BiDAF Pro Max for Question Answering System

Stanford CS224N Default Project
Mentor: Kendrick Shen

Zhengguan Dai
Department of Aeronautics and Astronautics
Stanford University
`garydai@stanford.edu`

Qingyue Wei
Department of Electrical Engineering
Stanford University
`qywei@stanford.edu`

Yitao Qiu
Department of Civil and Environmental Engineering
Stanford University
`yitaoqiu@stanford.edu`

Abstract

As one of the ultimate goals of natural language processing, machine comprehension (MC) can be assessed by answering one or multiple questions with a chunk of text. In order to improve the performance of our model on SQuAD 2.0, we explore different embedding operations (character embeddings, token features), attention mechanism (iterative reattention) and output prediction structures (conditioning prediction) based on the baseline model, compared with QANet. Our results show that token features significantly improve the prediction by raising F1 score and EM score by >10; the iterative attention mechanism could further improve the model, achieving **F1=81.65** and **EM=77.89** on dev examples, **F1=76.63** and **EM=73.27** on test examples.

1 Introduction

Machine comprehension (MC) is one of the ultimate and most challenging goals of natural language processing (NLP). In the past few decades, MC has gained extensive attention from academia and industry, as it is a promising technology for application including search engine and dialog systems. MC can be assessed by answering one or multiple questions with a chunk of text, such as a news article or a short biography. Rajpurkar *et al.*(2016) introduced the Stanford Question Answering Dataset (SQuAD) that contains questions whose correct answers can be any sequence of tokens from the given passage [1].Since the release of SQuAD, RNN-based models such as BiDAF [2], Dynamic Coattention Networks [3], Match-LSTM and Answer Pointer[4], have been reported with significant improvements. Vaswani *et al.* (2017) proposed a novel architecture, so-called Transformer, that fully relies on self-attention to compute the representation of inputs and outputs without using RNN-like blocks.

Recently, Rajpurkar *et al.*(2018) updated the SQuAD by adding 50,000 unanswerable questions written adversarially, in addition to the previous 100,000 answerable questions [1]. To perform well on SQuAD, models need to first determine whether questions are answerable, then predict answers if possible.

In this work, besides the baseline model, we explore different embedding operations (character embeddings[2], token features[5]), attention mechanism (iterative reattention[6], coattention[3]) and output prediction structures (conditioning prediction[4]), compared with QANet[7]. Among all these approaches, we find character embedding, token features, and iterative reattention significantly improve the model performance.

2 Related Work

The BiDAF baseline of this project, proposed by Seo *et al.*(2016) in [8], is one of the papers that stood as the state of the art in MC in 2016. It combined ideas from many previous works and developed a memory-less static bi-directional attention flow architecture. The model is highly adaptive through its modulized design, where the last layer can be changed for various tasks. On SQuAD[1] question answering tasks, it achieved a ensemble-model F1 score of 81.1 and used ablation experiments to show that all components is beneficial to the final results.

Unlike the BiDAF model that predicts the start location and the end location independently, Wang & Jiang (2016) conditioned the end location probability distribution on the start location probability distribution [4]. The architecture they proposed consists of two parts: match-LSTM for textual entailment, and Pointer Net for the prediction of answer boundaries in the passage. Though the model out-performs the feature-engineered solution, questions with longer answers or starting with "why" are harder to predict.

Our work is also inspired by Chen *et al.* (2017)[5] and Hu *et al.* (2017) [6]. Chen *et al.* (2017) introduces additional features in paragraph encoding: exact match (EM), part-of-speech (POS), named entity recognition (ENT) tags, and term frequency (TF). Hu *et al.* (2017) proposed a reattention mechanism by memorizing past attentions and using them to enhance current attention along with the query information.

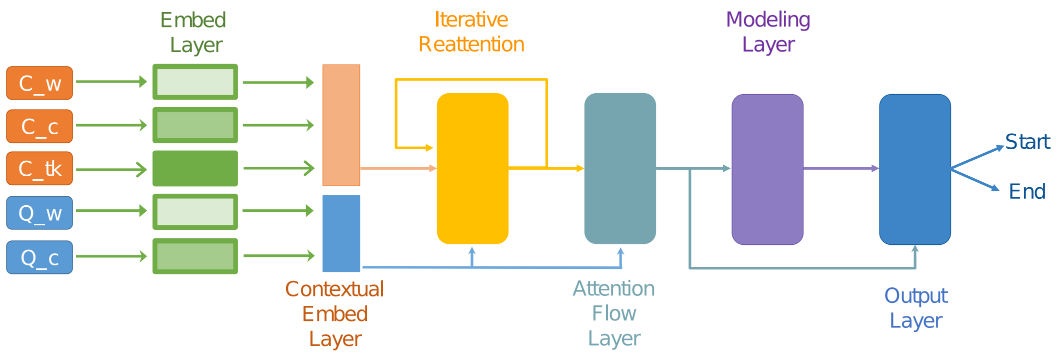


Figure 1: The architecture overview of BiDAF Pro Max (BPM).

3 Approach

We propose BiDAF Pro Max (BPM) for machine reading comprehension tasks. As shown in Fig. 1, BPM consists of six components including 1) Embed Layer for word/character/token feature embedding, 2) Contextual Embed Layer for word embedding refinement, 3) Iterative Reattention Blocks aiming at making the most of all inputs, 4) Attention Flow Layer for query and context vector coupling, 5) Modeling Layer for query-aware context representation encoding, and 6) Output Layer for predicting Start & End position.

3.1 Baseline model

BPM is based on the baseline model provided by the default project code[1] which is derived from BiDAF [2]. Compared to the original BiDAF, this baseline doesn't include Embed Layer for character embedding. Specifically, the baseline model includes

- Embed Layer for word embedding where pretrained word vectors from Glove[9] are fixed during training.
- Contextual Embed Layer adopts a one-layer bidirectional Long Short-Term Memory Network (LSTM) [10] for the context/query sequence encoding, thus could obtain the interaction among context/query words.

[1]`https://github.com/michiyasunaga/squad`

- Attention Flow Layer is used for obtaining the correlation and fusion information between context and query from both direction. To be specific, Context-to-query (C2Q) attention highlights the more related words in query to each words in context while Query-to-context (Q2C) attention focuses on similarities between words in context to every words in query.
- Modeling Layer utilize a two-layer bidirectional LSTM for attaining the interaction among context words that are attended to query.
- Output Layer includes two linear projections for outputs from Attention Flow Layer (denoted as G) and Modeling Layer (denoted as M) respectively for Start position prediction. A one-layer bidirectional LSTM is used to model M and then follows by two other linear projections for G and LSTM outputs respectively for the End position prediction.

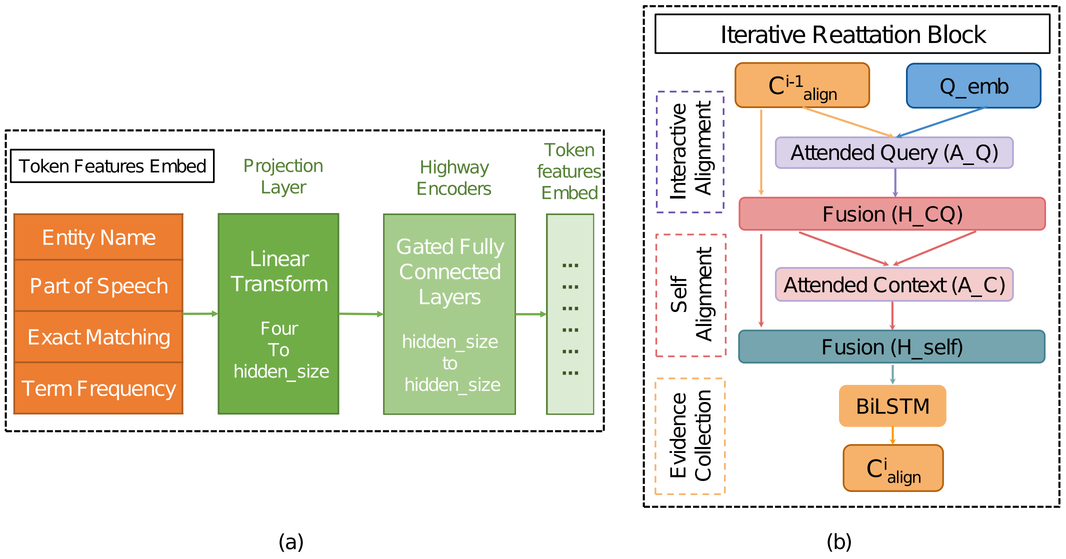


Figure 2: (a) The detailed overview of token features and the embed layer. (b) Illustration of one Iterative Reattetion Block.

3.2 Character embedding

We implemented the character embed layer as in the original BiDAF paper[2] following the original CNN embeding structure in the paper by Kim[11]. Each character are first embeded as a trainable vector. As shown in the Appendix Fig. 5, the character embeddings are convoluted. This convoluted matrix is maxpooled along the word span in each channel. The resultant vector is then projected into latent (hidden) layer size (d) and further processed by a small highway network[12] to generate the character embeded latent vector.

3.3 Token feature and its embedding

In addition to the Glove[9] word embed layer and character embed layer, we adopted ideas from Chen *et al.*[5] on additional token features to create a third latent vector. There are four features implemented by this project, and these features are all pre-computed for the train and test dataset for efficiency.

- ENT: During data pre-processing, all context are processed by spaCy's small language model based on WordNet 3.0[2]. The words in context belong to a noun phrase and is a named entity are tagged the "named entity type" (ENT). For example, in "*Apple is looking at buying U.K. startup for $1 billion*", "*Apple*" is tagged as an organization, "*U.K.*" is tagged as a geological entity, and "*$1 billion*" is tagged as money. Each entity type, including "not an entity name", is assigned an index and stored as part of the dataset.

[2]`https://spacy.io/models/en`

- POS: Similar to "ENT" token feature, spaCy processing also returns a "part of speech"(POS) tag. In the previous example, "*Apple*" is tagged as "proper noun singular". All words in the context are labeled a tag index.
- EM: For each word in the context, its lowercase version is checked against query's words in lowercase for an "exact match"(EM). If there exists a match in the query, the word in the context is labeled as 1, else 0.
- TF: The "term frequency" of a word is determined by normalizing statistics of a English word frequency data published on Kaggle by Rachael Tatman[3]. This statistics contain 1/3 million most common English words's frequency in Google Web Trillion Word Corpus. All words in the context are labeled a normalized frequency if the word is among the top 10000 frequent word, else labeled zero.

These four token features forms a vector length of four for each word in the context. As shown in Fig. 2(a), this vector is embeded into the token features latent layer with a projection layer and a small highway network to produce. The word, character, and token features latent layers are added together for downstream layers.

3.4 Iterative attention

Besides Attention Flow Layer, following the idea of reattention mechanism from Hu *et al.* [6], we also applies iterative reattention blocks to take fully advantage of the inputs which could temporally memorized the previous attention and refine the current attention based on it. As shown in Fig. 2(b), each reattention block is composed by three components. Specifically, given the inputs $C^{i-1}_{align} \in \mathbb{R}^{T\times 2d}$ which is also the output from $(i-1)^{th}$ reattention block where T is the word length of context, and $Q_{emb} \in \mathbb{R}^{J\times 2d}$ from Contextual Embed Layer for query encoding where J is the word length of query, processing details are shown as follows,

- 1) **Interactive Alignment.** To get the attended vector of the question into the context, similarity matrix between C^{i-1}_{align} and Q_{emb} is firstly calculated as
$$S^i_{CQ} = C^{i-1}_{align} Q^T_{emb}, \tag{1}$$
where $S^i_{CQ} \in \mathbb{R}^{T\times J}$ and the attended query vector is then obtained by
$$A^i_Q = softmax(S^i_{CQ}) Q_{emb}, \tag{2}$$
where $A^i_Q \in \mathbb{R}^{T\times 2d}$. Then, a fusion function is applied to mix together the information from C^{i-1}_{align} and the attended query vector A^i_Q. Denote the function as $Fusion(x, y)$, its output as o, then we have
$$\begin{aligned} g &= \text{ReLU}(W_r[x; y; x \circ y; x - y]) \\ h &= \sigma(W_g[x; y; x \circ y; x - y]) \\ o &= h \circ g + (1 - h) \circ x \end{aligned} \tag{3}$$
where σ indicates sigmoid activation function, $\circ$ denotes the Hadamard product, h serves as a gate to control the contribution from two different vectors. And the output of the fusion function in 1) is denoted as $H^i_{CQ} = Fusion(C^{i-1}_{align}, A^i_Q)$.
- 2) **Self Alignment.** To obtain the attended vector of the context conditioned on itself, similar as 1), a self similarity matrix is computed as
$$S^i_{self} = H^i_{CQ} H^{Ti}_{CQ}, \tag{4}$$
where $S^i_{self} \in \mathbb{R}^{T\times T}$ and the attended context vector is then obtained by
$$A^i_{self} = softmax(S^i_{self}) H^i_{CQ}, \tag{5}$$
where $A^i_{self} \in \mathbb{R}^{T\times 2d}$. Then fusion function is applied again to get the self-aware context vector $H^i_{self} = Fusion(H^i_{CQ}, A^i_{self})$.
- 3) **Evidence Collection.** In this part, a two-layer bidirectional LSTM is used to aggregate the information from the self-aware context vector H^i_{self} that
$$C^i_{align} = \text{BiLSTM}(H^i_{self}). \tag{6}$$

[3] `https://www.kaggle.com/rtatman/english-word-frequency`

3.5 Other approaches

Besides our proposed BPM, we also explored several different approaches.

3.5.1 Coattention Mechanism

Inspired by the Dynamic Coattention Network[3], the Coattention Layer is implemented, which involves a two-way attention between the context and query. The main difference from BiDAF is that Coattention includes a second-level attention computation that attends over attention outputs themselves. Given the context hidden state $C \in \mathbb{R}^{T \times 2d}$ and the question hidden state $Q \in \mathbb{R}^{J \times 2d}$, the question hidden state is first projected to Q' by

$$Q' = \tanh(WQ + b) \in \mathbb{R}^{J \times 2d}. \tag{7}$$

Then sentinel vectors are added to both hidden states. Specifically, $C = \{C; c_\emptyset\} \in \mathbb{R}^{(T+1) \times 2d}$ and $Q = \{Q; q_\emptyset\} \in \mathbb{R}^{(J+1) \times 2d}$. Next, the affinity matrix L is computed by

$$L = CQ^T \in \mathbb{R}^{(T+1) \times (J+1)}. \tag{8}$$

Outputs of the Context-to-Question Attention are obtained by

$$\alpha = \text{softmax}(L, \text{dim} = 1) \in \mathbb{R}^{(T+1) \times (J+1)}, \qquad a = \alpha Q' \in \mathbb{R}^{(T+1) \times (2d)}. \tag{9}$$

Outputs of the Question-to-Context Attention are obtained by

$$\beta = \text{softmax}(L, \text{dim} = 0) \in \mathbb{R}^{(T+1) \times (J+1)}, \qquad b = \beta^T C \in \mathbb{R}^{(J+1) \times (2d)}. \tag{10}$$

Then, the second-level attention outputs are computed as

$$s = \alpha b \in \mathbb{R}^{(T+1) \times (2d)} \tag{11}$$

Finally the overall output is BiLSTM($\{s[: T, :]; a[: T, :]\}$).

3.5.2 Conditioning end prediction on start prediction

As the baseline model predicts the start index and the end index independently, we connect the start prediction and the end prediction in the output layer, inspired by Wang and Jiang (2016)[4]. Two alternatives are implemented:

i) Based on BiDAF attention, the logits for start prediction are passed to a LSTM block, whose output is added to the logits for the end prediction.

ii) In the first step, an intermediate tensor F_1 and the logits for start prediction are computed. In the second step, the previous logits are passed to a LSTM block, followed by a linear layer. The outputs are added to F_1 to compute the logits for end prediction.

3.5.3 QANet

Besides BiDAF, we also adopt QANet [7] as another model for comparison. QANet only utilizes depth-wise separable convolution and self-attention in the encoders to extract the local feature and maintain the global interaction between pairs of words as well. Besides self-attention, QANet also employs attention mechanism between query and context. Besides the original QANet, we also try another different output layer. Specifically, the original output layer in QANet uses concatenation operation before linear projection, and we try another additive output layer where we use linear projection first and then do addition. Details could be found in Appendix.

4 Experiments

4.1 Data

The primary training, developing, and testing dataset is SQUAD 2.0[1]. Additionally, to generate token features for context data, we utilized spaCy's small English language model to generate named entity and part of speack tags. Finally, we used English word frequency data published on Kaggle, as mentioned in section 4.3.

4.2 Evaluation method

We use the F1 score as the primary metric. We also included EM score in the result section for additional information and experiments analysis.

4.3 Experimental details

All experiments are implemented in Pytorch. We train all models with a batch size of 64 for 40 epochs with a fixed learning rate at 0.5. We use the default hidden size which is 100 and default drop rate which is 0.2. Specially, the drop rate for character embed layer is set as 0.1. As for iterative reattention, we set the number of blocks as 2. We apply Adadelta optimizer for optimization. The objective function is the negative log likelihood loss. For our proposed BPM, training time is nearly 11 hours. Moreover, our proposed BPM is built from scratch (but also based on the starter code). And the compared model QANet is implemented based on `https://github.com/heliumsea/QANet-pytorch`. But we also have done some ablation experiments based on QANet where we build the modification from scratch.

4.4 Results

Table 1: Summary of models performance on dev examples

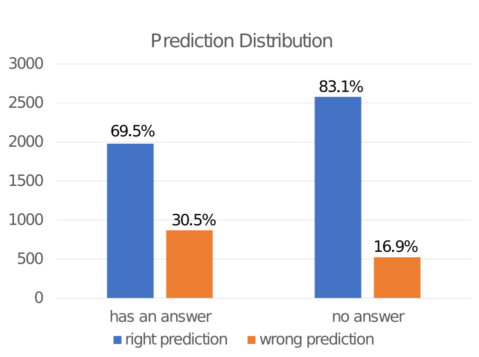
Experiments	F1	EM
BiDAF (baseline)	60.90	57.74
BiDAF + Coattention	54.62	50.93
BiDAF + Cond. Prediction	62.08	58.60
QANet	65.03	62.09
QANet + Add. Output Layer	66.13	63.13
BiDAF(C)	65.54	62.02
BiDAF(C) + Iter.Attn.	70.59	67.20
BiDAF(C) + Token Features	76.95	73.25
BiDAF(C) + Token Features + Iter. Attn.	80.32	76.59
BiDAF(C) + Token Features + Iter. Attn. hyper-parameter tuned	**81.65**	**77.89**

Besides the setting mentioned above, we also try to set the decaying rate of the learning rate as 0.9 for every 3 epochs. This finetuned BPM's test leader-board F1 score is **76.63** and EM is **73.27**. On dev examples, as shown in Table 1, it has F1 score **81.65** and EM score **77.89**, outperforms our other attempts by a large margin.

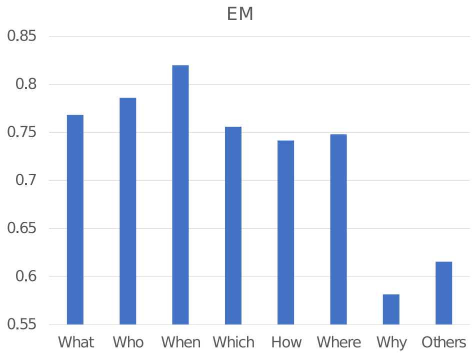
Besides the overall evaluation, we also do statistic analysis on 1) the ability for our proposed BPM to deal with questions with and without answers, and 2) the ability for BPM to deal with questions with specific interrogative words. As shown in 3a, we observe that our model has higher EM for questions without answers. We believe possible reasons might be that predicting no answer is more like a binary classification which might be much easier than finding the exact start/end position for questions with answers. We also select several commonly showing up interrogative words in questions. As shown in 3b, we notice that our BPM has the top2 EM (81.98, 78.60) for questions with 'when' and 'who', and lowest EM (58.14) for questions with 'why'. We think answers for questions like 'when', 'who' might be short, obvious and concentrative while answers for 'why' could be varied and thus resulting in low EM. More result details are shown in Table 2 in Appendix.

Coattention and Conditioning Prediction are implemented based on the baseline model, but are not included in the final model, as they either underperform more than expected (Coattention) or have limited improvements (Conditioning end prediction on start prediction).

For QANet, we notice that it has very comparable results with BiDAF(C). However, after training for 1 epoch , it could reach 57.17 in F1 while F1 of BiDAF(C) is only 50.75. As for the higher F1 shown in QANet + Additive output layer, we think it might be that additive layer are more flexible since there are four different learn-able matrices comparing to only two in the original output layer.

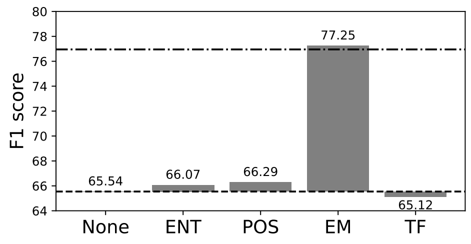


(a) Prediction distribution for questions w/ or w/o answers

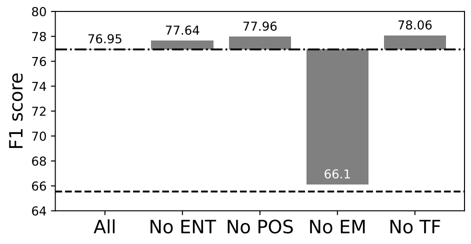


(b) EM for questions with specific words

Figure 3: Quantitative analysis



(a) Comparing BiDAF(C) to BiDAF(C) with one token feature



(b) Comparing BiDAF(C) with four token features to BiDAF(C) with three

Figure 4: Ablation Study Results

5 Analysis

5.1 Ablation Study

The token features bring such a significant jump in F1 and EM metric compared to only character embedding. Thus, we conducted a ablation study on the four features to analyze which one is most effective. To make the comparison fair, we have two sets of experiment.

- Single Token Feature
  In these experiments, all other three token features are replaced by zeros, while keeping the rest of the network identical (BiDAF(C) configuration). All of these models are trained under the same initialization seed and same hyper-parameters. As shown in Fig. 4a, the EM token features is the most effective feature among the four.
- All Token Features Except One
  Additional studies are conducted by erasing one feature to zero while other model architecture and training procedures are identical. The results in Fig. 4b show that by removing some features may help model to achieve slightly better F1 score. However, the difference is almost negligible. Again, these experiments confirm previous observation of the importance of EM token feature, removing such would result in a plunge of F1 score.

One interesting finding is that while we include token features in the model, our model could result in a good performance after very few epochs. BPM could achieve 70 in F1 after only 3 epochs. We think this indicates that our token features have a very strong correlation with the answers which makes the training much easier. Another interesting finding is that even though BiDAF(C) with all four token features has lower F1 than BiDAF(C) with three (no TF), after combining with iterative reattention, our BPM could result in higher F1 with all four token features than BPM with three (no TF) (80.32 vs. 79.36). We believe the reason might be that the iterative reattention mechanism could have a better use of input features and obtain useful information which could contribute more to the final predictions.

5.2 Failure Examples

1. Example 1

   Question: Why did France choose to give up no continental lands?
   Context: The war in North America officially ended with the signing of the Treaty of Paris on 10 February 1763, and war in the European theatre of the Seven Years' War was settled by the Treaty of Hubertusburg on 15 February 1763. The British offered France the choice of surrendering either its continental North American possessions east of the Mississippi or the Caribbean islands of Guadeloupe and Martinique, which had been occupied by the British. France chose to cede the former, but was able to negotiate the retention of Saint Pierre and Miquelon, two small islands in the Gulf of St. Lawrence, along with fishing rights in the area. They viewed the economic value of the Caribbean islands' sugar cane to be greater and easier to defend than the furs from the continent. ...
   Answer: N/A
   Prediction: able to negotiate the retention of Saint Pierre and Miquelon

   This is an example of a challenging adversarial written question. The question asks for why France give up no continental lands but the context shows that France give up continental lands. Thus, there shouldn't be an answer. However, the model fails to pick up the subtleties of how the question is framed. Its answer seems to be quite reasonable for "why France give up continental lands?". Thus, we suspect that the model is too insensitive to words like "no" that flip the question or context completely. To improve the model, we suggest to allow fine-tune of word vectors for such special words.

2. Example 2

   Question: What is the expression used to denote a ***worst case complexity*** as expressed by time taken?
   Context: For example, consider the deterministic sorting algorithm quicksort. This solves the problem of sorting a list of integers that is given as the input. The ***worst-case*** is when the input is sorted or sorted in reverse order, and the algorithm takes time **O(n2)** for this case. If we assume that all possible permutations of the input list are equally likely, the average time taken for sorting is O(n log n). The best case occurs when each pivoting divides the list in half, also needing O(n log n) time.
   Answer: O(n2)
   Prediction: N/A

   This example shows one potential failure mode of our model. The "hint" from question is *worst-case* and *complexity*. However, neither of these two words show up in the context at its original form. "worst-case" becomes "worst case" and "complexity" is implied by the underlying big O definition. On one hand, the exact match, our most effective approach would fail completely. Additionally, Other mechanism would easily fail to infer the big O definition if the training set does not contain enough time complexity related problem. On the other hand, this problem would even be hard for one without a computer science background to answer correctly. One way to better deal with this problem is to have better matching algorithm that will catch slightly difference phrases.

Confirming our thoughts in Section 4.4 Fig. 3b, the question answering for certain interrogative words such as "why" would most likely to cause failures. The above examples also shows that the model experience failures when there is limited 'hint' in the context that match the question well.

6 Conclusion

In this project, we present BiDAF Pro Max (BPM) for machine comprehension tasks. Based on the default BiDAF in the starter code, we implement character embedding, include four other token features with its corresponding embed layer, apply an iterative reattention mechanism and together with these components form our final model BPM. And our model shows a good performance on both dev and test dataset. All the added components show improvements in the results and EM feature among all four token features has the greatest contribution. This also indicates that if questions don't contain words shown up in contexts or only have synonyms, EM might only have little contribution to the performance. Moreover, using iterative reattention blocks could increase the training time compared to BiDAF.

References

[1] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Association for Computational Linguistics (ACL)*, 2018.

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

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

[4] Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *arXiv preprint arXiv:1608.07905*, 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] Minghao Hu, Yuxing Peng, Zhen Huang, Xipeng Qiu, Furu Wei, and Ming Zhou. Reinforced mnemonic reader for machine reading comprehension. *arXiv preprint arXiv:1705.02798*, 2017.

[7] 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 preprint arXiv:1804.09541*, 2018.

[8] Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension. *CoRR*, abs/1611.01603, 2016.

[9] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[11] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[12] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.

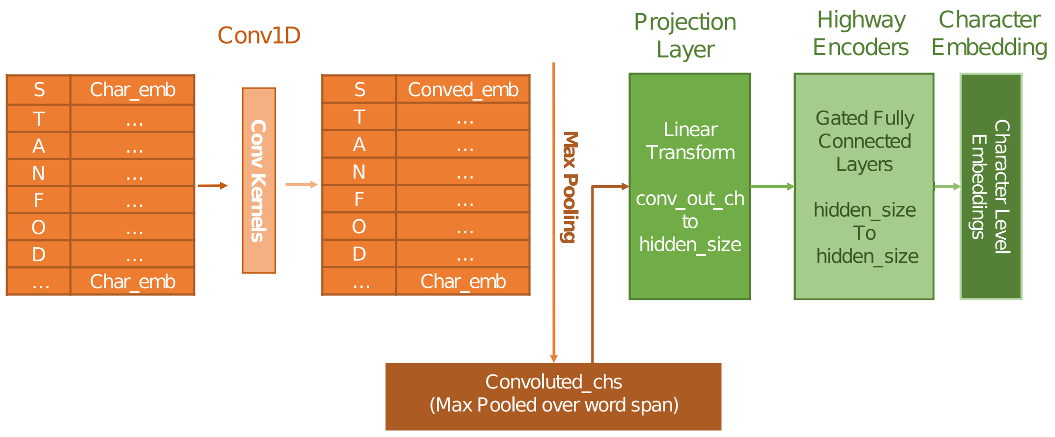


Figure 5: The detailed overview of Character Embed Layer Architecture.

Table 2: EM for questions with different interrogative words

interrogative words	# questions	# right prediction	EM
What	3649	2803	76.82
Who	687	540	78.60
When	444	364	**81.98**
Which	213	161	75.59
How	561	416	74.15
Where	245	184	74.80
Why	86	50	58.14
Others.	65	40	61.64

A Appendix

A.1 QANet ablation study

As for the output layer, QANet use the concatenation operation where

$$p^1 = softmax(W_1[M_0; M_1]), \quad p^2 = softmax(W_2[M_0; M_2]), \tag{12}$$

p_1, p_2 represent the start and end position respectively. W_1, W_2 are trainable variables, M_0, M_1, M_2 are the outputs from the three consecutive Encoder Blocks which share same weights. Inspired by the character embed layer, we also decided to use addition as another choice that

$$p^1 = softmax(W_0^1 M_0 + W_1^1 M_1)), \quad p^2 = softmax(W_0^2 M_0 + W_2^2 M_2), \tag{13}$$

where $W_0^1, W_1^1, W_0^2, W_2^2$ are trainable variables.

A.2 Character Embed Layer Architecture

Please see Fig. 5 for detailed structure.

A.3 Details results about the interrogative words