# Improved QANet on SQuAD 2.0

Stanford CS224N Default Project

**Chenyang Dai**
Stanford University
qddaichy@stanford.edu

## Abstract

In this project, we built reading comprehension (a.k.a question answering) systems for the Stanford Question Answering Dataset (SQuAD) 2.0 focusing on two models (BiDAF[1] and QANet[2]) and their variations for the best performance. We have three major contributions. Firstly, we improved the baseline BiDAF model by introducing learnable character embeddings and a co-attention fusion function. Secondly, we implemented QANet from scratch and searched for the best model size. Finally, we explored QANet variations and showed that using an input embedding refine layer and a condition output layer can further improve original QANet performance. More precisely, input embedding refine layer better refines the word and character embeddings and condition output layer allows the answer's ending position prediction to be based on the starting position. Our best QANet single model achieved **64.82 / 67.81** EM and F1 score on non-PCE leaderboard testing sets.

## 1 Key Information

- TA mentor: Grace Lam

## 2 Introduction

Reading comprehension is a task for AI systems to understand a passage of text well enough to answer questions about it, where the text is presented in natural language. More specifically, given a context paragraph and a question, the system needs to either output a continuous span from the context that answers the question or predict that the question is not answerable.

This is a challenging task that traditional machine learning approaches tend to struggle with for the following reasons: First of all, reading comprehension requires accurate modeling of each word's semantics in the context and question. Without a decent understanding of words meaning, it is almost impossible to answer any questions. Secondly, it requires the system to keep track of long term information, as it may be necessary to combine information from different parts of the context to figure out the answer. Finally, not all questions are answerable. The system not only has to develop a deep understanding of the context, but needs to determine if the context and the questions are related as well.

Stanford Question Answering Dataset (SQuAD 2.0) is one of the most common ways to determine the performance of an AI system in reading comprehension. The dataset contains many (context, question, answer) triples where each context is an excerpt from Wikipedia. The question is the question to be answered based on the context. And the answer is a span from the context. Currently, the most successful models on SQuAD 2.0 leaderboard have been based on large pre-trained language models like ALBERT[3] with the ensemble approach.

However, in this work, we are restricted from using pre-trained models because the purpose of this work is to improve model architecture design for SQuAD 2.0. We explored the performance of two deep learning model architectures and their variations: Bidirectional Attention Flow (BiDAF) and

QANet. We applied modifications to BiDAF and implemented QANet from scratch. Moreover, we found that better refining input embeddings and conditioning end prediction on start prediction can significantly boost QANet performance.

## 3 Related Work

Attention mechanism is the key idea to tackle the SQuAD challenge for both BiDAF and QANet. BiDAF first introduced the bi-directional attention idea. It means attention should flow both ways – from the context to the question and from the question to the context. This two-way attention representation allows the model to better understand given context while paying attention to the question at the same time. However, except the bi-directional attention layer, BiDAF still uses RNN based approach (LSTM[4]) to refine the representation within context and questions as well as to transform the bi-directional attention representation into answer prediction.

As transformer takes off, the pure attention based approach is gradually replacing recurrence in the NLP area. QANet is a feed-forward model that consists of only convolutions and self-attention for reading comprehension. It adopts the core bi-directional attention idea from BiDAF while replacing all recurrence components with multi-head self-attentions and depthwise convolutions[11]. The motivation behind the model design is that convolution captures the local structure of the text, while self-attention learns the global interaction between each pair of words. In the original QANet paper, the authors point out two weaknesses of the traditional RNN approach. Firstly, the sequential nature of RNN prevents parallel computation because tokens must be fed into RNN in order. Slow training and inference of RNN based QA model prevents the systems from being trained on large dataset or being deployed in real-time applications. Secondly, RNNs have difficulty modeling long dependencies. Even though GRU and LSTM solved the problem to some extent, it is still not clear if such methods can handle complicated tasks such as reading comprehension. As a result, the QANet model surpassed all existing approaches to SQuAD challenges at that time.

## 4 Approach

**BiDAF baseline.** The baseline model we used was the BiDAF model with 300-D pre-trained GloVE[5] embeddings provided in the final project handout. After training, it achieved 57.65 EM 60.90 F1 on dev set.

**BiDAF with character embeddings.** We improved the embedding layer of the baseline model by introducing character level embedding. Beside the 300-D GloVE vector, each word will have an additional 200-D learnable character-level embedding concatenated to the word vector. The 200-D character-level embedding is obtained by passing the character embedding matrix through a 1D convolution layer and a max pooling layer. It takes maximum value of each row of this matrix to get a fixed-size vector representation of each word. We also changed the optimizer from Adadelta to Adam[6] for training. This improved model achieved 59.95 EM 63.25 F1 on dev set.

**BiDAF with character embeddings and fusion function.** Suggested by the original paper, we also introduced a fusion function after the co-attention layer to better fuse different attention components. Fused attention is computed as: $A_{fuse} = ReLU(W[c, a, c \odot a, c \odot b] + b)$. Fusion function further improves model performance to 62.28 EM 65.39 F1 on dev set.

**QANet.** We implemented the QANet model from scratch. There are several important model details not cover in the original QANet paper so we are trying to implement those details based on our own understanding. We reused the context query attention layer and Highway encoder[7] provided in the starter code. We also used a standard positional encoding function from official pytorch documentation [12]. Everything else was implemented ourselves using pytorch standard library. Figure 1 has the overall model structure and the following are our implementation details:

1. **Input Embedding Layer**. We reused the same embedding layer from BiDAF as described above. The output of a given word $x$ from this layer is a 500-D word embedding and character embedding concatenation $[x_w; x_c]$.

2. **Embedding Encoder Layer**. The core building block of QANet is encoder block. It consists of a sinusoidal positional encoding layer, followed by X convolution layers, a self multi-head attention layer using 8 heads, and a feed-forward layer as shown in the right part of Figure 1. Except the positional encoding, all the layers within the encode block has residual connections. The self attention layer and the feed-forward layer are standard transformer[8] building blocks. The convolution layer is kind of special. We adopted the idea from the original paper and used depthwise separable convolutions for memory efficiency. This layer uses one encode block with 4 convolution layers. One thing to notice is that the input dimension to this layer is 500. Before passing through the encoder block, We added a projection layer to map the 500 dimensional input to 128 (default hidden size) dimensions.

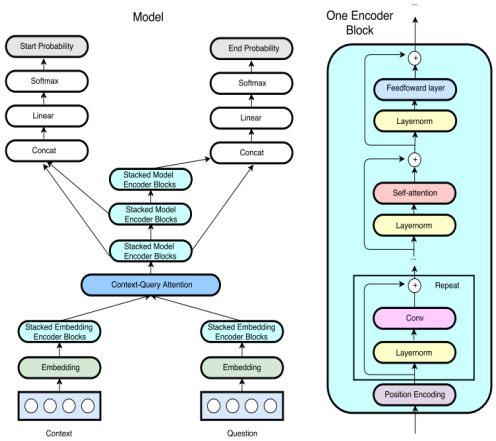


Figure 1: QANet architecture

3. **Context-Query Attention Layer**. This layer is the heart of the QANet model. We used the bidirectional attention flow idea from BiDaF paper. The context-to-query attention layer works as follows: It first computes a similarity matrix $S \in R^{n \times m}$ for each pair of words in context and question using similarity function: $f(q, c) = W_0[q, c, q \odot c]$

To compute context to question attention, this layer normalizes each row of $S$ by applying the softmax function, getting a matrix $\overline{S}$. And the context-to-query attention is computed as $A = \overline{S} \cdot Q^T \in R^{n \times d}$.

To compute question to context attention, this layer computes the column normalized matrix $\overline{\overline{S}}$ of S by softmax function, and the query-to-context attention is $B = \overline{S} \cdot \overline{\overline{S}}^T \cdot C^T$. Finally, the output of this layer is a concatenation of 4 parts $[c, a, c \odot a, c \odot b]$.

4. **Model Encoder Layers**. There are three model encoder layers stacked together and weights are shared among the 3 repetitions. Each layer is very similar to the embedding encoder layer except that it uses 7 encoder blocks, with each containing 2 convolutions. The input to this layer is $4 \times 128$ because the previous layer has four parts concatenated together. Once again, before passing through the encoder block, We added a projection layer to project the input to 128 dimensions.

5. **Output layer**. This layer predicts the start and end position of the answer as:
$P_{start} = softmax(W1[M0; M1])$, $P_{end} = softmax(W2[M0; M2])$, where $W1$ and $W2$ are two trainable linear transformations that maps the embedding to a single value and $M0$, $M1$, $M2$ are respectively the outputs of the three model encoders.

Finally, the loss function is a standard average negative log-likelihood over all the training examples: $L = (-1/N) \sum_{i=1}^{N} [log(p^1_{y_i^1}) + log(p^2_{y_i^2})]$ where $y_i^1$ and $y_i^2$ are respectively the ground truth starting and ending position of example $i$. Regular QANet achieved 63.25 EM 66.63 F1 on dev set.

**QANet Variations.** Besides the original QANet architecture, we explored several model variants for better performance. The detailed results of these variations are in the Experiments section.

1. **Input embedding layer with convolution and linear projections**. In the original paper, it is not clear how the 300-D word embedding and 200-D character embedding are concatenated and projected to the 128-D hidden size. The BiDAF starter code directly projects the concatenated embedding through a linear layer. Worrying about losing information with a single linear projection, we added two additional 1D convolutions while adopting the linear projection from the starter code (Figure 2).

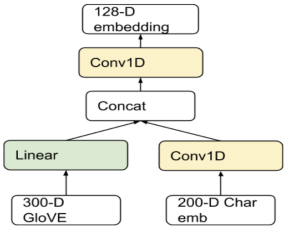


Figure 2: Improved Input Embedding Layer

The first convolution refines the 200-D character embedding into 128-D hidden size and the second convolution further refines the 256-D concatenated embedding into the final 128-D representation. We are inspired by the BiDAF fusion function when designing this layer. It allows word embedding and character embedding to first refine themselves and then it fuses the embeddings together. This original variation gives us a performance boost (+0.8 EM, +0.5 F1).

2. **Output layer with conditioning end prediction on start prediction**. In the QANet paper, the start and end positions ($P_{start}$, $P_{end}$) are predicted independently. The $P_{end}$ prediction only depends on the hidden states $M_0$ and $M_2$ from the first and third encoder layers. However, intuitively, it's helpful to know where the answer starts when predicting the end of the answer. We designed the new output layer with this conditioning (Figure 3).

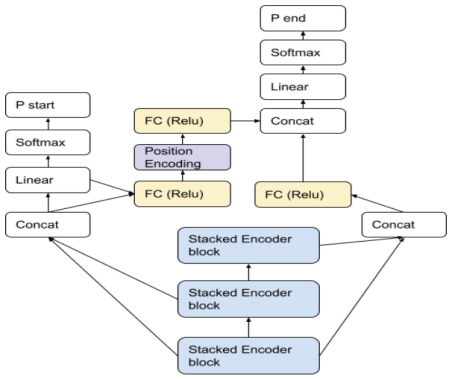


Figure 3: Improved Output Layer

The $P_{start}$ is computed the same as before, the calculations for $P_{end}$ are as follows:

$$A = [M0; M1] \quad B = [M0; M2]$$
$$A_{weight} = W_0 A \quad B_2 = ReLU(W_1 B)$$
$$A_{weighted} = A \circ A_{weight}$$
$$A_2 = ReLU(W2 A_{weighted})$$
$$A_2' = PositionEncoding(A_2)$$
$$A_3 = ReLU(W_3 A_2')$$
$$p_{end} = softmax(W_4[A_3; B_2])$$

$M_0, M_1, M_2 \in R^{h \times l}$ are the hidden states from the stacked model encoder layers and $W_0, W_4 \in R^{1 \times 2h}, W_1, W_2 \in R^{h \times 2h}$ and $W_3 \in R^{h \times h}$ are learnable parameters. We first compute the starting position's weight distribution $A_{weight}$ as usual. Then, we compute an element-wise product between $A$ and $A_{weight}$ using broadcasting to get $A_{weighted}$. Words in $A_{weighted}$ with

higher probability of being the answer start will be more activated than others. Next, the $A_{weighted}$ is sent through a fully connected layer and a position encoding[8] function for position information hardening. Finally, the output is passed through another fully connected layer and used as additional information when predicting the end position. This improved output layer gave us a remarkable performance boost (+1.5EM, +1.5 F1).

3. **Model encoder layer with non-shared weights**. Based on the paper, weights are shared between each of the 3 repetitions of the model encoder. We explored the idea of not sharing weights and letting each encoder layer learn on its own. We observed that the output from the second encoder layer is used for $p_{start}$ prediction and the output from the third encoder layer is used for $p_{end}$ prediction. Intuitively, the second and third encoder layers should have different weights for predicting $p_{start}$ and $p_{end}$ separately. Unfortunately, not sharing weights hurts the performance (-2.7 EM, -2.8F1, early stop) by a lot. We believe sharing weights help performance because it allows the encoder layers to encode information more consistently, the real start/end position prediction happens in the linear sublayer within the output layer.

4. **Modified feed forward layers**. The original paper didn't explain how the feed forward layers are configured. Given QANet adopts transformer architecture heavily, our initial assumption was that it uses the same parameters as the transformer: $FF(x) = relu(xW_1 + b_1)W_2 + b_2$.
Specifically, $W_1 \in R^{4h \times h}$ and $W_2 \in R^{h \times 4h}$. The dimension of input and output is $h = 128$, and the inner-layer has dimension $d_{ff} = 4 \times h = 512$. However, we discovered that using the same dimension for input, output and inner-layer gives us better performance (+1.0 EM, +1.0 F1). Our hypothesis is that given FF layer is repeated many times within QANet, using the $4h$ inner dimension will scale up the model a lot. Unlike the original paper, we didn't apply data augmentation and the training set is relatively small, using big FF layer may cause the model to under fit. Consequently, we used $W_1, W_2 \in R^{h \times h}$ for all FF layers in all our experiments.

5. **Stochastic Dropout**. The paper mentioned using stochastic dropout[9] to shut off sublayers with an encoder block for better regularization but didn't explain in detail. We implemented stochastic dropout based on our understanding. Within an encoder block there are N convolution sublayers followed by the self-attention and the feed forward sublayers. In total, there are L = 2 + N sublayers in a given block. Then, within each individual block each sublayer "survives" according to the following probability equation: $p_l = 1 - \frac{l}{L}(1 - p_L)$
where $p_L$ = 0.9 is the survival probability of the final layer. We didn't measure the actual performance gain but it is clear that the model's over-fitting tendency during training is mitigated by applying Stochastic Dropout. As a result, we applied Stochastic Dropout for all the experiments.

6. **Scale up / down QANet**. We experimented four scaled versioned of the QANet to observe how model size affects performance. The first version, QANet small, contains 3 encoder blocks rather than 7 with single head attention. The second version, QANet medium, contains 5 encoder blocks instead of 7 with 4 attention heads . The third version, QANet large, increases the number of encoder blocks to 9 with 8 attention heads. Finally, QANet large (2) uses the same configuration but with one additional encoder block in the embedding encoder layer and 160 as hidden size instead of 128. Please see next section for their performances.

7. **Ensemble model**. Due to the time constraint as well as the unexpectedly long training time for QANet, we only briefly touched on ensemble model. We assemble 3 models: BiDAF with character embedding and fusion function, regular QANet and QANet with improved embedding and output layers. The ensemble method is majority voting for start and end positions separately where all models are weighted equally. This ensemble model gives a slight performance boost (+0.6 EM, +0.6 F1).

# 5 Experiments

## 5.1 Data

We only used the SQuAD 2.0 dataset provided by the default project setup.py script. It has three splits: train, dev and test. The train set consists of roughly 130K examples in the official SQUAD 2.0 training set. The dev set and test set are roughly equally sized with about 5.5K examples each.

| Model Name | Dev F1 | Dev EM | Dev AvNA | Test F1 | Test EM |
|---|---|---|---|---|---|
| BiDAF | | | | | |
| BiDAF (Baseline) | 60.90 | 57.65 | 68.14 | - | - |
| BiDAF + Char Embedding | 63.25 | 59.97 | 70.87 | - | - |
| BiDAF + Char Embedding + Fusion | **65.39** | **62.28** | **71.52** | - | - |
| Scaled QANet | | | | | |
| QANet small (3 blocks, single head) | 64.80 | 61.35 | 71.11 | - | - |
| QANet medium (5 blocks, 4 heads) | 65.39 | 61.5 | 72.09 | - | - |
| QANet (7 blocks, 8 heads) | **66.63** | **63.25** | **72.89** | 65.11 | 61.52 |
| QANet large (9 blocks, 8 heads) | 66.15 | 62.87 | 71.18 | - | - |
| QANet large (2) (160 hidden size) | 65.16 | 61.86 | - | - | - |
| Improved QANet | | | | | |
| QANet + Condition Output Layer | 68.10 | 64.73 | 73.70 | - | - |
| QANet + Input Emb Refine Layer + Condition Output Layer | **68.61** | **65.54** | **73.90** | **67.81** | **64.82** |
| Ensemble Model | | | | | |
| Bidaf + QANet + QANet Improved | 69.26 | 66.21 | 74.26 | - | - |

Table 1: Performance of experimented models

### 5.2 Evaluation method

Different models' performances are primarily evaluated based on the Exact Match (EM) score, F1 score as well as Answer versus No Answer (AvNA) score.

### 5.3 Experimental details

**BiDAF** We trained the baseline BiDAF model with no modification using default parameters as described in the original BiDAF paper.
We trained modified BiDAF models with 200-D character embedding using Adam optimizer with $\beta_1 = 0.8$, $\beta_2 = 0.999$ and $\epsilon = 10^{-7}$. We used hidden state size of 128, batch size of 32, EMA decay of 0.999, dropout rate of 0.2, and learning rate of 0.001.
We trained the modified BiDAF with character embedding and fusion function using the same configuration.

**QANet** For QANet and its variations, We use Adam optimizer with $\beta_1 = 0.8$, $\beta_2 = 0.999$ and $\epsilon = 10^{-7}$. Learning rate is inverse-exponential in first 1000 steps and constant at 0.001 afterwards. We used batch size of 32 or 16, EMA decay of 0.9999, dropout rate of 0.1, character embedding dropout rate of 0.05 and L2 weight decay $3 \times 10^{-7}$. Except the QANet large (2) (hidden size 160), all other models use hidden size of 128. The batch size for training BiDAF is 32 but we have to cut the batch size to 16 for some QANet variations to fit into GPU memory.
As described in the Approach sections, all the experiment QANet models adopted Stochastic Dropout with $p_L = 0.9$ and are using $W_1, W_2 \in R^{h \times h}$ for all feed forward layers.

### 5.4 Results

Table 1 summarizes all our model experiment results on the non-PCE leaderboard.

**BiDAF** Introducing learnable 200-D character embedding greatly improved the BiDAF performance. We believe character level embedding allows the model to better capture the the meaning of words and subwords. Consequently, this leads to better prediction of the answer. Adding fusion further improves the performance. Rather than simply concatenating bi-direction attentions, the model can better fuse the attentions after the attention layer, which can be useful when predicting answers. The improvement to BiDAF aligns with our expectation.

**Scaled QANet** QANet in general exceeds the best BiDAF suggesting it is a more powerful model design. Surprisingly, non-of the scaled QANet out performs the regular QANet indicating that the original model size is at least at local optimal. This observation is a bit counterintuitive because

according to scaling law[10], scaling up the model should help with performance. Our hypothesis is that the bottleneck is not the model size but the training data size or the model architecture. Blindly scaling up QANet without more training data or more helpful features (e.g. better embedding, output conditioning) will only make the model underfitting.

**Improved QANet** Among all our single models, QANet with input embedding refine layer and condition output layer performs the best (F1:67.81 EM:64.82 on test set). This observation is sensible because refining the input embedding can better represent each word and can provide more information for answer prediction. Also, condition the end prediction based on the start helps the performance a lot. Knowing the answer start word and position will lead to a more reasonable end word prediction.

**Training time** We observed a much longer training time for regular QANet (15+ hours on Azure NC6) compared with BiDAF (3 hours on Azure NC6), which contradicts the QANet paper conclusion that QANet training speed is 3x to 13x faster. We believe hardware is the limitation. QANet is a much larger model than BiDAF and Azure NC6 instance can barely fit all parameters in GPU memory leaving almost no space for loading large training batches. As a consequence, we cannot benefit from the parallel computing when training QANet.

## 6 Analysis

We examined the performance of the best single model to understand the strengths and limitations of the improved QANet. In dev set, there are 5951 questions in total and they are classified by the leading question words. If a question is not using any leading question words, it is classified as other category. Number breakdown: $what = 2759$, $when = 440$, $where = 231$, $who = 601$, $which = 146$, $how = 525$, $why = 824$, $whose = 24$ and $other = 1131$.

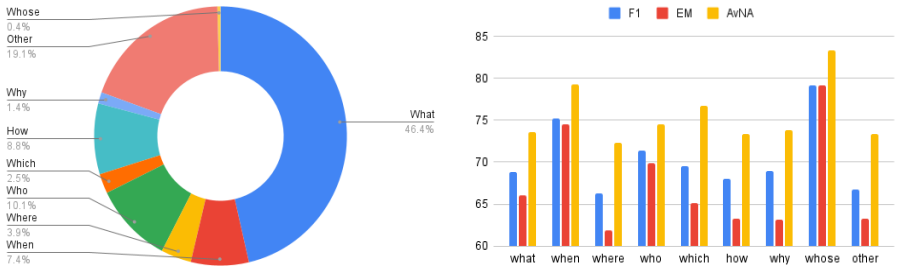


Figure 4: Question categories breakdown

**Performance by question categories** We first examined the model performance by different question categories. We observed relatively consistent performance across all categories (61.9-79.1 EM, 66.2-79.17 F1). The top 3 model performing categories are "Whose", "When" and "Who". The model achieves the highest score (79.17 EM, 79.17 F1) on **"Whose"** category, most likely due to this category has the smallest fractions ($0.4\%$ of all questions) and "Whose" is a strong indicator for a name. The model also performs well on **"When"** (EM: 74.55, F1: 75.26) and **"Who"** (EM: 69.87, F1: 71.36), which is sensible since these two questions words suggest that the answer should be a time or a person's name. The 3 least performing categories are "Where", "How" and "Why". **"How"** (EM: 63.24, F1: 68.04) and **"Why"** (EM: 63.10, F1: 68.90) are naturally more difficult to answer because these categories require reasoning and more complex learning. Unexpectedly, **"Where"** (EM: 61.09 F1: 66.23) is the worst performing category. Conceptually, "where" questions should be relatively easy to answer.

**Attention analysis for "Where"** To explain why the model has low performance on "Where" questions and to better understand attention mechanisms, we performed additional attention analysis on 20 wrongly predicted examples. After analysis, We realized that majority(16 out of 20) wrongly predicated examples are due to the model is trying to answer an unanswerable question. Taking a deeper look, the context of "Where" questions usually contains more than one locations and other important information. But the model tends to pay too much attentions solely on locations while ignoring other important information or paying attention to wrong places. For example:

| |
|---|
| **context**: Tamara de Lempicka was a famous artist born in Warsaw. She was born Maria Górska in Warsaw to wealthy parents and in 1916 married a Polish lawyer Tadeusz Łempicki. Better than anyone else she represented the Art Deco style in painting and art. Nathan Alterman, the Israeli poet, was born in Warsaw, as was Moshe Vilenski, the Israeli composer, lyricist, and pianist, who studied music at the Warsaw Conservatory. Warsaw was the beloved city of Isaac Bashevis Singer, which he described in many of his novels: Warsaw has just now been destroyed. No one will ever see the Warsaw I knew. Let me just write about it. Let this Warsaw not disappear forever, he commented. |
| **question**: Where was the famous artist Tamara de Maria born? |
| **answer**: []<br>**prediction**: Warsaw |

Table 2: Example uuid: aa91c5d46b54f54beb667febc

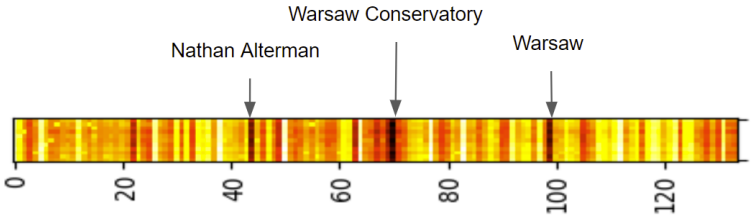


Figure 5: Co-attention for example uuid: aa91c5d46b54f54beb667febc

In the above example, it is important to realize that Tamara de Maria is never mentioned in the context so the question is not answerable. However, the attention map shows that our model is paying attention to various location names as well as the wrong person (Nathan Alterman). Our theory is that maybe the word embedding for person's name is not strong enough for the model to realize the person in the context and question are very different. Also, since "Where" is a strong signal for a location, the model is overly confident to predict the location mentioned multiple times in the context to be the answer.

After analyzing, we summarized that the model struggles mainly under two scenarios. Firstly, when the question is asking a place associated with a person or a group (e.g. the question asks person A's birth place but the context only mentions person B's birth place), the model sometimes cannot distinguish different subjects resulting in answering an unanswerable questions.
Secondly, on one hand, when the same location appears multiple times in the context, the model tends to predict that location as the answer. On the other hand, when there are multiple locations in the context, the model sometimes don't know which location to pay more attention to (Appendix for examples).
We believe that two potential ways to improve our model are either adding more "hard" training examples that the model struggles with to force the model learning or using pre-trained contextual embeddings that can better capture complex words relationships.

## 7 Conclusion

In conclusion, we improved baseline BiDAF with character embedding and fusion function. We presented an original implementation of QANet that achieves better results on the SQUAD 2.0 than BiDAF. We concluded that increasing or decreasing the model size does not help with the model's performance. Additionally, we improved the QANet model by adding refining to the input embedding layer and adding conditioning to the output layer. Finally, we conducted analysis on attention mechanism and understood why the model has relatively low performance under certain scenarios. Due to time constraint, we didn't explore ensemble model a lot. In the future, we would like to see how ensemble model can better improve performance.

# 8 Reference


[1] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *arXiv: 1611.01603*, 2016.

[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] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv: 1909.11942*, 2020.

[4] Ralf C. Staudemeyer, Eric Rothstein Morris. Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks. *arXiv: 1909.09586*, 2019.

[5] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word//w representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

[6] Diederik P. Kingma, Jimmy Ba. Adam. A Method for Stochastic Optimization. *arXiv: 1412.6980*, 2017.

[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 N. Gomez, Lukasz Kaiser, Illia Polosukhin. Attention Is All You Need. *arXiv: 1706.03762*, 2017.

[9] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV, pp. 646–661, 2016

[10] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, Dario Amodei. Scaling Laws for Neural Language Models. *arXiv: 2001.08361*, 2020.

[11] Lukasz Kaiser, Aidan N Gomez, and Francois Chollet. Depthwise separable convolutions for neural machine translation. *1706.03059*, 2017.

[12] Pytorch transformer tutorial. http://pytorch.org/tutorials/beginner/transformer_tutorial.html


# 9 Appendix

More wrongly predicted examples:

| |
|---|
| **context**: Their local rivals, Polonia Warsaw, have significantly fewer supporters, yet they managed to win Ekstraklasa Championship in 2000. They also won the country's championship in 1946, and won the cup twice as well. Polonia's home venue is located at Konwiktorska Street, a ten-minute walk north from the Old Town. Polonia was relegated from the country's top flight in 2013 because of their disastrous financial situation. They are now playing in the 4th league (5th tier in Poland) -the bottom professional league in the National – Polish Football Association (PZPN) structure.<br>**question**: Where is Ekstraklasa's home venue located? |
| **answer**: []<br>**prediction**: Konwiktorska Street |

Table 3: Example uuid: aa91c5d46b54f54beb667febc

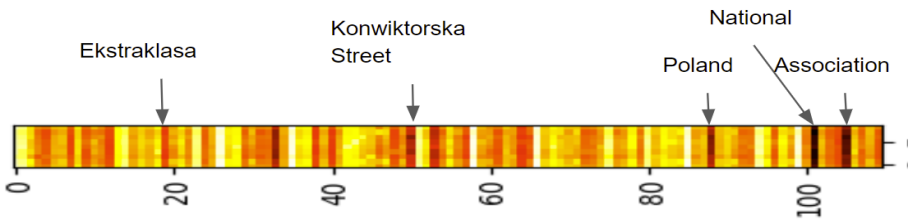


Figure 6: Example uuid: f7819f3c16cbb7ff50abe1c1d. Too many locations in context

| |
|---|
| **context**: Both before and after the 1708 passage of the Foreign Protestants Naturalization Act, an estimated 50,000 Protestant Walloons and Huguenots fled to England, with many moving on to Ireland and elsewhere. In relative terms, this was one of the largest waves of immigration ever of a single ethnic community to Britain. Andrew Lortie (born André Lortie), a leading Huguenot theologian and writer who led the exiled community in London, became known for articulating their criticism of the Pope and the doctrine of transubstantiation during Mass.<br>**question**: Where was Andrew Lortie originally from? |
| **answer**: []<br>**prediction**: London |

Table 4: Example uuid: 407e7e1eeeda0f3acd8cbc3dd

| |
|---|
| **context**: The origin of the legendary figure is not fully known. The best-known legend, by Artur Oppman, is that long ago two of Triton's daughters set out on a journey through the depths of the oceans and seas. One of them decided to stay on the coast of Denmark and can be seen sitting at the entrance to the port of Copenhagen. The second mermaid reached the mouth of the Vistula River and plunged into its waters. She stopped to rest on a sandy beach by the village of Warszowa, where fishermen came to admire her beauty and listen to her beautiful voice. A greedy merchant also heard her songs; he followed the fishermen and captured the mermaid.<br>**question**: Where did one of Oppman's daughters decide she wanted to hang out and stay? |
| **answer**: []<br>**prediction**: on the coast of Denmark |

Table 5: Example uuid: 68cf05f67fd29c6f129fe2fb9