Question Answering with Various Attention Mechanisms

Yinghao Sun
Stanford University
Stanford, CA 94305
sunmo@stanford.edu

Abstract

In this CS224N Winter 2018 final project, several variations of encoder, attention, and output layer are implemented for an end-to-end question answering system. Bidirectional attention flow (BiDAF) outperforms other models on the Stanford question answering dataset, with a single BiDAF model achieving a dev F1 of 74.37% and a test F1 of 74.72%, and an ensemble achieving a test F1 of 76.02%.

1 Introduction

Reading comprehension is a daily task for human beings and one of the fundamental ways to communicate. Machine Reading Comprehension (MRC) researches aim to build a intelligence system that can comprehend written language, which can see wide applications in many real-world scenarios. One crucial, extrinsic task to evaluate MRC systems is through question answering (QA), which is a real-world task that requires both natural language understanding and real world knowledge.

QA tasks could be formulated in different ways, for example, in MCTest [1] each task is a tuple of context, question, and candidate answers. The goal is to identify the correct answer from the given candidates. In the present project, we will implement our models for the Stanford Question Answering Dataset (SQuAD) [2], where each task is a tuple of $\langle c, q, a \rangle$. Given the context c and a query q, the system needs to identify a span in c that constitutes an answer a for q.

The present project is an attempt to build an end-to-end neural system for the QA tasks in SQuAD. Several variations are implemented and bi-directional attention flow was found to give the best performance on dev set. The system will be consisted of three major modules: encoder that learns the representation of context and query, an attention mechanism that results in refined, query-aware context representations, and an output layer that predicts the start and the end position of the answer span. I have experimented with several variations for these three major layers.

2 Related Work

In Jan 2018, the best model has already achieved an F1 of 89.281% and an Exact Match (EM) of 82.482% on SQuAD which was released by Rajpurkar et al [2] in 2016. Below I will summarize some major advancements in terms of the three major layers, i.e., encoder, attention, and output layer.

Encoder Layer This is the layer to learn context and question representations. Typically, we first start with word embedding, e.g., Glove [3, 4, 5], or concatenated with character embeddings [6, 7], which helps to address out-of-vocabulary (OOV) words. Other features may also be helpful, e.g., part-of-speech tags, named entity recognition tags, etc. [5], or even structural embedding of syntactic trees [8]. These embeddings are then passed to recurrent neural network (RNN) [5] or multilayer Highway Network [7, 9].

Attention Mechanism Attention is crucial in generating query-aware context representations. There are variations of attention mechanism, e.g., bi-directional attention flow (BiDAF) [7], Co-attention [4], self-matching in R-Net [10], fine-grained gating mechanism that dynamically combine word-level and character-level representations [11], gated-attention mechanism [12], bilateral multi-perspective matching [13].

Output Layer Output layer predicts the answer span, typically through softmax. In the one hop approach, predictions are made once, e.g., BiDAF [7], Match-LSTM and Answer Pointer Network [14], or dynamic chunk reader [15]. In the multi-hop approach, predictions of the start position and the end position are iterated several times, e.g., [4, 9, 16], and this helps the model to move away from local minima.

Recent advancements have also incorporated more sophisticated modules, for example, a "memory-like" module as in reinforced mnemonic reader [17].

3 Architecture

In this project, several variations of model architecture has been implemented, and BiDAF [7] is found to have the best performance on dev set. I will start with a description of BiDAF and then other variations. Note that baseline from default project handout is also included in the experiment but not described in details here.

3.1 Encoder Layer

GloVe embeddings $\{\boldsymbol{x}_i\}_{i=1}^N, \boldsymbol{x}_i \in \mathcal{R}^d$ for context and $\{\boldsymbol{y}_j\}_{j=1}^M, \boldsymbol{y}_j \in \mathcal{R}^d$ for questions are first passed to a bidirectional RNN to obtain hidden states for contexts $\boldsymbol{c}'s$ and questions $\boldsymbol{q}'s$. RNN helps to capture interaction between words in the paragraph. In Baseline, the RNN is a single layer bidirectional GRU while for BiDAF, I chose two-layer bidirectional LSTM.

3.2 Attention Layer

Attention layer refines the hidden states of context and questions to allow them to be aware of each other. Baseline has implemented a simple attention mechanism.

3.2.1 BiDAF [7]

First, a similarity matrix $\boldsymbol{S} \in \mathcal{R}^{N \times M}$ is computed, where each $\boldsymbol{S}_{ij}$ is a similarity score for each pair $(\boldsymbol{c}_i, \boldsymbol{q}_j)$:

$$\boldsymbol{S}_{ij} = \boldsymbol{w}_{\text{sym}}^T [\boldsymbol{c}_i, \boldsymbol{q}_j, \boldsymbol{c}_i \circ \boldsymbol{q}_j] \in \mathcal{R}$$

where $\boldsymbol{w}_{\text{sym}}$ is a trainable weight vector.

Next, context-to-question attention is computed as $\boldsymbol{a}_i = \sum_{j=1}^M \alpha_j^i \boldsymbol{q}_j \in \mathcal{R}^{2h}$, where $\alpha^i = \text{softmax}(\boldsymbol{S}_{i,:}) \in \mathcal{R}^M$. Question-to-context attention is computed as $\boldsymbol{c}' = \sum_{i=1}^N \beta_i \boldsymbol{c}_i \in \mathcal{R}^{2h}$, where $\beta = \text{softmax}(\boldsymbol{m}) \in \mathcal{R}^N$ and $\boldsymbol{m}_i = \max_j \boldsymbol{S}_{ij} \in \mathcal{R}$.

The final learned representation for each context token i is

$$\boldsymbol{b}_i = [\boldsymbol{c}_i; \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{c}'] \in \mathcal{R}^{8h} \qquad \forall i \in \{1, \ldots, N\}$$

where $\boldsymbol{a}_i$ and $\boldsymbol{c}_i \circ \boldsymbol{a}_i$ incorporate question-aware context representation, and $\boldsymbol{c}_i \circ \boldsymbol{c}'$ captures the information of interaction of context and context-aware question representations.

3.2.2 Coattention [4]

The implementation is based on the default project handout. The key is to compute context-to-question attention $\{\boldsymbol{a}_i\}_{i=1}^N$ for each context token i and question-to-context attention $\{\boldsymbol{d}_i\}_{j=1}^M$ for each question token j. On top of this, compute second-level attention as $\boldsymbol{s}_i = \sum_{j=1}^{M+1} \alpha_j^i \boldsymbol{d}_j \in \mathcal{R}^{2h}$, for each context token i. Concatenation of first-level and second-level attention outputs i.e., $\{[\boldsymbol{s}_i; \boldsymbol{a}_i]\}_{i=1}^N$ are then passed to a biLSTM to obtain the attention output $\{\boldsymbol{u}_i\}_{i=1}^N$, which is then concatenated with context hidden states $\{\boldsymbol{c}_i\}_{i=1}^N$.

3.2.3 Self-attention [10]

Self-attention is another self-matching attention layer that could be applied on top of other attention outputs. Given a sequence of attention outputs $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N \in \mathcal{R}^l$, the attention output is

$$\{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N\} = \text{biRNN}(\{[\boldsymbol{v}_1; \boldsymbol{a}_1], \ldots, [\boldsymbol{v}_N; \boldsymbol{a}_N]\})$$

where $\{\boldsymbol{a}_i\}_{i=1}^N$ are self-matching (i.e., between $\boldsymbol{v}$'s) attention scores.

3.3 Output Layer

This is the layer where we generate the final predictions for the start and the end position for the answer span. Baseline has a simple softmax layer based on a fully-connected layer.

3.3.1 BiDAF Modeling and Output Layer

In the notation of BiDAF paper [7], the blended representations $\{\boldsymbol{b}_i\}_{i=1}^N$ is denoted as $\boldsymbol{G}$, which are passed to a two-layer biLSTM to obtain $\boldsymbol{M} \in \mathcal{R}^{2d \times N}$. Then the start position is predicted as

$$\boldsymbol{p}^1 = \text{softmax}(\boldsymbol{w}_{\boldsymbol{p}^1}^T [\boldsymbol{G}; \boldsymbol{M}])$$

To predict the end position, first pass $\boldsymbol{M}$ to another single layer biLSTM to obtain $\boldsymbol{M}^2 \in \mathcal{R}^{2d \times N}$, then end position is predicted as

$$\boldsymbol{p}^2 = \text{softmax}(\boldsymbol{w}_{\boldsymbol{p}^1}^T [\boldsymbol{G}; \boldsymbol{M}^2])$$

3.3.2 Answer Pointer Layer

The main idea is to condition end position prediction on the start position prediction. This implementation is similar but slightly modified to that in R-Net [10] and Match-LSTM [14]. First, the probability of each context token i being the start position is predicted as

$$\boldsymbol{p}_i^1 = \text{softmax}(\boldsymbol{v}^T \tanh(\boldsymbol{W_b} \boldsymbol{b}_i + \boldsymbol{W_h} \boldsymbol{h}_0))$$

and the probability of each context token i being the end position is predicted as

$$\boldsymbol{p}_i^2 = \text{softmax}(\boldsymbol{v}^T \tanh(\boldsymbol{W_b} \boldsymbol{b}_i + \boldsymbol{W_h} \boldsymbol{h}_1))$$

where $\boldsymbol{h}_0$ is initialized as $\boldsymbol{h}_0 = \sum_{j=1}^M \beta_j \boldsymbol{q}_j$ and $\beta_j = \text{softmax}(\boldsymbol{v_q}^T \tanh(\boldsymbol{W_q} \boldsymbol{q}_j + \boldsymbol{b_q}))$. On the other hand, $\boldsymbol{h}_1 = \text{RNN}(\boldsymbol{h}_0, \sum_{i=1}^N \boldsymbol{p}_i^1 \boldsymbol{b}_i)$, where $\boldsymbol{b}$ is the blended representation.

3.3.3 Other Variations

In self-attention and co-attention, we could choose multilayer biLSTM instead of a single layer one, or we could stack two fully-connected layer instead of one in the output layer.

3.3.4 Adjustment

Since in the training set, about 98.98% of context-question pairs have answer span less than 20, the start index l^{start} and the end index l^{end} are predicted as the pair (i, j) with $i \leq j \leq i + 19$ that maximizes $p_i^{\text{start}} p_j^{\text{end}}$. This constraint gives an improvement of 2-5 points in F1.

4 Experiments

I have experimented with several variations of different layers. I did not have the resource/time to fully examine each possible combination, and the goal is mainly to search for the architecture and hyperparameters that work the best, which turns out to be BiDAF based on my implementation. I will focus on BiDAF first and then present results on other variations.

4.1 Dataset

Models are evaluated on SQuAD v1.1, which has 23,215 paragraphs from 536 articles on Wikipedia covering a wide range of topics. In total there are 107,785 questions. The dataset has been split into a 80% training set, a 10% dev set, and a 10% hidden test set.

4.2 Implementation Details

For BiDAF, hidden size is 120 and dropout is 0.20, which is applied to all RNN layers and fully connected layers. Word embedding is GloVe 100d, batch size is 100, context length is 320 since in training set 98.88% of context-question pairs' context length are less than 320, and question length is 30, which covers 99.93% cases in the training set. Optimizer is Adam with learning rate of 0.001, and the model was trained for 16.5k iterations. Hyperparameters are tuned separately for other variations of model architectures but shared hyperparameters are in general similar to BiDAF, e.g., context length, question length, batch size, and training iterations.

4.3 Model Evaluation

Exact Match. A binary measure of whether the system output matches the ground truth answer exactly. Taken to be the maximum across three human answers, and then aggregated across all document-question pairs.

F1 score. The harmonic mean of precision and recall, and taken to be the maximum across three human answers, and then aggregated across all document-question pairs.

4.4 Results

This section starts with an overall comparison among different variations, and then a deep dive into BiDAF and other model architectures. **Detailed analysis of loss examples are also included in the supplementary materials**.

4.4.1 BiDAF

Results Table 1 presents the performance comparison across main model architectures and variations implemented in this project. Each variation has been tuned separately.

The main observations are that post hoc adjustment (i.e., imposing the constraint start_pos $\leq$ end_pos $\leq$ start_pos $+$ 19) has significantly boosted the performance for baseline and for other model variations. Replacing baseline output layer with answer pointer network does not appear to improve the performance that much, and will be discussed in more details later. Self-attention on top of baseline brings huge improvement in F1 and EM. Combing self-attention and coattention gives an even better performance than baseline plus self-attention. On the other hand, a better tuned coattention outperforms the hybrid of self-attention and coattention. These variations will be discussed in more details later.

Overall, BiDAF outperforms other variations based on my specific implementation, with a single model achieving 74.72% test F1, and an ensemble achieving 76.02% test F1. In the following section, we will take a closer look at BiDAF.

Table 1: Results on the SQuAD dev and test set

	Dev Set		**Test Set**	
	EM	F1	EM	F1
Baseline w/o any start_position $\leq$ end_position adjustment	34.68	43.69	-	-
Baseline	40.41	51.31	-	-
Baseline + Answer Pointer Network	41.11	51.72	-	-
Baseline + Self-Attention	53.93	65.42	-	-
Coattention + Self-Attention	56.52	68.00	-	-
Coattention + 3-layer biLSTM in encoder layer	58.39	69.70	-	-
BiDAF (single)	**64.27**	**74.37**	**64.51**	**74.72**
BiDAF (ensemble)	**64.84**	**74.93**	**66.27**	**76.02**

Effect of Dropout The effect of different dropout probabilities are explored. As shown in Figure 1, dropout probability has an direct effect on the training F1, that is, with lower dropout probability, the training F1 is higher. On the other hand, dropout is effective in preventing overfitting: when

there is no dropout, the model quickly overfits at around 4k iterations and dev F1 starts to drop after that. On the other hand, the difference in a dropout probability of 0.15 or 0.20 does not appear to be very notable. Note that F1 in Figure 1 is based on tensorboard and hence slightly different from the final dev F1.

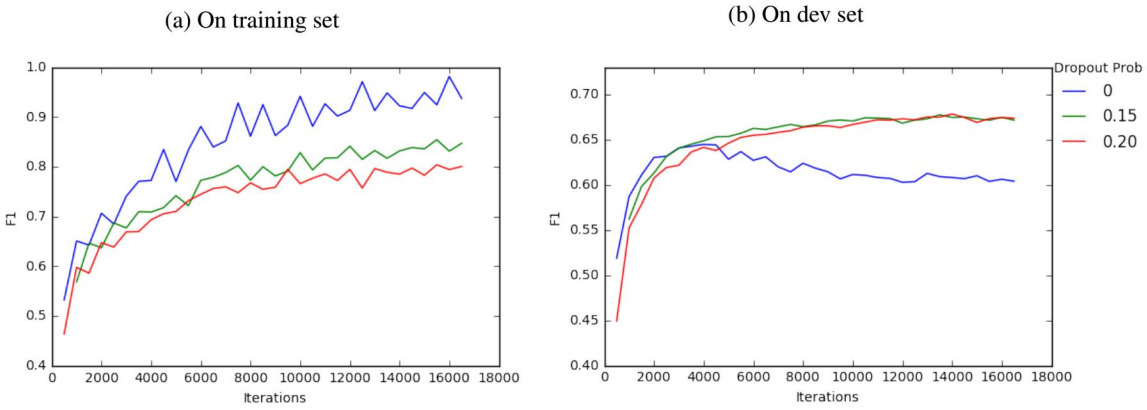

Figure 1: BiDAF with different dropout probabilities

Ablations The impact of different modules in BiDAF are explored; however, the experiments are by no means comprehensive due to long model training time.

By no proper bidirectional attention, it means implementing the attention in BiDAF as $\boldsymbol{b}_i = [\boldsymbol{c}_i; \boldsymbol{a}_i; \boldsymbol{c}']$ rather than $\boldsymbol{b}_i = [\boldsymbol{c}_i; \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{a}_i; \boldsymbol{c}_i \circ \boldsymbol{c}']$. By no modeling layer, it means using the baseline output layer rather than the modeling layer in BiDAF which involves multilayer RNN.

As shown in Table 2, modeling layer plays a critical role in the system's performance. Even when attentions are implemented properly, the system performance will be suboptimal without the modeling layer. One intuition is that the modeling layer allows to capture more complicated patterns of interactions via applying multilayer RNN on the blended representation (i.e., concatenation of context hidden states and attention outputs). An extra layer of RNN when predicting the end position also gives the model extra complexity to adjust its start and end position predictions.

On the other hand, the 'proper' bidirectional attention is critical since it combines the information from context hidden states and context-to-question attentions, as well as capturing the interaction between context and context-aware question representations.

Table 2: Ablations of BiDAF on the dev set

	Dev Set	
	EM	F1
No proper bidirectional attention, no modeling layer	41.93 (-22.34)	53.21 (-21.16)
No modeling layer	44.21 (-20.06)	55.87 (-18.50)
BiDAF	64.27	74.37

Other Hyperparameter Tuning Tuning of other hyperparameters is also attempted (e.g., different hidden size, different word embedding dimensionality, or different number of RNN layers for the encoder), but overall the change in performance is less notable, except that some variations may slow down the training process due to more parameters, or even cause GPU memory issues.

Ensemble We've tested an ensemble of five BiDAF models with slightly different hyperparameters. The ensemble is based on majority voting and it gives slight boost in performance. Ideally, if time allows, an ensemble of more BiDAF with different runs should work better.

Performance by Context/Question Type In this section, BiDAF performance are examined on slices of different document-question-answer attributes. As shown in Figure 2, F1 and EM tend to be lower for context length longer than 320, which is the maximum context length in our model. On

the other hand, F1 is relatively stable across different question length and there are only a few data points beyond question longer than 30 words.

Answer length is calculated as the average answer length of three ground truth answers. For answer length, the system performance appears to be worse for longer answers, which could be more complicated and hard to predict accurately. In addition, beyond answer length of 20, f1 and exact match are either zero or close to zero, which is expected due to the constraint on the answer length. Note that there is an outlier having 100% EM for an average answer length of 23, but this is possible since for this question, the three ground truth answers are of length 16, 26, 28, hence not all of them are greater than the cut-off 20.

Figure 2 shows that BiDAF predicts the most accurate answer spans for 'when' questions, and performs the worst on 'why' questions, which could involve more logical inductions. Another plausible reason is that we have less 'why' questions. It will be discussed in more details in the error analysis section and in the **error analysis supplementary materials**.

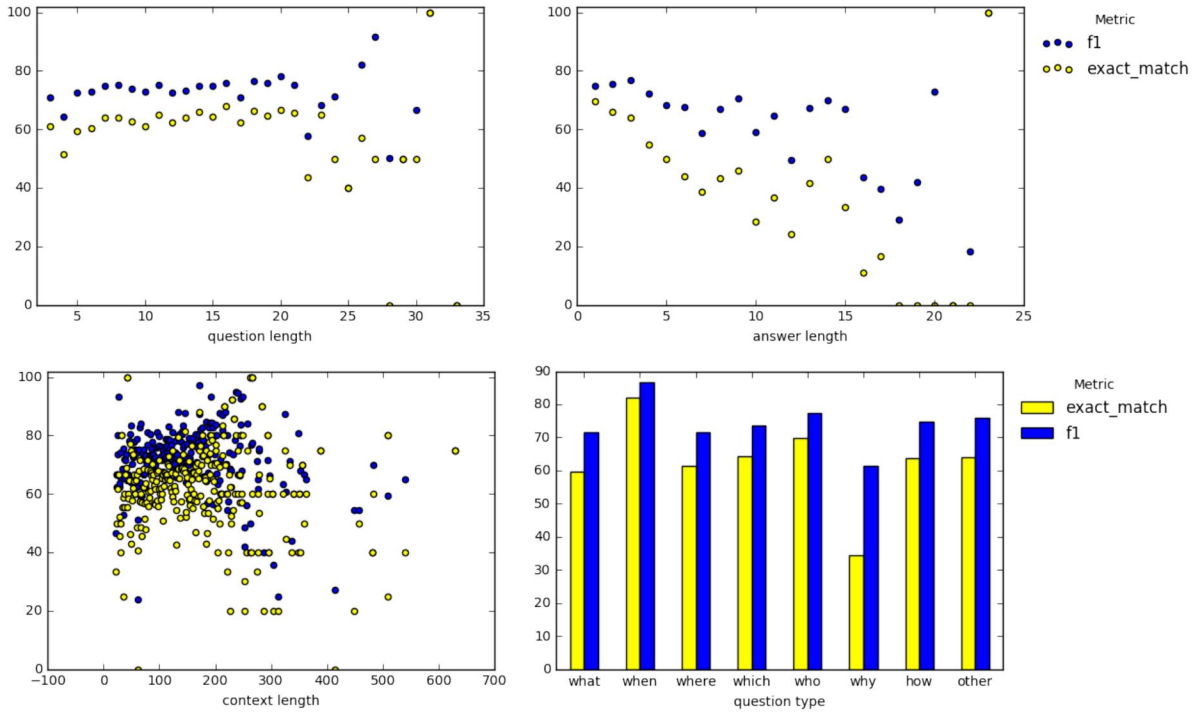

Figure 2: BiDAF F1/EM by context length, question length, answer length, and question type

Error Analysis More details of the error analysis and loss examples can be found in **the error analysis supplementary materials**. The main observation in looking through loss examples are that 1) the system in general is able to identify a span in the context paragraphs that is at least close to where the correct answer is, which indicates that the attention mechanism is effective; 2) even though the system may identify a span that is close to the true answer, it fails to be capable of performing logical inductions: for example, when the relationship between the question and the context requires logical induction but there are minimal overlapping words, it appears to be difficult for the system to predict the right answer; for example, the question asks for the team with 'most sacks' in super bowl and the context only mentions the number of sacks for each team, and the system has to infer what number of sacks is the 'most sacks'; 3) the answer produced by the system is sometimes informationally correct, but less readable compared with human answers due to missing a few words that would make the answer reads more smoothly.

4.4.2 Other Variations

In this section, we will discuss experiment results for model architecture variations other than BiDAF.

Answer Pointer Network Layer The key idea of answer pointer network is to condition the prediction of end position on the predicted start position. However, when replacing baseline output layer with my implementation of answer pointer network, it only improves very slightly upon baseline, while increasing the number of parameters and slowing down the performance. Hence, this is not pursued further given limited time frame for this project.

Variations of Self-Attention Self-attention is an extra layer applied to other attention outputs. When combined with baseline, it significantly boosts the performance. Since co-attention is performing better than baseline, I apply self-attention on top of co-attention, and the hybrid attention further boosts the performance as shown in Table 1. On the other hand, imposing self-attention on top of other attention mechanisms increase model size by a considerable amount, and frequently ran into GPU memory issues.

Variations of Co-Attention In co-attention, the attention output is the concatenation of forward and backward hidden states of the RNN, and hence, the attention output is two times the size of the hidden state. As shown in Figure 3, one major finding is that increasing the attention output size (or the hidden state size) may significantly boost the performance, which reaches a peak at round 1.2, and stays at a plateau after that. Note that increasing the hidden state size slows down the training and may cause GPU memory issues.

As in Table 3, other co-attention variations are experimented, e.g., replacing the single layer bidirectional GRU in encoder layer with multilayer bidirectional LSTM, or replacing the single layer bidirectional LSTM in the attention output with a two layer bidirectional LSTM, or adding another fully-connected layer in the output layer. However, these variations do not give any significant boosts in performance. With an extra fully-connected output layer, the performance is even worse but perhaps due to insufficient tuning of hyperparameters.

It is worth noting that I have not fully replicated co-attention model as in the original paper, for example, one major component missing is the dynamic part, i.e., the dynamic pointing decoder that involves iterative reasoning and could be effective in avoiding local minima,

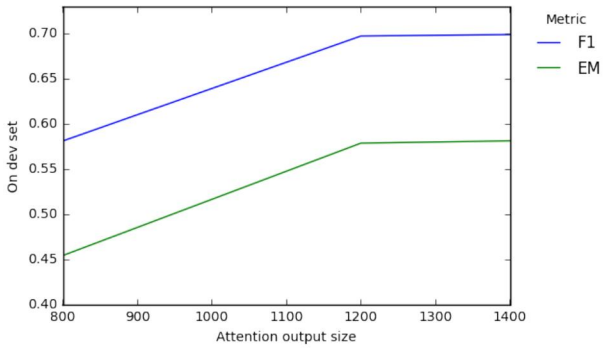

Figure 3: Co-Attention with different attention output size

Table 3: Results on different variations of co-attention

	Dev Set	
	EM	F1
Co-attention	58.09	69.85
Co-attention + 3-layer biLSTM for encoder	58.39	69.70
Co-attention + 2-layer biLSTM for attention output	56.59	68.17
Co-attention + 2 fully-connected output layer	42.71	54.09

5 Conclusion

In this project, based on the provided starter code, I have implemented several variations of end-to-end systems on the SQuAD tasks. BiDAF outperforms other variations implemented in this project, and a single model achieves 74.37% dev F1, 64.27% dev EM, 74.72% test F1, and 64.51% test EM, with ensemble achieving 76.02% test F1, and 66.27% test EM. The **detailed error analysis (included in the supplementary material)** shows that the system is in general able to identify a span in the context paragraph that is close to the correct answer, but may make errors when it requires more logical inductions to correctly answer the questions.

Disclaimer: even though I may use the same/similar model name as in their original papers, all of these names only refer to my own specific implementation, which are likely partial implementations rather than exact replications of the original papers, and if they do not perform well, it is likely my own fault.

References

[1] Matthew Richardson, Christopher JC Burges, and Erin Renshaw. Mctest: A challenge dataset for the open-domain machine comprehension of text. *In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 193-203. 2013.

[2] Pranav Rajpurkar , Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[3] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. *In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532-1543. 2014.

[4] Caiming Xiong, Victor Zhong, and Richard Socher. Dynamic coattention networks for question answering. *arXiv preprint arXiv:1611.01604*, 2016.

[5] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

[6] Zhilin Yang, Ruslan Salakhutdinov, and William Cohen. Multi-task cross-lingual sequence tagging from scratch. *arXiv preprint arXiv:1603.06270*, 2016.

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

[8] Rui Liu, Junjie Hu, Wei Wei, Zi Yang, and Eric Nyberg. Structural embedding of syntactic trees for machine comprehension. *arXiv preprint arXiv:1703.00572*, 2017.

[9] Rupesh Kumar Srivastava, Klaus Greff, and Jrgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[10] R-NET: Machine reading comprehension with self-matching networks. *https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf*.

[11] Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W. Cohen, and Ruslan Salakhutdinov. Words or characters? fine-grained gating for reading comprehension. *arXiv preprint arXiv:1611.01724*, 2016.

[12] Bhuwan Dhingra, Hanxiao Liu, Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Gated-attention readers for text comprehension. *arXiv preprint arXiv:1606.01549*, 2016.

[13] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. *arXiv preprint arXiv:1702.03814*, 2017.

[14] Shuohang Wang, and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*, 2016.

[15] Yang Yu, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. End-to-end answer chunk extraction and ranking for reading comprehension. *arXiv preprint arXiv:1610.09996*, 2016.

[16] Xiaodong Liu, Yelong Shen, Kevin Duh, and Jianfeng Gao. Stochastic Answer Networks for Machine Reading Comprehension. it arXiv preprint arXiv:1712.03556, 2017.

[17] Minghao Hu, Yuxing Peng, and Xipeng Qiu. *Reinforced mnemonic reader for machine comprehension. CoRR, abs/1705.02798*, 2017.