Question Answering Training on SQuAD (QATS)

Carson Yu Tian Zhao, Louise Qianying Huang, Mi Jeremy Yu
Department of Statistics
Stanford University
{czhao333, qyhuang, miyu1996}@stanford.edu

Abstract

In this paper, we explore question answering in the SQuAD 2.0 framework. We built our end-to-end Question Answering (QA) system by combining ideas from high-performing SQuAD models: BiDAF, R-Net, and QANet. We first improve upon the provided BiDAF baseline with character embedding and a self-attention layer. We also implement QANet which outperforms BiDAF. Our final ensemble model achieved 70.12 F1 and 67.13 EM on the dev set, and 68.10 F1 and 64.75 EM on the hidden test set.

1 Introduction

The tasks of machine comprehension (MC) and question answering (QA) have gained a significant amount of scholarly attention over the past few years. The goal is to teach machines to read, process and comprehend text and answer questions given a passage or a document. In this paper, we specifically focus on the Stanford Question Answering Dataset (SQuAD) 2.0 released by Rajpurkar et al (8) in 2018. SQuAD 2.0 combines the 100,000 questions in SQuAD 1.1 with over 50,000 new, unanswerable questions. Such characteristics pose additional challenges to neural QA systems. Now, the system not only needs to answer questions accurately, but also needs to comprehend information sufficiency to determine whether a question is actually answerable.

Our system combines ideas from some of the best performing QA systems. On one hand, we improve upon the provided BiDAF baseline (9) by incorporating character level embedding and adding an extra layer of R-Net inspired (5) multiplicative self-attention after the bidirectional attention layer. On the other hand, we re-implement QANet based on the paper published by Yu and his colleagues (11). Details of modifications to the models can be found in Section 3. We also fine-tuned the models by extensive experimentation of various architecture choices and hyperparameters search, which can be found in Section 4. We report the results of our single models and ensemble model in Section 5. Comparisons between BiDAF and QANet and detailed error analysis can be found in Section 6.

2 Related Work

Research on the SQuAD dataset has been prevalent for the last few years, leading to numerous papers, vast improvements, and diverse models. Here, we outline a few that most contributed to our project. The baseline of our work is the BiDAF model (9), which uses a bi-directional attention flow layer to combine both Context-to-Query and Query-to-Context attention. R-Net is another successful model on SQuAD, which uses a self-matching attention mechanism to encode information from the context (5). Vaswani et. al [2017] demonstrates stacked self-attention, and how a model architecture with only attention mechanisms can outperform those with recurrent and convolutional layers (10). Finally, QANet incorporates both transformers and convolution layers, eschewing recurrent layers for a computational speed boost (11).

3 Approach

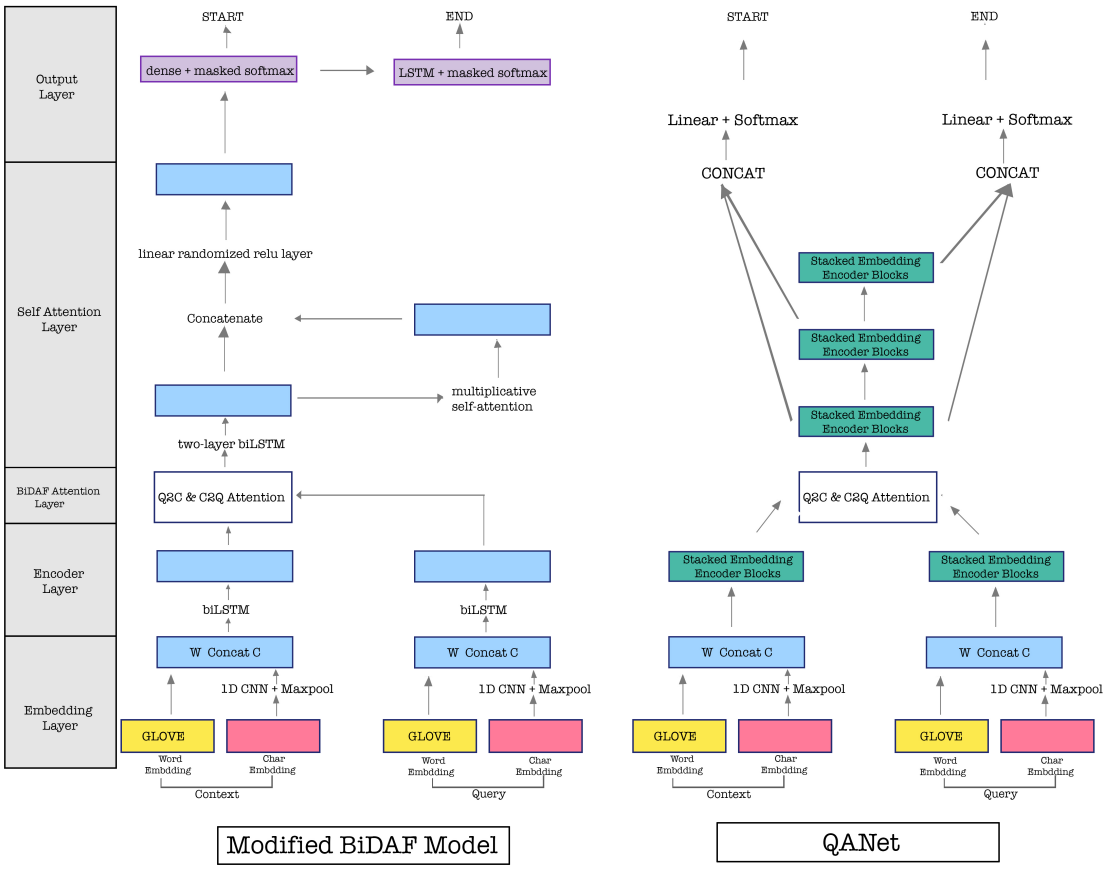


Figure 1: Modified BiDAF Architecture and QANet Architecture

We implement two different overall model architectures, BiDAF and QANet. Individual models are trained under different hyperparameters and slight changes (i.e. deeper layers, replacing linear with convolutional layers) to the architecture. Then, an ensemble is constructed by taking a weighted majority vote of the (start, end) tuple for each test observation.

3.1 BiDAF with Self-attention

The baseline model for our project is BiDAF (Bidirectional Attention Flow), which consists of an embedding layer, encoder layer, attention layer, modeling layer, and output layer. A more detailed description of the model can be found in (9). Our model improves upon the provided BiDAF baseline by adding character embedding and an additional self-attention layer. Refer to Figure 1 for an overview of the high-level model architecture.

3.1.1 Embedding Layer

The task of the embedding layer is to map each word token into a more dense representation using character-level and word-level features. The word-level embedding uses GloVe (7) and the character-level embedding uses 1D-CNN as described in (9). They are incorporated through a two-layer Highway Network. The output will be a dense representation for context $W^C \in R^{T \times d}$, and representation for query $W^Q \in R^{J \times d}$, where d is the hidden size for BiDAF and the number of convolution filters for QANet.

3.1.2 Encoding Layer

In the encoding layer, we use a bi-directional LSTM on top of both the query and context embeddings to get the temporal interactions between words. We have $H \in R^{T \times 2d}$ obtained from vector W^C and $U \in R^{J \times 2d}$ obtained from vector W^Q, where

$$h_t = BiLSTM_C(h_{t-1}, W_t^C) \tag{1}$$

$$u_t = BiLSTM_Q(u_{t-1}.W_t^Q) \tag{2}$$

3.1.3 BiDAF Attention Layer

We first of all construct a similarity matrix using H and U from the previous layer.

$$S_{tj} = W_{(S)}^T[h_t; u_j; h_t \circ u_j] \tag{3}$$

Context-to-Query(C2Q) attention. This attention captures which query words are most relevant to each context word by taking the weighted average of the question word representation based on the similarity of each question word to the context word, i.e.

$$a_t = softmax(S_{t:}) \tag{4}$$

$$\tilde{u_t} = \sum_{j=1}^{J} a_{tj} u_j \tag{5}$$

This will provide us with an attended query matrix for the entire context with dimension $\tilde{U} \in R^{T \times 2d}$.

Query-to-Context(Q2C) Attention. We will get the context words that have the closest similarity to each of the query words. We obtain the weight by taking the softmax of the max across the column.

$$b = softmax(max_{col}(S)) \in R^T \tag{6}$$

Using this weight, we can calculate the vector $\tilde{h} = \sum_t b_t H_{:t} \in R^{2d}$ which indicates the most important words in context with respect to the query. This will be done through all the context embedding columns in H, which will give us $\tilde{H} \in R^{T \times 2d}$. In the end, we will fuse the context representation along with the two attentions using concatenation, which will give us the final output of this layer G in the form

$$G = [h; \tilde{u}; h \circ \tilde{u}; h \circ \tilde{h}] \in R^{T \times 8d} \tag{7}$$

3.1.4 Self-Attention Layer

Inspired by R-Net, we change the BiDAF model by adding a self-attention layer (5). Currently we have the contextual representations conditioned on the questions, yet we do not have temporal information of these contextual representations. Our strategy is to, first of all, blend the contextual representations using two layers of LSTMs, which computes

$$m_{fwd} = LSTM(m, G) \in R^{T \times d} \tag{8}$$

$$m_{rev} = LSTM(m, G) \in R^{T \times d} \tag{9}$$

$$m_i = [m_{i,fwd}; m_{i.rev}] \in R^{2d} \tag{10}$$

The blending gives us $m \in R^{T \times 2d}$, the contextual information with the temporal information. This information is then further passed into a self-attention layer to build a deeper layer that can capture more temporal information among the query-aware context words. The type of attention used is multiplicative attention, which we found has faster performance than additive attention. The output of the blending m is passed into a linear layer, and the result is then multiplied by the temporal query-aware contextual information m again, and this will hopefully increase the temporal information the representation will carry. We then calculate the softmax and get the weighted average of the previous temporal information m, which we denote as c. Then this information is fused with the previous temporal information m to mediate between the original information and the new information we calculate from the self-attention. This is achieved through concatenation, a linear layer, and a

randomized Relu layer. Randomized Relu was chosen to add a little bit of randomness into the model. The above procedure can be expressed mathematically as the following:

$$S_{new} = mW_P m^T \in R^{T \times T} \tag{11}$$

Using this S_{new}, we get the softmax along the columns, similar to what was done earlier for the context-to-query attention.

$$a_t = softmax(S_{t:,new}) \in R^{T \times T} \tag{12}$$

Then c_t is calculated using weight a_t

$$c_{tk} = \sum_{j=1}^{T} a_{tj} m_{jk} \tag{13}$$

We concatenate together m, c, and the element-wise multiplication $m \circ c$ and feed that through another linear layer to get $M \in \mathbb{R}^{T \times 2d}$. Then, that is fed through the leaky randomized Relu. We have that

$$M = LeakyRRelu(W_m * [m; c; m \circ c] + b_m) \in R^{T \times 2d} \tag{14}$$

where $\circ$ stands for element-wise multiplication and ; stands for concatenation.

3.1.5 Output Layer

In the final layer, we compute a linear transformation of the BiDAF attention and the self-attention outputs. Then we proceed to take the softmax of the output of the linear layer to get the start pointer. After that, we apply another biLSTM to the self-attention output, which is then fed into another linear transformation with the BiDAF attention output. This allows the model to incorporate both the original output of BiDAF attention layer and the information calculated using temporal information to mitigate the effect of incorrect attendances from any previous time step. To predict the end token, another bi-directional LSTM is used to strengthen the temporal signals. Finally, we compute the softmax of this second output, which yields the end pointer.

$$p_{start} = softmax(g * w_{att\_linear\_1} + M_1 * w_{mod\_linear\_1}) \in R^T \tag{15}$$

$$M_2 = BiLSTM(M) \in R^{T \times 2d} \tag{16}$$

$$p_{end} = softmax(g * w_{att\_linear\_2} + M_2 * w_{mod\_linear\_2}) \in R^T \tag{17}$$

The softmax operation uses the context mask, and the training loss is the sum of the negative log probabilities of the true start and end indices by the predicted distributions, averaged over all examples:

$$L(\theta) = -\frac{1}{N} \sum_{i}^{N} log(p_{y_{i,start}}) + log(p_{y_{i,end}}) \tag{18}$$

where θ represents all the weights in the model, N is the number of examples, p_k is the k-th value of the vector p, and $y_{i,start}$ and $y_{i,end}$ indicates the true start and true end. For testing, we simply pick the span (k, l) such that the probability $p_{k,start} * p_{l,end}$ is greatest.

3.2 QANet

QANet follows similar structures to BiDAF. It consists of an input embedding (same as BiDAF), embedding encoder, Q2C and C2Q attention (same as BiDAF), modeling encoder, and output layers (11). One major improvement of QANet over BiDAF is that QANet replaces recurrent architecture in BiDAF with convolutions and self-attention for faster computation. The reader may refer to Figure 1 for an overview of this model.

3.2.1 Important Components

We first provide a detailed description of one of the principal components in QANet, the encoder block.

Depthwise Seperable Convolution: The depthwise separable convolution can be understood as two convolutions. The depthwise convolution is performed independently over every channel of the input, and later the separable convolution projects each channel computed by the depthwise convolution

onto a new channel space. This idea was first explored by Chollet [2016] (3). Comparing to a regular 2D or 3D convolution, this is a powerful simplification under the assumption that the inputs the convolutions operate on feature both independent channels and highly correlated spatial locations.

Multi-Head Attention: The idea of multi-head attention adpats from Vaswani et. al [2017] (10). We project the queries, keys, and values h times with different linear projections to d_k, d_k, and d_v dimensions. Then each head is computed by putting the projected version through scalar dot product attention. Each i^{th} head is defined by

$$Head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{19}$$

where the projection matrices have dimension $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, and $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ The scaled dot-product attention has form:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}}V)) \tag{20}$$

At the end, the heads are concatenated and put through a final linear layer W^O. We have that

$$MultiHead(Q, K, V) = [head_1; \ldots; head_h]W^O \tag{21}$$

where W^O has the dimension $\mathbb{R}^{hd_v \times d_{model}}$ and ; represents concatenation. For this part, we adapted code from an official non-SQuAD Transformers-XL implementation for this part [1].

Positional Encoding: Since we get rid of the RNNs from BiDAF, there needs to be a way to encode the order of the sequence into the model. Positional encoding handles this issue. Therefore, we add positional encoding before every encoding block. The positional encoding takes the form:

$$PE(pos, 2i) = sin(pos/10000^{2i/d_{model}}) \tag{22}$$

$$PE(pos, 2i + 1) = cos(pos/10000^{2i/d_{model}}) \tag{23}$$

Such encoding might allow the model to easily learn the relative positions (10).

Layer Norm: The layer normalization is described in Ba et. al [2016] (1). One of the challenges of deep learning is that the gradients with respect to one layer are highly dependent on the previous layers. In order to reduce the shift caused by correlation, we need to normalize each layer. Normalization is given by the formula:

$$y = \frac{x - E[x]}{\sqrt{var[x] + \epsilon}} \times \gamma + \beta \tag{24}$$

which is calculated over the hidden size (filters) dimension. Here both γ and β are learnable affine transforms.

Position-Wise Feed-Forward Networks: In addition to the attention sub-layers, we also have a fully connected feed-forward network, which consists of two linear layers with input and output equal to d_{model}. The output of the first linear layer goes through a ReLU layer and then dropout. The inner layer has dimension $4 \times d_{model}$. For this part, we also adapted code from the official Transformers-XL (non-SQuAD) opensource implementation[1].

Encoder Block Overview
All of the above individual components are used to build the essential encoder block substructure in QANet. At the beginning of the encoder block, we incorporate positional encoding. The output of the positional encoding is then passed into several depthwise separable convolution layers. The corresponding output then goes through the multi-head attention layer. This output is then passed into the position-wise feed-foward layer. We use layernorm and residual connection between every sublayer so each output is of the form $f(layernorm(x)) + x$, where x is the original input into the sublayer. Figure 2 in the appendix shows one encoder block with all the layers.

3.3 Embedding Encoder Layer

For this part, we will use the encoder block introduced above to construct a even more condense representation for both context and query. The inputs of this layer are $W^C \in R^{T \times d_{model}}$ and

[1] $https://github.com/kimiyoung/transformer-xl/blob/master/pytorch$

$W^Q \in R^{J \times d_{model}}$, which are the outputs of the input embedding layer (same as BiDAF). The kernel size for all of the convolutions in the encoding block is 7. The number of filters is denoted as d_{model}. For our model, the number of filters and the hidden size of the embedding layer are the same. For the embedding encoder layer, we only use one encoder block for context and query, and the weights of the encoder block are shared between question and query. The embedding encoder layer output for context is $C \in R^{T \times d_{model}}$, and the output for query is $Q \in R^{T \times d_{model}}$.

3.4 Model Encoder Layer

This layer is used to process the information from the similarity matrix $G \in R^{T \times 8d_{model}}$ from the C2Q and Q2C layer (same as in BiDAF). We again use the encoder block structure. The parameters for the encoder blocks are the same as the embedding encoder layer except now we have 7 blocks, and the number of convolution layers is 2. We repeat the stacked encoder blocks 3 times, where the repetitions share weights with each other. The outputs of the three stacked encoder blocks are denoted as $M_0 \in R^{T \times d_{model}}$, $M_1 \in R^{T \times d_{model}}$ and $M_2 \in R^{T \times d_{model}}$.

3.5 Output layer

For our purposes, we predict the start and ending probabilities by first concatenating together M_0 and M_1, and M_0 and M_2. Then we feed each concatenation through separate linear layers without bias terms. Then we apply the masked softmax to get the start and ending probabilities.

$$p_{start} = softmax(W_1[M_0; M_1]) \tag{25}$$

$$p_{end} = softmax(W_2[M_0; M_2]) \tag{26}$$

We refer the readers section 3.1.5 for details of the loss function, which is the same as in BiDAF.

4 Experiments

4.1 Data

The Stanford Question Answering Dataset (SQuAD) 2.0 is a reading comprehension dataset comprised of paragraphs (context), questions, and answers (8). Questions are either answerable from their respective Wikipedia paragraphs or have no answer (NA). The task here is to select the start and ending point of the text that successfully answers the question posed (8). The dataset is split into train, dev, and test sets.

4.2 Evaluation method

The training loss used was the sum of the negative log probabilities of the true start and end indices by the predicted distributions. The two metrics used to evaluate the results were EM (Exact Match) and F1 score.

4.3 Experimental details

For both BiDAF with self-attention and QANet implementations, the character limit is set to 16, context word limit set to 400, and question word limit set to 50. To avoid exploding gradients in the LSTM layers, we use gradient clipping with a maximum gradient norm of 5.

In the BiDAF with self-attention model, we use EMA decay of 0.999, hidden size of 100, dropout rate of 0.2, and Adadelta optimizer with learning rate=0.6, weight decay=10^{-5}, and ρ=0.9 (coefficient used for computing a running average of squared gradients). A higher learning rate of 0.6 was chosen due to the fact that the model trained slowly in the first few epochs and the default PyTorch Adadelta learning rate was 1.0. Increasing the weight decay and learning rate improved the F1 score by 1.5. The weight decay and dropout counteract against observed overfitting in later epochs of models without the weight decay and dropout. Furthermore, increasing the hidden size to 150 did not improve the model, and deeper/stacked RNNs with residual connections in the attention and modeling layers worsened the performance. A decaying learning rate starting at 0.65 and decaying to 0.4 over the epochs also did not improve performance. In the self-attention layer, we choose multiplicative attention over additive attention for faster computation. Also, replacing the LSTMs

with GRUs decreased Dev F1 slightly by 0.3, but improved computational speed by a few minutes an epoch.

In our implementation of QANet, we experiment with convolutional filter numbers 96 and 128, number of heads in multi-head attention sublayer 1 and 4, and batch size 32, and we obtain the best performance with filter numbers 96 and single attention head. In the embedding encoder layer, we have 4 convolutional layers, kernel size 7, and 1 block. In the modeling encoder layer, we have 2 convolutional layers, kernel size 5, and 7 blocks. Our optimizer was ADAM with hyperparameters $\beta_1 = 0.8$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$. We follow faithfully the paper's learning rates, which warms up with an inverse exponential increase from 0.0 to 0.001 in the first 1000 steps (11). The learning rate is then a constant 0.001 for the rest of the steps. We use a higher dropout rate of 0.2 on the word and character embeddings, but keep the 0.1 dropout rate between every two layers mentioned in the paper (11). Instead of stochastic layer dropout, we instead do layer dropout a constant 0.05 of the time.

5 Results

	Dev Set F1 / EM	**Test Set F1 / EM**
Single Model		
BiDAF CS 224N Baseline	58.02 / 54.85	59.920 / 56.298
BiDAF + Char-CNN	63.14 / 59.65	
BiDAF + Char-CNN + self-attention	65.18 / 62.17	
QANet (1 attention head, D_{model} = 96)	**68.57 / 65.25**	
QANet (4 attention heads, D_{model} = 96)	68.10 / 64.75	
QANet (4 attention heads, D_{model} = 128	67.64 / 63.87	
Ensemble Model		
3 QANet ensemble	69.83 / 66.68	66.72 / 63.16
3 QANet + 2 best-performing BiDAF (5 ensemble)	**70.12 / 67.13**	**68.10 / 64.75**

The performance of our models are summarized in the above table. Our single model of modified BiDAF achieves 65.176 F1 and 62.174 EM on the Dev set. Our best single model of QANet achieves 68.57 F1 and 65.25 EM on the Dev set. Our final ensemble model, which consists of 3 QANet models and the 2 best performing modified BiDAF models, achieves 70.12 F1 and 67.13 EM on the Dev set, and 68.10 F1 and 64.75 EM on the hidden test set.

For BiDAF, we observe that adding character embeddings to the provided BiDAF baseline boosts the performance drastically, since character-level embeddings capture semantic and orthographic information that word vectors fail to learn. Adding a self-attention layer also enables a boost in the evaluation metrics. From Tensorboard, it is apparent that the performance after adding the self-attention layer outperforms the model without self-attention from the very beginning.
In terms of QANet, our hyperparameter sweep of number of attention heads and filter number (size of model) yields surprising results. We find that increasing the number of heads from 1 to 4 and increasing the filter number from 96 to 128 does not improve performance. From Tensorboard, we see that when we increase the number of attention heads and size of model, QANet overfits severely on the Dev set after 1.5 million iterations.

6 Analysis

6.1 Comparison between Modified BiDAF and QANet

- **QANet has precise answer span**

> **Context:** Fourth, national courts have a duty to interpret domestic law "as far as possible in the light of the wording and purpose of the directive"…
> **Question:** Which courts have a duty to interpret domestic law as far as possible?
> **Answer:** national courts
> **QANet Answer:** national courts
> **Modified BiDAF Answer:** Fourth, national courts

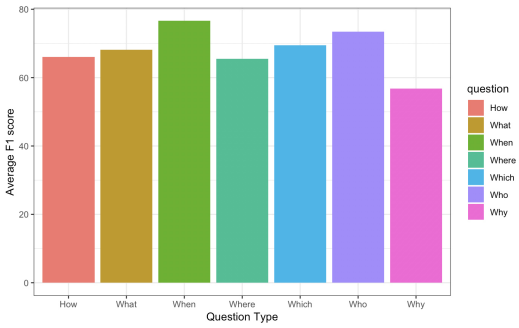
Compared to BiDAF, QANet often predicts answers with a more precise span boundary. In the above example, BiDAF prediction incorrectly includes irrelevant words to answer the question. QANet demonstrates better reading comprehension ability than BiDAF in general.

- **QANet predicts an answer for unanswerable questions**

  Context: …The two forces finally met in the bloody Battle of Lake George between Fort Edward and Fort William Henry….
  Question: Who won the battle of Lake Niagara?
  Answer: No Answer
  QANet Answer: Fort Edward and Fort William Henry
  Modified BiDAF Answer: No Answer

  Compared to BiDAF, QANet is more likely to generate predictions for unanswerable questions, which is a caveat and might limit its performance.

6.2 Error by Question Type



The graph above shows the average F1 score of the predicted answer by different question types. We observe that our model can do a good job in answering questions that involve low-level reasoning, such as "when," "who," and "what." However, our model struggles on questions that require deeper logical reasoning, such as "why" questions.

6.3 Qualitative Error Analysis

Below are some representative errors in the predictions of our final ensemble model.

- **Imprecise answer boundary**

  Context: …the totality of the exhaust steam cannot evacuate the cylinder, choking it and giving excessive compression (\"kick back\").[citation needed]…
  Question: What is another term for excessive compression?
  Correct answer: kick back
  Prediction: kick back").[citation

  The most common mistakes of our model is that it fails to decide on the correct boundary of the answer, and the answer is partially correct. This might also be due to the fact that there is an inherent ambiguity in deciding the boundary for an answer.

- **Incorrect prediction of No Answer to answerable questions**

  Context: …Because oil was priced in dollars, oil producers' real income decreased…from then on, they would price oil in terms of a fixed amount of gold.
  Question: Why did oil start getting priced in terms of gold?
  Correct answer: oil was priced in dollars, oil producers' real income decreased
  Prediction: No Answer

  We found that our model struggles to determine whether a question is answerable based on the context. Such mistakes are much more common in "why" questions, which require deeper logical reasoning than other types of questions.

7 Conclusion

In this paper, we implement an end-to-end neural Question Answering system for reading comprehension task on SQuAD 2.0. Our model combines high-performing models such as RNN-based BiDAF, and CNN and attention-based QANet. The final ensemble model achieves 70.12 F1 and 67.13 EM on the dev set, and 68.10 F1 and 64.75 EM on the hidden test set. By the time of submission, our model ranks 6^{th} place on the test leaderboard. From the result and the error analysis, we found that our model can effectively comprehend the provided context and synthesize information, but it struggles to determine whether a question is answerable. It also struggles to understand the contexts and questions that require more reasoning. Due to lack of time and computational resources, there are multiple more experiments we wanted to try. Further work includes making the QANet model faster, exploring alternative attention methods like Transformers-XL, and ensembling using average logits instead.

References

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

[2] D. Chen, A. Fisch, J. Weston, and A. Bordes, "Reading wikipedia to answer open-domain questions," arXiv preprint arXiv:1704.00051, 2017.

[3] François Chollet. Xception: Deep learning with depthwise separable convolutions. arXiv preprint arXiv:1610.02357, 2016.

[4] R. Liu, J. Hu, W. Wei, Z. Yang, and E. Nyberg, "Structural Embedding of Syntactic Trees for Machine Comprehension," arXiv preprint arXiv:1703.00572, 2017.

[5] Natural Language Computing Group, Microsoft Research Asia. R-Net: Machine Reading Comprehension with Self-Matching Networks.

[6] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. Ms marco: A human generated machine reading comprehension dataset. arXiv preprint arXiv:1611.09268.

[7] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In EMNLP, 2014.

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

[9] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. arXiv preprint arXiv:1611.01603, 2016.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, page 5998–6008. Curran Associates, Inc.

[11] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. 2018. QAnet: Combining local convolution with global self-attention for reading comprehension. arXiv preprint arXiv:1804.09541.

8 Appendix

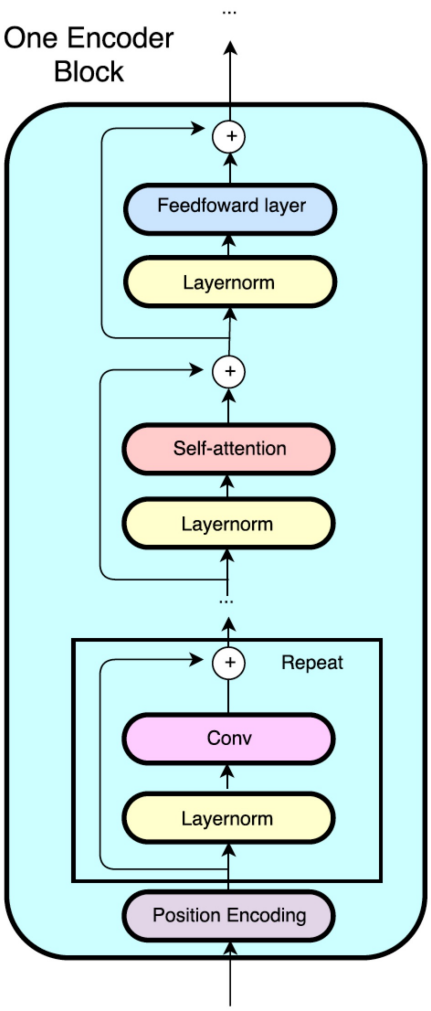


Figure 2: EncoderBlock for QANet, from (11)

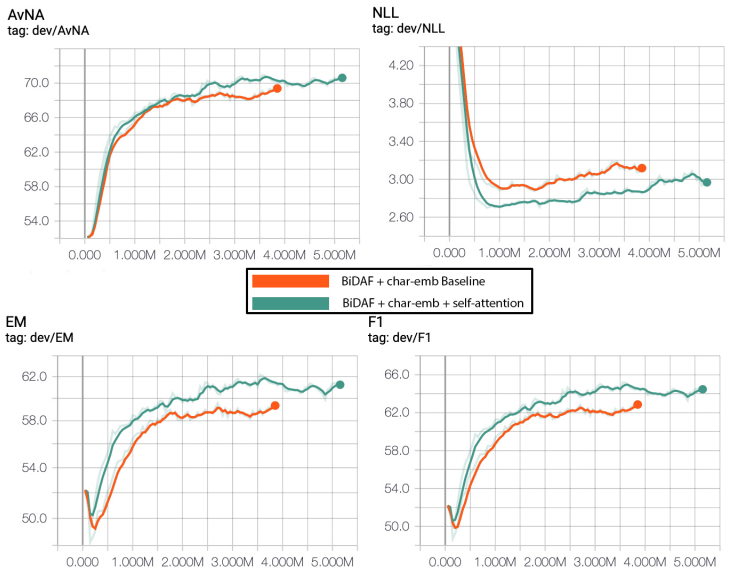


Figure 3: Training Statistics for 2 BiDAF models

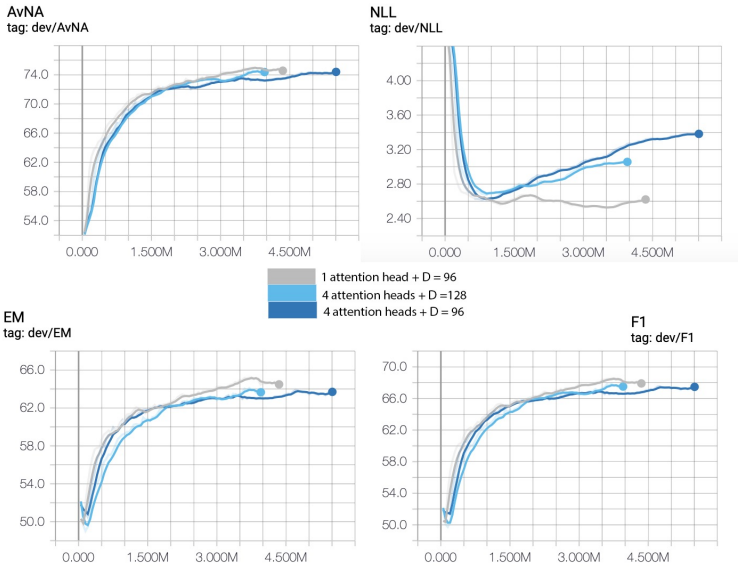


Figure 4: Training Statistics for 3 QANet models