). This results in 3 output matrices: $\mathbf{M}^1, \mathbf{M}^2, \mathbf{M}^3$, which are fed into the output layer.

QANet Output Layer Inside the output layer, the start and end probabilities is calculated with the following equations:

$$\mathbf{p}^1 = \mathtt{softmax}(\mathbf{w}_1[\mathbf{M}^1; \mathbf{M}^2]); \quad \mathbf{p}^2 = \mathtt{softmax}(\mathbf{w}_2[\mathbf{M}^1; \mathbf{M}^3])$$

where $\mathbf{w}_1$ and $\mathbf{w}_2$ are a linear layers. The softmax is also a masked softmax similar to the output layer from BiDAF.

3.3 QANet Ensembling

After training multiple QANet runs, we ensemble the three best runs into a final model to test. To ensemble the models, we let each model predict their own start and end probabilities and then average the results to form a final start and end probabilities vector. This is then discretized and evaluated in the standard way.

[2]Our implementation borrows from the minGPT/Assignment 5 version, with modifications to make sure the masking is correct: https://github.com/karpathy/minGPT/blob/master/mingpt/model.py

[3]This resizing was primarily done due to memory constraints

4 Experiments

4.1 Data

We utilize the standard question-answering dataset SQuAD v2.0, which contains 141,934 examples. 129,941 of those examples are used for training while half of the official dev set (around 6,000 examples) will be used for tuning and cross validation. The other half of the dev set will be used as the final test set. SQuAD v2.0 adds onto the original dataset by including over 50,000 unanswerable questions, forcing the machine to also understand when the answer might not be within the context paragraph [9].

4.2 Evaluation method

For quantitative evaluation, we look to F1 and Exact Match (EM) scores, which are the standard evaluation metrics for the question-answering task. F1 is defined as the harmonic mean of precision and recall of the predicted answer span; in other words, it represents the portion of overlapping tokens. EM is a binary measure, requiring an exact match of predicted tokens to ground truth. The final score for an individual example is the maximum EM/F1 score across all three human-provided target answer, and the model's overall performance is then the average of all maximum EM/F1 scores. This evaluation metric is used to compare performance across multiple model architectures and specifications.

4.3 Experimental details

BiDAF In training for both the baseline and BiDAF with character-level embeddings, we used a batch size of 64, a learning rate of 0.5, and a hidden size of 100. For regularization, we used an exponential moving average (EMA) decay of 0.999 and a dropout probability of 0.2. The run was trained for 30 epochs.

QANet For all runs of QANet, we maintained a hidden size of 128 to fit with an 8-headed self-attention network within the transformer. For regularization, we kept constant an l2-weight decay of 3×10^{-7} and an EMA decay of 0.999. Dropout layers, which we tuned specifically, are also utilized after each layer. Each run was trained for 30 epochs.

Table 1: Hyperparameter Tuning for QANet

Tuned Parameter	Value	EM	F1
Learning Rate	0.0005	67.29	64.02
	0.001	69.17	66.82
	0.005	52.19	52.19
Dropout Prob	0.05	69.62	65.89
	0.1	70.17	66.81
	0.2	69.17	65.82
# of Blocks and Batch Size	# = 7; B = 30	70.17	66.81
	# = 5; B = 40	68.39	64.86
	# = 4; B = 48	70.17	66.83

For hyperparameter tuning (Table 1), we first tested for the best learning rate, which was found to be 0.001. Using this learning rate, we tested various levels of dropout probabilities for regularization purposes, and the best dropout probability was found to be 0.1.

Finally, using the selected learning rate and dropout probability, we examined the relationship between number of encoder blocks in one stacked encoder blocks (we have three stacked encoder blocks in total, see Figure1) and batch size. Due to the limit of CPU memory, when having 7 encoder blocks, the maximum batch size possible is 30. With an hypothesis that an increase in batch size may lead to better performance, we decreased the encoder blocks while maximizing batch size under the memory constraint. We found that 7 encoder blocks tended to give more consistent and better results with the tradeoff being longer training time; however 4 encoder blocks also seem to give reasonably comparable results to the larger encoder block stacks while drastically cutting the training time. While this sequential tuning is not ideal, in the future we can test a more expansive grid search of these hyperparameters.

These experiments were all run on a NVIDIA V100 GPU unit.

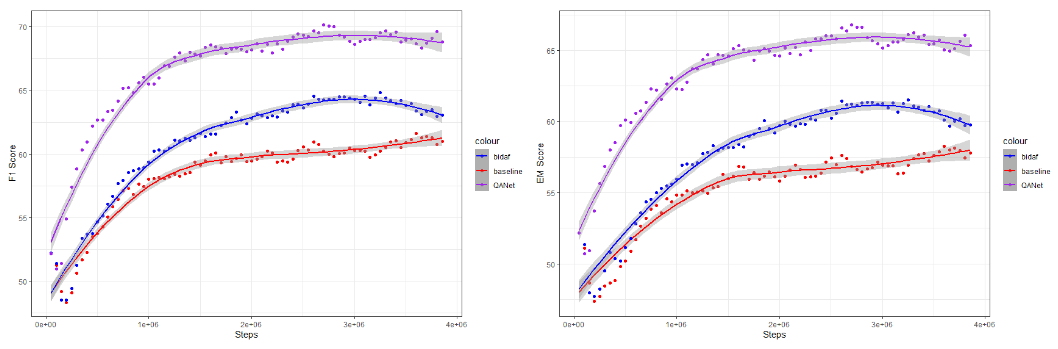
4.4 Results

The following table shows the results for each best performing model with their respective F1 and EM scores on the development set. In addition, the ensemble method, which performed best on the development set, is evaluated on the final test set, and its metrics are also reported.

In terms of performance, QANet performs far better than both the baseline and the BiDAF with character-level embeddings, with a 17.7%/18.7% and 10.8%/11.9% increase in F1/EM for the respective models. This suggests that the high-powered nature of the transformer blocks are able to generate better contextual representations and understandings of the underlying linguistic elements compared to LSTMs. Interestingly, while the original paper suggested having 7 encoder blocks in each of the three stacked encoder blocks (details in Figure1), the 4 encoder block version has results that nearly matches the more complex model, and its performance on the test set is marginally better.

Table 2: Model F1 and EM Scores on SQuAD v2.0

Model	F1	EM
Baseline BiDAF	61.0	58.0
Full BiDAF	64.83	61.54
QANet Single Model (7Blks, 30B)	70.17	66.81
QANet Ensemble (7 Blks, 30B)	71.59	68.36
QANet Single Model (4 Blks, 48B)	70.17	66.83
QANet Ensemble (4 Blks, 48B)	71.85	68.86
QANet Ensemble (7 Blks; Test)	68.33	65.21
QANet Ensemble (4 Blks; Test)	**69.11**	**65.85**



(a) Model F1 Scores

(b) Model EM Scores

Figure 2: Model Comparisons

That being said, our methodology does have its disadvantages. First, the ensembled QANet model seems to be prone to overfitting. When running on the test set, the ensembled models exhibit large drops in performance. This could be due to a combination of the original model complexity of QANet's transformer architecture and the ensembling methodology of averaging the highest performing F1 scores based on the development set. Overall, these results are as expected, and this suggests more future work in improving QANet's structure.

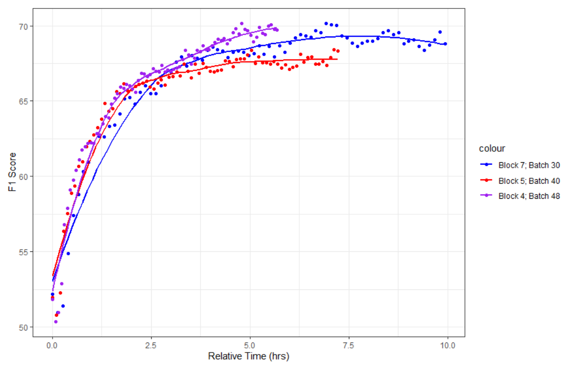


Figure 3: Number of Encoder Blocks Comparison

Another drawback is QANet models' training time. Figure 3 shows that the standard QANet trains around 10 hrs, much longer than the baseline model. However, as reducing the number of blocks from seven to four results. our training time decreased around 40%. Whether this is a result of idiosyncrasies within the particular data/run or a larger potential model complexity trade-off requires more exploration. However, if QANet's model complexity can be reduced to some degree without large ramifications on performance, it would surely stand to benefit in low-compute-resource settings.

5 Analysis

AvNA We want to first check if QANet can successful identify unanswerable questions. The result is shown in Table 3(a). Overall, our model classify questions as either answerable or unanswerable with a 76.5% correctness. It also wrongly answer unanswerable questions (15.33%) much more frequently than not answering an answerable question. This can be problematic because one would assume that it is worse to mislead a user rather than informing the user that the model is unable to provide an answer.

Potential solutions to this problem include (1). train a model or system that specifically predicts if a question is answerable, so only those that are predicted to be answerable are then passed into our QANet; (2). penalizing the probabilities in a way to reduce the likelihood of errors in predicting answers for unanswerable questions.

	+	-
A	2361 (39.67%)	912 (15.33%)
NA	487 (8.18%)	2191 (36.82%)

(a) All data.

	A+	NA+	NA-	A-	Correct %
how	217	199	40	104	74.29
others	112	164	17	55	79.31
what	1330	1279	306	520	75.95
when	212	156	31	57	80.70
where	108	74	19	47	73.39
which	110	67	17	22	81.94
who	240	219	43	99	76.37
why	31	33	14	8	74.42

(b) By Question Types

Table 3: AvNA. *Note.* +: ground truth: has an answer, -: ground truth: not have an answer; A: prediction: answered, NA: prediction: not answered

Correctness by Types of Questions We also want to examine our model performance across different types of questions. The seven types of questions in addition to "Others" are shown in Table 3(b). A question's type is determined by the following three rules: (1). if one and only one type keyword is observed in a question, we classify that question to the corresponding type. (2). if more than one keywords are observed (ex: a question contains both "what" and "how"), then it is classified to others. (3). all the "whose" and "whom" questions are classified as "who" questions.

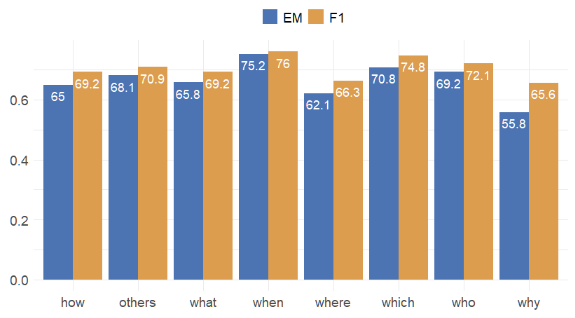


Figure 4: Scores by Question Types

We looked at two aspects of correctness: correctly predicting if a question is answerable (Table 3(b)), and the question type's corresponding f1/em scores (Figure 4). From our table, "where" and "why" questions have a relatively lower correctness. Examples include:

Question: What city later became Alaska?
Context: Kublai readied the move of the Mongol capital from Karakorum in Mongolia to Khanbaliq in 1264, constructing a new city near the former Jurchen capital Zhongdu, now modern Beijing, in 1266... Kublai proclaimed Khanbaliq the "Great Capital" or Daidu (Dadu, Chinese: <U+5927><U+90FD> in Chinese) of the dynasty. ...
Answer: N/A
Predicted: Beijing

At a glance, a human reader would know that the context is about Mongol history and is unrelated to Alaska. However, since the context contains many city names, if the model cannot tell the systematic difference between Alaska and cities like Karakorum and Beijing, it is very likely to answer incorrectly. This shows one of the major benefits of using pre-training and fine-tuning, since the pre-training step allows the model to learn implicit information about the world such as the fact that Alaska is far away from Mongol Empire and thus can deem the question as unanswerable. Without using explicit pre-training, however, it may still be possible to train on text that contains geographical information such that the model still has some of the factual information within itself.

From Figure 4, "why" and "where" questions are still the low-performers. Since EM score measures the exact match (the same start and end index), it is extremely difficult (even for humans) to provide an exact match to a provided answer. Examples include:

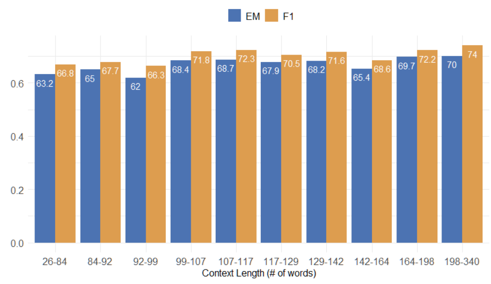
Question: Why is Warsaw's flora very rich in species?
Context: The flora of the city may be considered very rich in species. The species richness is mainly due to the location of Warsaw within the border region of several big floral regions comprising substantial proportions of close-to-wilderness areas ...
Answer: the location of Warsaw
Predicted: the location of Warsaw within the border region of several big floral regions

Even though this question received an EM score of 0 and F1 score of 0.43, one can argue that the predicted answer should be partially considered as correct and should have probably received a higher score. This demonstrates the trickiness of evaluating "why" questions, because the answers can often come in different forms. This particular example also suggests that reducing the maximum answer length allowed could potentially help with performance as the prediction is over-predicting the span. Below is another example that shows answering "why" questions may require extremely strong reading comprehension for the model in understanding cause and effect and temporality:

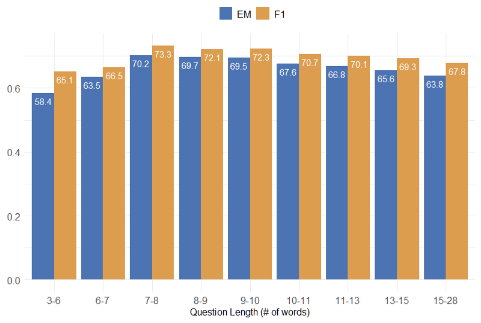
Question: Why did the Shah of Iran gave an interview?
Context: On October 6, 1973, Syria and Egypt, with support from other Arab nations, launched a surprise attack on Israel, on Yom Kippur. This renewal of hostilities in the Arab–Israeli conflict released the underlying economic pressure on oil prices. At the time, Iran was the world's second-largest oil exporter and a close US ally. Weeks later, the Shah of Iran said in an interview: "Of course [the price of oil] is going to rise..."
Answer: renewal of hostilities in the Arab–Israeli conflict
Predicted: Of course [the price of oil

Lengths vs. Performance Context lengths is of interest because one might expect that a short context length helps narrow down irrelevant and distracting variables and thus improve the score, while others may argue that a longer context length allows the model to learn more. In Figure5a, the x-axis is determined by the quantiles of the lengths, so that each x has the same number of observations. The result seems to support the latter argument. Therefore, to improve performance, one can try to increase context lengths.

Another variable of interest is question length, because a longer and thus likely more complex question may lead to a worse answer. The results in Figure5b demonstrates that a question that is either too short or too long can affect performance.



(a) Scores by Context Length



(b) Scores by Query Length

Figure 5: Lengths vs. Performance

6 Conclusion

Question-answering will always be an important issue for NLP research. Not only does it represent an important litmus test for how well systems understand language and reasoning, it ultimately impacts much of modern technology that a person would interface with. While most state-of-the-art QA models rely on pre-training and fine-tuning strategies, it is still helpful to have models that do not use pre-training to understand how well the underlying systems are performing.

Our achievements include adding character level embeddings to the baseline model, building the QANet model by referring to the original paper, tuning the hyperparameters, and evaluating both quantitatively and qualitatively our model performance. Our results show that QANet's transformer-based QA model does reasonably well on the SQuAD v2.0 dataset, with a final test F1 of 69.11 and an EM of 65.85. We have also highlighted the possibility of reducing model complexity in favor of training time as long as the overall performance of the model stays in a comparable range.

This project has its own share of limitations. First, the hyperparameter tuning was done in a sequential manner, which biases the results due to utilizing the development data more than once. Instead, a grid search is necessary to make sure the hyperparameters are tuned without internal bias, which requires far more time and compute resources. Moreover, we had a limited sample examining the relationship between the encoder blocks and block size on the overall model performance. While our initial tests seemed to indicate that we could reduce model complexity without overly harming the performance, it would be good to perform further analysis on different datasets to make sure the effects presented here are generalizable. Finally, at the outset of the project we wanted to be able to adapt parts of the hierarchical attention fusion networks from Wang et al. (2018) into the QANet architecture, because their work seems to suggest that fusion networks would help attention information flow more freely throughout the model. Due to time constraints we were not able to fully implement the additions, but we think that this is still a potentially fruitful avenue of inquiry for finding the best-performing non-pretrained model.

References

[1] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. In *Association for Computational Linguistics (ACL)*, 2017.

[2] 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. *arXiv:1804.09541*, 2018.

[3] The stanford question answering dataset leaderboard. `https://rajpurkar.github.io/SQuAD-explorer/`.

[4] Wei Wang, Ming Yan, and Chen Wu. Multi-granularity hierarchical attention fusion networks for reading comprehension and question answering. In *Association for Computational Linguistics*, 2018.

[5] Cs 224n default final project: Building a qa system (iid squad track). `http://web.stanford.edu/class/cs224n/project/default-final-project-handout-squad-track.pdf`.

[6] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[7] Rupesh Kumar Srivastava, Klaus Greff, and Jurgen Schmidhuber. Highway networks. *arXiv:1505.00387*, 2015.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Lukas Kaiser, and Illia Polosukhin. "attention is all you need". In *Association for Computational Linguistics (ACL)*, 2017.

[9] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv:1606.05250*, 2018.