Extending BiDAF and QANet NLP on SQuAD 2.0

Stanford CS224N {Default} Project

Minakshi M
Stanford University
adaboost@stanford.edu

S Mukherjee
Stanford University
suvasism@stanford.edu

Abstract

The goal for the project is to build an end to end neural network architecture for the question answering system that captures the complex interaction of Machine Comprehension (MC) between the context and the query. Our implementation is motivated by the recent high performance models that involve attention mechanism. We start with Bi-Directional Attention Flow (BIDAF) network as a baseline and aim to improve the MC of answering a query about a given context paragraph by making architectural changes to the baseline BiDAF model. QANet speeds up reading comprehension by replacing RNNs with encoder blocks. QANet model predicts the start and end indices of an answer or no answer from a context given a query. We improved BiDAF model performance by adding character embedding and modifying the attention mechanism. We also re-implemented the QANet to get better performance. Our best BiDAF model achieved 63.22 EM and 67.08 F1 score on validation set and QANet achieved 61.45 EM and 74.11 F1 score on validation set.

1 Introduction

The Bidirectional Attention Flow (BiDAF) end-to-end model [1](Seo et al. 2017) employs a recurrent architecture to process sequential inputs and uses an attention component to cope with long term interactions. BiDAF is slow for both training and inference due to the model's recurrent nature for long texts. QANet model makes machine comprehension faster by removing the recurrent nature of BiDAF model. Convolutions and self-attentions are building blocks of encoders that separately encode the query and context. The interactions between context and question is learned by standard attentions. [2](Xiong et al., 2016; Seo et al., 2016; Bahdanau et al., 2015). The resulting representation is again encoded with recurrency-free encoder before finally decoding to the probability of each position being the start or end of the answer span. Our motivation is to understand the limitation of QANet against SQuAD 2.0 dataset and improve the QANet architecture [3].

2 Related Work

It has been a prime topic both in academia and in corporate world as how well can a model be built such that the model can provide correct answer to a question from a given paragraph where the answer is a span from the context. [1] BiDAF was one of the earlier models that captured C2Q and Q2C attentions and presented a context representation to the Modeling layer that was well conditioned on the question.The. model predicts the spans with highest p_{start} and p_{end}. The Coattention layer from Dynamic Coattention (DCN) [15] model improved BiDAF's basic attention by attending to the question and context simultaneously and finally by fusing both attentions. DCN estimates the start and end points of the answer span multiple times, each time conditioned on its previous estimates.Hence, the model is able to explore local maxima corresponding to multiple plausible answers. R-Net [5] improved attention layer further by including a self-matching attention that refines the representation by matching the context against itself, which effectively encodes information from the whole passage. QANet took the language model to the next level by replacing

RNN with self-attention and convolution. It's Encoder Block uses stacked convolutional sublayers with depthwise-separable convolution to capture local dependencies in the input sequence. Later on these transformer based language models became the norm as BERT and GPT models were released. Also, neural machine comprehension and language models adopted the notion of pre-training a large language model using a large corpus and acquired world knowledge which then can be fine-tuned based on individual corpus.

3 BiDAF approach

3.1 Embedding

Baseline BiDAF uses only word-level embeddings. To handle OOV, **we added** character level embeddings. **We used** 1D-CNN to find numeric representation of words by checking their char-level compositions. **We also experimented** with 2D-CNN, but this turned out to be computationally expensive and the batch size had to be reduced from 64 (default) to 16 on 1 gpu azure.
A word is represented by $C \epsilon R^{dXl}$, d is the height of the matrix and l is the word length. The convolution filter (kernel) $H \epsilon R^{dXw}$ of width w, will scan the word. H is randomly initialized and adjusted during training. Then, we take max pool of the vector, f , Hadamard product of H and its projection on C. Fig 1 and Fig 2 below demonstrate the process. We adopted our idea from open-sourced TensorFlow BiDAF implementation Abadi et al., 2016.[14]. **We decided** to use Leaky ReLU as it's "mean activation" is close to 0 which makes training faster and it also fixes "dying ReLU" [9]problem.

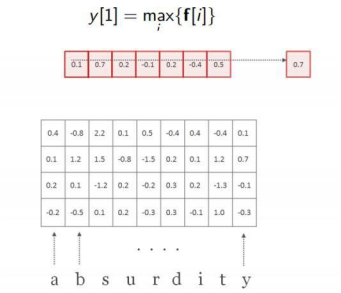

Figure 1: Char Representation.

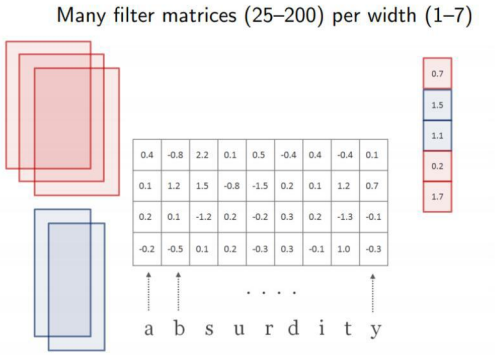

Figure 2: Summary scalars from different scanning processes

3.2 Highway Network

Highway network [4] adjusts the contribution of the word embedding and the character embedding steps. Since character embedding deals with OOV, highway network increases the 1D-CNN representation. Now, we have obtained two sets of vector representations for words: one from the GloVe (word provided in baseline) embedding and the other from 1D-CNN (character added) embedding. These 2 representations are vertically concatenated. The concatenated embedding matrix(height d - height of the word matrix and char matrix) from both context and query are passed to the Highway Network as an input. **We experimented** with 1D-CNN instead of Linear (default) in the highway network, this resulted in memory issues, so we reverted back to default Linear function. In highway network, only a fraction of the input is subjected to transformation and rest is allowed to pass.
If y be the input to highway network, W_H and b_H be the affine transformations, the transformation gate, $t = \sigma(W_H y + b_y)$, the carry gate will be $1 - t$.
The highway network transforms the input to, $z = t \cdot g(W_H + b_H) + (1 - t) \cdot y$.
Highway network has 2 outputs, one for context of dimension (dXT), where T is the length of context and another one for query (dXJ), where J is the length of the query.

3.3 Contextual Embedding

The output of the highway network doesn't take into account the contextual meaning of word. So RNNEncoding implements the contextual embedding using the Long-Short-Term-Memory (LSTM) sequences (default). The output of this layer will contain contextual information from words that comes before it. Here the LSTM is bidirectional, this way each word is aware of it's surroundings in both forward and backward directions. **To get performance advantage, we decided** to use bidirectional Gated Recurrent Unit cell (GRU) instead of LSTM. The output of this layer are 2 matrices, one for context and another for query.

3.4 BiDAF Attention

We have not made any changes in the BiDAFAttention class. The main function of this layer is creation of similarity matrix S, a tall matrix of dimension (TxJ), corresponding to applying similarity function to each column of the context and each column of query. The similarity matrix, S, serves as input to Bidirectional attention which computes attention in two directions - the context attends to the query (context-to-query) and the query attends to the context (query-to-context). The goal of context-to-query step is to find out which Query words are most relevant to each Context word.

3.5 Self Attention

BiDAF attention module deals with question-aware context representation. The problem of this representation is that it's highly focused on the relation between context and question, but has limited knowledge of the context itself, especially between parts of context. R-Net[5] inspired self-match attention mechanism deals with context to context representation. **We adopted** the idea of self-matching attention from R-Net where it directly matches the question-aware passage representation against itself. It dynamically collects evidence from the whole passage for words in passage and encodes the evidence relevant to the current passage word and its matching question information into the passage representation. **As per R-Net, our self-matching similarity** has

c_t = att(v^P, v_t^P), the attention pooling vector of the whole passage (v^P) and

h_t^P = BiGRU($h_{t-1}^P, [v_t^P, c_t]$)

(We have used GRU[6] instead of RNN as in the original paper, because GRU is computationally cheaper without any loss of performance)

We also consulted few other papers on self-attention[7] to implement Simple and Effective Multi-Paragraph Reading Comprehension techniques using self-matching attention. We also took ideas from the implementation: attention-is-all-you-need-pytorch [8].

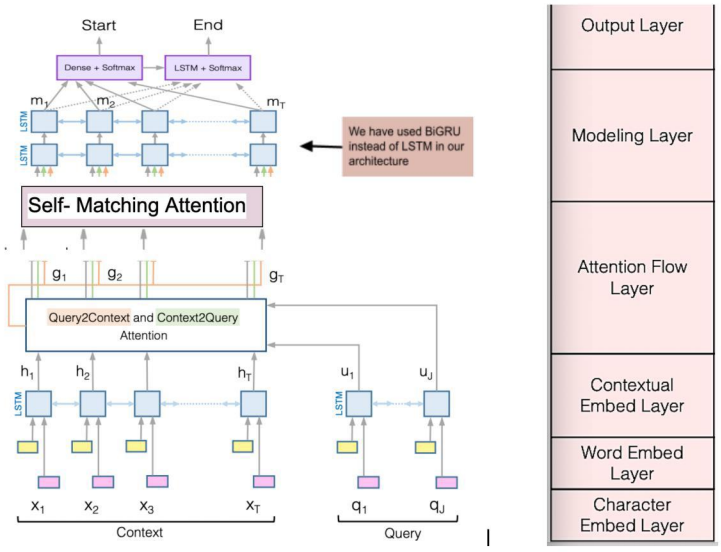

Figure 3: Our Modified BiDAF architecture

4 QANet approach

In QANet, given a context C of length n, C = $\{c_1, ..., c_n\}$ and a query Q of length m, Q=$\{q_1, ..., q_m\}$, the output of span S = $\{c_i, c_{i+1}, ..., c_{i+j}\}$ from the original paragraph C is called answer. QANet model contains five components: an embedding layer, an embedding encoder layer, a context-query attention layer, a model encoder layer and an output layer, as shown in Figure 4. For both the embedding and modeling encoders, QANet only uses convolutional and self-attention mechanism in place of RNNs, which is used by BiDAF model. QANet model is much faster than BiDAF, as it can process the input tokens in parallel.
We have summarized below our changes to QANet:

4.1 Input Embedding Layer

The embedding of each word w is obtained by concatenating its word embedding, $p_1 = 300$-Dim pretrained Glove word vectors (fixed during training) and character embedding. All the out-of-vocabulary words are mapped to an $< UNK >$ token trainable with random initialization. Each character in character embedding is represented as a trainable vector of dimension $p_2 = 200$. In QANet, unlike BiDAF, **we decided** that character embedding should use 2D-CNN to find numeric representation of words by checking their char-level compositions. 2D-CNN has more parameters than 1D-CNN and handles more features. **We initialized** the 2D-CNN with kaiming normal and used relu activation.

Hence, each word is a concatenation of the embedding vectors for each of its characters. The length of each word is either truncated or padded to 16. The output of a given word x from this layer is the concatenation $[x_w; x_c]$ $\epsilon \mathrm{R}^{p1+p2}$, where x_w and x_c are the word embedding and the convolution output of character embedding of x respectively. The concatenated embedding of word and char representation is passed thru 1D-CNN before passed on to a two-layer highway network which is on top of this representation and x is output of this layer.

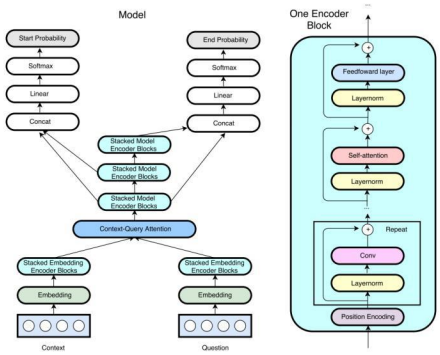

Figure 4: QANet framework.

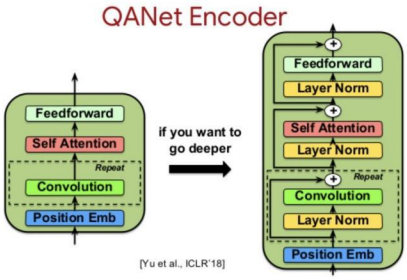

Figure 5: QANet Encoder.

4.2 Embedding Encoder Layer

Ref Figure 5, this layer is a stack of
$[convolution - layer + self - attention - layer + feed - forward - layer]$.
We decided to use depthwise separable convolutions for the convolution block similar to [16] Xception. The depthwise separable convolution is made of two operations: a depthwise convolution and a pointwise convolution. The computational cost of the depthwise separable convolution is the sum of the costs of the depthwise and pointwise operations. Compared to a normal convolution, it offers a computation reduction.
The input of this layer is a vector of dimension $p_1 + p_2 = 500$ for each individual word, which is immediately mapped to kernel=7, filters d = 128 one-dimensional convolution layer and there are 3

of them in the block. The output of this layer is a also of dimension d = 128.
We used Multi-head attention mechanism in the self-attention layer. In this layer, for each position of the query, we did a weighted sum of all the positions in the input based on similarity between the query and the key as measured by the dot product. Each layer of the stack
$[convolution - layer + self - attention - layer + feed - forward - layer]$
is placed inside a residual block.

4.3 Highway

As explained in BiDAF's Highway Network, in QANet we used 1D-convolution for the network. This worked without any issues.

4.4 MultiHeadAttention

We closely followed homework4 and implemented multi-head attention.
Mathematically, self attention for input matrix (Q,K,V) is
$Attention(Q, K, V) = softmax(QK^T/\sqrt{d_k})$, where d_k is the dimension of the key, ref: figure 6. Q,K,V are essentially same and the attention is performed multiple times, then the output heads are concatenated as the final output hidden state h. We took the picture from transformer, we have not implemented scaled Dot-Product attention as shown in the diagram, figure 7.

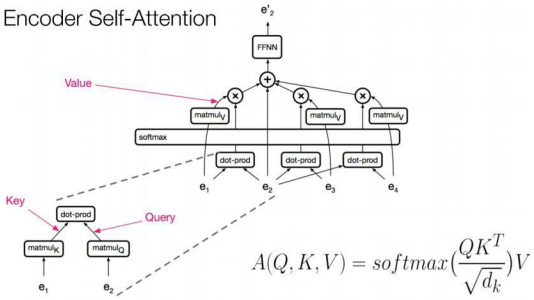

Figure 6: Encoder SelfAttention.

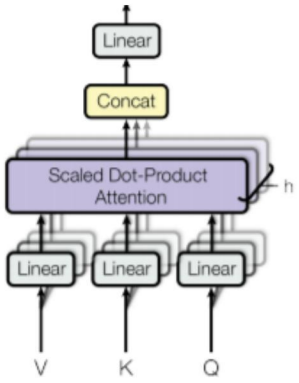

Figure 7: Multihead Attention.

4.5 Context Query Attention Layer

Ref fig.4, A similarity between context C ϵR^n and query QϵR^m is calculated, the resulting similarity matrix is, SϵR^{n+m}. Each row of S is normalized by applyng softmax function, resulting matrix is $S^{'}$. The context-query attention is computed as
$A =S^{'} * Q^T \epsilon R^{n*d}$. To compute the query-context attention a column normalization matrix, $S^{''}$ of S is computed by softmax. The resulting matrix is
$B = S^{'} * S^{''T} * C^T$.

4.6 Model Encoder Layer

Ref fig.5, Input at each position is $[c, a, c * a, c * b]$ where a and b are the row of the attention matrix A and B respectively. The layer parameters are the same as the Embedding Encoding layer and **We have reused** the model Encoding layer mostly, **except that the number of convolution layers is 2** within a block and the total number of blocks are 7.

4.7 Output Layer

Ref fig.4, SQuAD is labeled with a span in the context containing the answer. **We have not** changed this layer. The probability of each position being start and end of an answer is calculated as:

$p^1 = softmax(W_1[M_0, M_1]$
$p^2 = softmax(W_2[M_0, M_2]$
where W_1, W_2 are trainable variables and M_0, M_1, M_2 are the output of the model encoders from bottom to top. The score of a span is the product of its start and end position probabilities. Finally, the objective function is defined as the negative sum of the log probabilities of the predicted distributions indexed by true start and end indices, averaged over all the training examples:
$L(\theta) = -(1/N)\Sigma_i^N[log(p^1_{y_i^1}) + log(p^2_{y_i^2})]$,
where y_i^1 and y_i^2 are the ground truth of example i at the starting and ending position respectively , and θ contains all the trainable variables. The proposed model can be customized to other comprehension tasks, e.g. selecting from the candidate answers, by changing the output layers accordingly.

5 Experiments

5.1 Data

Stanford Question Answering Dataset (SQuAD) 2.0 [10] (Rajpurkar et al. 2016) is a reading comprehension dataset consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answers to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswered. The figure 3 and 4 below show the distribution of different types of questions in SQuAD dataset.

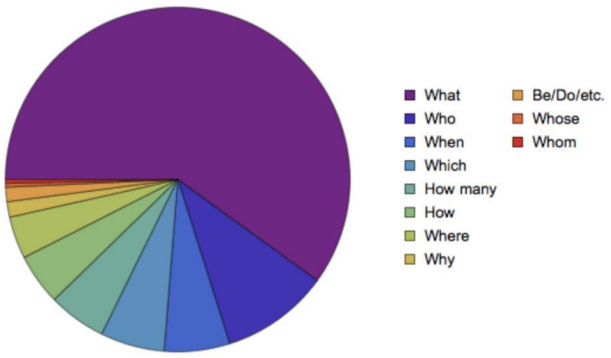

Figure 8: SQuAD dataset question types

What	85301
Who	14412
When	8720
Which	8576
How many	7728
How	6832
Where	5635
Why	2058
Be/Do/etc.	1926
Whose	510
Whom	494

Figure 9: SQuAD distribution

SQuAD 2.0 has about 150k questions, unanswerable questions written adversarially by crowdworkers to look similar to answerable ones. The typical length of a paragraph is 250 and question is 10 tokens. SQuAD 2.0 is divided into train/dev and test sets.

5.2 Evaluation method

Experiments are evaluated on F1 score and Exact Match (EM) for the performance on the SQuAD 2.0 dataset. **For BiDAF**, we iterated our experiments on SQuAD 2.0 using a batch size of 64 on NVIDIA NV6 GPU (6 vcpus, 56 GiB memory). **For QANet**, we reduced our batch size to 16 in order to iterate on NVIDIA NV6 GPU and it took significantly longer due to the reduced batch size.

5.3 Experimental details

We used pretrained GloVe vectors [11] 300- dimensional embeddings trained on the 840B Common Crawl corpus. We experimented with a wide range of values for the hyperparameters: hidden state size, learning rate, training time, decay rate and seed values. It was noticed in general, that model performed very poorly for small values of learning rate. By lowering the decay rate, performance degraded. Reduction of the dropout [12] probabilities and increase of hidden layers helped to improve model performance. Our loss function was the sum of the negative loglikelihood (crossentropy) loss

for the start and end locations of the answer. During training, we averaged across the batch and used the Adadelta [13] optimizer to minimize the loss.
We attempted few variation of self-attention and self-matching attention. One attempt was to pass the output of self-attention to Modeling layer to refine the context vector further. Another attempt was to add self-attention after Modeling layer, so the input was the refined context representation after attention layer. Lastly, the self-matching attention per R-Net implementation, as explained earlier in the 'Approach' section. The self-matching attention implementation provided the best model. Our BiDAF and QANet implementation results are presented below.

5.4 Results

Model(Baseline BiDAF + modification)	Training Time	F1	EM
1.Baseline (lr=.5 , decay_rate = .999, drop_prob=.2, seed=224, hidden_size =100)	10h 19m 4s	60.45	57.30
2.Baseline + Char Embedding + Self Attention+GRU (lr =.5 , decay_rate =.999, drop_prob=.2, seed=224, hidden_size =100)	18h 55m 20s	66.37	62.81
3.Baseline + Char Embedding + Self Attention+GRU (lr =.3 , decay_rate =.997, drop_prob=.3, seed=2321, hidden_size =100)	21h 6m 55s	63.11	60.17
4.Baseline + Char Embedding + Self Attention+GRU (lr =.6 , decay_rate =.9999, drop_prob=.2, seed=224, hidden_size =110)	22h 32m 47s	66.26	62.73

Figure 10: BiDAF experiments

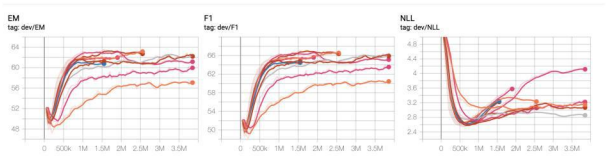

Figure 11: BiDAF results comparison

```
Loading checkpoint: /home/minasm/suvasis/tools/stanford/checkpoints1/model_best.pth.tar ...
Checkpoint '/home/minasm/suvasis/tools/stanford/checkpoints1/model_best.pth.tar' (epoch 25) (best f1 74.11414733
807817) (best em 61.45) (step 406026 )  loaded
```

Figure 12: QANet experiments

Figure 13: QANet parameters

- **F1 and EM scores on the dev and test leaderboard**
  Since, we were able to run only one iteration of QANet due to batch size = 16 and limited compute, we did not put our QANet results to test leaderboard because the EM value in QANet was lower than the EM value in our BiDAF best model.However, QANet result showed a lot of promises for future improvements.

Dev LeaderBoard Model details (BiDAF Modification)			
Baseline + Char Embedding + Self Attention+GRU (lr =.5 , decay_rate =.9999, char-embedding drop_prob=.05, drop_prob elsewhere = 0.1,seed=224, hidden_size =200)	26h 48m 40s	67.08	63.22

Figure 14: Validation Leaderboard

Test Leaderboard Model details (BiDAF Modification)			
Baseline + Char Embedding + Self Attention+GRU (lr =.7 , decay_rate =.9999, drop_prob=.1 everywhere, seed=224, hidden_size =200)		65.88	62.09

Figure 15: Test Leaderboard

6 Analysis

- **Quantitative error analysis**
  We performed error analysis by looking at the answerability of questions. The baseline model performed quite poorly on the unanswerable questions. After adding character embedding to BiDAF , it improved the performance of both question types approximately equally. By adding a few fundamental changes to BiDAF, like self-matching attention and GRU, we enhanced the model performance both from metrics perspective as well as computationally as it improved the unanswerable performance significantly across the entire dev set, without sacrificing the performance of answerable questions. One possible explanation for the slightly worse performance on answerable questions might be that the baseline model is

biased for predicting an answer on border cases.
We also compared the performance of our model with baseline model by question type and noticed that our model shows improvement across different question types. Even though there are majority of 'what' questions in the set, we observed the most improvement in 'who' questions.

- **Qualitative error analysis**
  Here are two examples of 'what' and 'who' questions where our model understood the context really well and also applied relevant reasoning even though it was not a direct question.

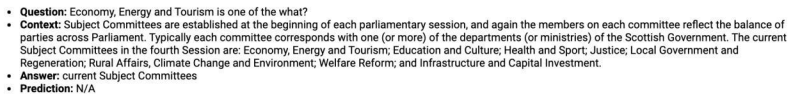

Figure 16: What question

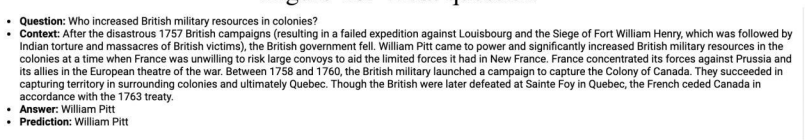

Figure 17: Who question

- **Qualitative Evaluation**
  Here are two instances where our model failed to predict and our model predicted the output wrong.

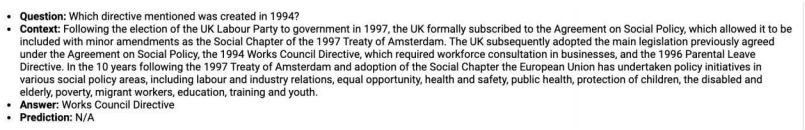

Figure 18: Failed to predict

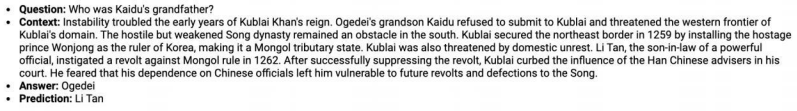

Figure 19: Predicted wrong

Ref Fig 18: By inspecting the key characteristics of the context it looks like, there was no direct match of the word 'created' in the answer span, hence the p_{start} and p_{end} indices could not be found. We can definitely look into these edge cases and improve our embedding and attention mechanism further.
Ref Fig 19: This is not a direct question and it requires some understanding in capturing relationships like grandfather, son-in-law etc. So, embedding using dependency parser might improve these use cases.

7 Conclusion

We have experimented with two completely different approaches, one using BiDAF and one using QANet. Adding character-level embedding to the baseline model, and then implementing self-matching attention layer, BiDAF performance improved. We also re-implemented QANet to check the design advantages it has over BiDAF. We demonstrated that just by using QANet attention ideas in BiDAF framework and by getting rid of RNN layers the model performance improved 20% for F1 score but the EM did not perform as well. Since QANet model works differently, we require more time to understand the model and higher compute resources to iterate QANet model in order to achieve better performance.

References

[1] Minjoon Seo1, Aniruddha Kembhavi, Ali Farhadi, Hananneh Hajishirzi. Bi-Directional Attention Flow for Machine Comprehension
[2] Minh-Thang Luong Hieu Pham Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation
[3] Adams Wei Yu1, David Dohan2, Minh-Thang Luong. QANet: Combining Local Convolution with global self-attention
[4] Rupesh Kumar Srivastava, Klaus Greff, Jurgen Schmidhuber. Highway Networks
[5] Natural Language Computing Group, Microsoft Research Asia. R-NET: Machine Learning Comprehension with Self-matching Networks.
[6] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, Ming Zhou. Gated Self-Matching Networks for Reading Comprehension and Question Answering
[7] Christopher Clark, Matt Gardner. Simple and Effective Multi-Paragraph Reading Comprehension
[8] https://github.com/jadore801120/attention-is-all-you-need-pytorch
[9] https://ai.stanford.edu/ amaas/papers/relu_hybrid_icml2013_final.pdf
[10] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. In Empirical Methods in Natural Language Processing (EMNLP), 2016
[11] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In EMNLP, volume 14, pp. 1532–43, 2014
[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting
[13] Matthew D. Zeiler. Adadelta: An Adaptive Learning Rate Method 2012
[14] https://github.com/allenai/bi-att-flow
[15] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. arXiv preprint arXiv:1611.01604, 2016
[16] https://keras.io/api/applications/xception/