Reading Comprehension using BiDirectional Attention Flow

Khine Myat Thwel
5KE-6
University of Information Technology
khinemyatthwe@uit.edu.mm

Su Myat Noe
5KE-10
University of Information Technology
sumyatnoe@uit.edu.mm

Abstract

We implemented bi-directional attentional flow to solve the reading comprehension problem. The problem is, given a context (that contains the answer to the question) and question related to the context,then find the start and end index of answer in the context. We use bi-attention to make the link from question to context and from context to question, to make good use of the information of relationship between the two parts. According to experimental evaluations, our model has performance of 65.563%EM and74.952% F1 scores.

1 Introduction

We implement this reading comprehension system with SQuAD (Standford Question Answering Dataset), a reading comprehension dataset built by crowdworkers on a set of Wikipedia articles. The dataset contains more than 100,000 question-answer pairs on 500+ articles. Humans generated questions using that paragraph as a context, and selected a span from the same paragraph as the target answer. The following is an example of a triplet ⟨question, context, answer⟩.

> Question: Why was Tesla returned to Gospic?
> Context paragraph: On 24 March 1879, Tesla was returned to Gospic under police guard for **not having a residence permit**. On 17 April 1879, Milutin Tesla died at the age of 60 after contracting an unspecified illness (although some sources say that he died of a stroke). During that year, Tesla taught a large class of students in his old school, Higher Real Gymnasium, in Gospic.
> Answer: not having a residence permit

In the SQuAD task, answering a question is defined as predicting an answer span within a given context paragraph. In our project, we need to give the indexes of the start word and the end word. We explored different models to solve this problem.

The rest of the paper is organized as following: Section 2 defines the problem; Section 3 BiDAF model architecture and evaluation. section 4 ,the related work; In section 5, experiments of the model and analysis will be discussed.; and section 6 is the conclusion and future direction.

1.1 Problem Definition

We define our problem as the following:

Given word sequences of context with length T,$X=\{x_1x_2,x_{3,\ldots,}x_T\}$ and question with length J,$Q=\{q_{1,}q_{2,}q_{3,\ldots,}q_J\}$.The model needs to learn the function f:(X,Q) maps to $\{a_s,a_e\}$ with the condition $1\leq a_s\leq a_e\leq T.$ $\{a_s,a_e\}$ is a pair of scalar indices pointing to the start position and end position respectively in the context X,indicating the answer to the question Q.

2 Related Work

There has been a lot of work on building deep learning systems for the SQuAD Dataset, as can be seen in the leaderboard (https://rajpurkar.github.io/SQuAD-explorer/) .The top models utilize some form of attention mechanism, which has been proven useful in improving accuracies in many NLP tasks. It was first introduced in Neural Machine Translation [], where it generate a translation word based on the whole original sentence states, not just the last state. The idea is then applied to reading comprehension, allowing the model to select a subset of context paragraph and a subset of question that are most relevant. That way, the model can use the most relevant information to give a better answer.

One successful model for this task is the Dynamic Coattention Networks. This model consists of document encoder, question encoder, coattention encoder, and dynamic pointer decoder. The coattention encoder is the attention mechanism for this model. It encodes the interaction between the encoded question and the encoded document. The dynamic pointing decoder uses highway maxout network to compute the probability of the start index and the end index. The process is repeated a few times, using the information of the previous prediction to improve the next prediction. This iterative process allows the model the escape from a local maxima.

Another successful model for reading comprehension is the Bidirectional Attention Flow model. This model introduces a bidirectional attention flow mechanism to obtain query-aware context and context-aware query without early summarization. This is the model that we have implemented that which will be discussed in Section 3.

3. Proposed System

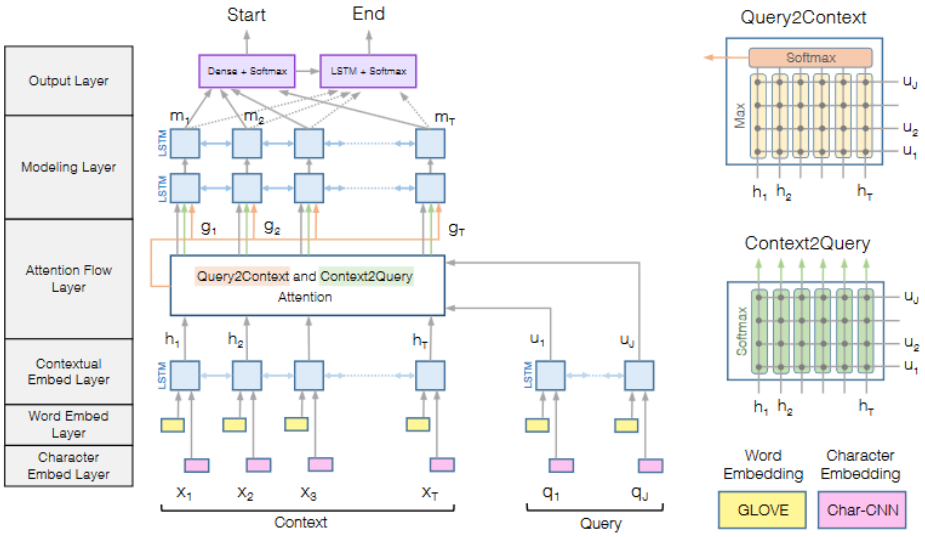

Figure 1: BiDirectional Attention Flow Model *(best viewed in color)*

We refer the original BiDAF paper and do some revision to set up our very first model. The method in paper consists of six layers and each layer is illustrated as follows(we modified some of them to improve the performance, so some of them will be different from the ones showed in paper but we keep the definition of symbols to be the same with that in the original paper.)

1. **Character Embedding Layer:** maps each word to a vector space using character-level CNNs.

2. **Word Embedding Layer:** Word embedding layer maps each word to a high dimensional vector space. We use retrained word vectors, Glove, to obtain the fixed word embedding of each word.

3. **Contextual Embedding Layer:** utilizes contextual cues from surrounding words to refine the embedding of the words. These first three layers are applied to both the query and context.

4. **Attention Flow Layer:** couples the query and context vectors and produces a set of query-aware feature vectors for each word in the context.

5. **Modeling Layer:** employs a LSTM to capture the interaction with query and context.

6. **Output Layer:** provides an answer to the query.

1. **Character Embedding Layer:** Character embedding layer is responsible for mapping each word to a high-dimensional vector space. Let$\{x_1 ... e_{xt.}\}$ and $\{q_1$, Qu} represent the words in the input context paragraph and query, respectively. Following Kim (2014), we obtain the character-level embedding of each word using Convolutional Neural Networks (CNN). Characters are embedded into vectors, which can be considered as 1D inputs to the CNN, and whose size is the input channel size of the CNN. The outputs of the CNN are max-pooled over the entire width to obtain a fixed-size vector for each word.

2. Word Embedding Layer: Word embedding layer also maps each word to a high-dimensional vector space. We use pre-trained word vectors, Glove (Pennington et al., 2014), to obtain the fixed word embedding of each word. The concatenation of the character and word embedding vectors is passed to a two-layer Highway Network (Srivastava et al., 2015). The outputs of the Highway Network are two sequences of d-dimensional vectors, or more conveniently, two matrices: $X \in R^{d \times T}$ for the context and $Q \in R^{d \times J}$ for the query.

3. **Contextual Embedding Layer**. We use a Long Short-Term Memory Network (LSTM) (Hoch Reiter & Schmidhuber, 1997) on top of the embeddings provided by the previous layers to model the temporal interactions between words. We place an LSTM in both directions, and concatenate the outputs of the two LSTMs. Hence we obtain $H \in R^{2d \times T}$from the context word vectors X, and $U \in R^{2d \times J}$ from query word vectors Q. Note that each column vector of H and U is 2d dimensional because of the concatenation of the outputs of the forward and backward LSTMs, each with d-dimensional output.

It is worth noting that the first three layers of the model are computing features from the query and context at different levels of granularity, akin to the multi-stage feature computation of convolutional neural networks in the computer vision field.

4.Attention Flow Layer: To find the "resonance" between contexts and query, namely find the most relevant query words for each context words and

the most relevant context words for each context words In our work, the attention vector will change in every step, based on the attention vector at last step and the current word. So the attention vector at each time step and the embeddings from previous layers flow during the reading process of the whole text. And this temporal related property of LSTM successfully prevents information losing. Two attentions, namely query-to-context attention and context-to-query attention, are computed from question encoding H and the query encoding U. The outputs of the layer are the query-aware vector representations of the context words, G, along with the contextual embeddings from the previous layer. We compute attentions in two directions: from context to query as well as from query to context. We create a matrix S to describe the similarities between the contextual embeddings of question(U) and the embeddings in context(H), $S \in R^{T \times J}$, where S_{tj} indicates the similarity between t-th context word and j-th query word and it I_s computed via

$$S_{tj}=\alpha\ (H{:}\ t, U{:}j \in R.)$$

A is a function that computes the similarity between its two input vector with trainable weights, H:t is t-th column vector of H, and U:j is j-th column vector of U. There multiple ways of choosing αand w did experiment to find out the best function. We will come back to this in later part of this report. Now we use S to obtain the attentions and the attended vectors in both directions. In our work we use different kinds of inner product to describe the similarity of words in question and in context. For example, one way we used to calculate α(h, u)is $\alpha(h, u)=h^{\top}W_{bi}u$, and the matrix W_{bi} can be optimized by training.

Context-to-query Attention: Context-to-query (C2Q) attention signifies which query words are most relevant to each context word. In other words, it likes that we first read the context then find the match part in the question. $a_t \in R^J$ represents the attention weights on the query words by t-th context word,Satj=1forallt.The attention weight is computed by $a_t=\text{softmax}(S_t{:}) \in R^{J\cdot}$, and each attended query vector is $\tilde{U}{:}t=S_j a_{tj} U_{:j}$. Hence $\tilde{U}$ is a $2^{d \times T}$ matrix

containing the attended query vectors for the entire context.

Query-to-context Attention: Query-to-context (Q2C) attention signifies which context words have the closest similarity to one of the query words and are hence critical for answering the query. We get the attention weights on the context words by b = softmax (maxcol(S))$\in R^T$, where the maximum function (maxcol) is performed across the column. Then the attended context vector is $\tilde{h} = S_t b_t H{:}t \in R^{2d}$. This vector indicates the weighted sum of the most important words in the context with respect to the query. $\tilde{h}$is tiled T times across the column, thus giving $\tilde{H} \in R^{2d \times T}$After the two steps we get two sets of attentions $\tilde{U}$and $\tilde{H}$. Finally, the contextual embeddings and the attention vectors are combined to yield G, where each column vector can104be considered as the query aware representation of each context word. We define G by

$$G{:}t = \beta(H{:}t, \tilde{U}{:}t, \tilde{H}{:}t) \in R^{dG}$$

where G:tis the t-th column vector (corresponding to t-th context word), β is a trainable vector function that fuses its (three) input vectors, and ^{dG} is the output dimension of the β function. β can be different, such as in the paper: $\beta(h, \tilde{u}, \tilde{h}) = [h;\tilde{u};h\circ\tilde{u};h\circ\tilde{h}] \in R8^{d \times T}$(i.e. ^{DG}=8d).In our model, we use a modified version of$\beta(h, \tilde{u}, \tilde{h})$ =max(0, $W_{mlp}[h;\tilde{u};h\circ\tilde{u};h\circ h]+ b_{mlp} \in R^{8d \times T}$, which is a linear transformation of the original β, and it's like a ReLU. After this process, we combined the information in question and in context.

5. Modeling Layer: We use again, LSTM, to encode query-aware representation for final output. The output of the modeling layer represents the interaction among the context words conditioned on the query. We use two layers of bi-directional LSTM, with the output size of d for each direction. So we get a matrix $M \in^{R2d \times T}$, which is passed onto the output layer to predict the answer. Each column vector of M contains contextual information about the word with respect to the entire context paragraph and the query.

6. Output Layer.

To get the start word and the end word in the answer, we try to transform the question to be that finding the probability of every word to be the start word. We obtain the probability distribution of the start index over the entire paragraph by

$$p_1 = \text{softmax}\ (w^T(p_1)[G;M])$$

where $w(p_1) \in R^{10d}$ is a trainable weight vector. For the end index of the answer phrase, we pass M to another bidirectional LSTM layer and obtain $M^2 \in R^{2d \times T}$. Then we use M^2 to obtain the probability distribution of the end index in a similar manner:

$$p_2 = \text{softmax}(w^T(p_2)[G;M^2])$$

7. Training. We define the training loss (to be minimized) as the sum of the negative log probabilities of the true start and end indices by the predicted distributions, averaged over all examples:

$$L(\theta) = -\frac{1}{N}\sum_{i}^{N} \log\left(P^1_{y_i^1}\right) + \log\left(P^2_{y_i^2}\right)$$

8. Test The answer span(k,l) where $k \leq l$ with the maximum value of $p^1_k\, p^2_l$ is chosen, which can be computed in linear time with dynamic programming.

3.1 Evaluation

The original SQuAD paper introduces two metrics to evaluate the performance of a model: Exact-Match (EM) and F1 score. We use the same metrics to evaluate our model. Exact match is a metric that measures the percentage of predictions that match one of the ground truth answers exactly. F1 score is a metric that loosely measures the average overlap between the prediction and ground truth answer. We treat the prediction and ground truth as bags of tokens, and compute their F1. We take the maximum F1 over all the ground truth

answers for a given question, and then average over all the questions.

4 Experiment Results and discussion

The system is implemented using Tensorflow(tensorflow.org). For the 1D convolution and highway networks implementation, we adapted the code from the (https://github.com/allenai/bi-attflow/blob/master/my/tensorflow/nn.py) and changed the code to support dropout keep probability from a tensor and replace the old linear functions with the new tf.fully connected functions. To combine the answer tokens into a sentence, we utilize the untokenize function from https://github.com/commonsense/metanl/blob/master/metanl/token_utils.py and added another step to remove whitespaces between double quotes and the phrase (e.g.” foo bar”→”foobar”).

F1 score and Exact Match (EM) score are used for evaluating the model.F1 score is the harmonic mean of precision and recall (the higher the better). For each question, precision is calculated as the number of correct words divided by the number of words in the predicted answer. Recall is calculated as the number of correct words divided by the number of words in the ground truth answer. The F1 score is computed per question and then averaged across all questions. EM score (the higher the better) is the number of questions that are answered in exact same words as the ground truth divided by the total number of question. My BiDAF implementation achieved 74.952 % F1 and65.563 % EM on the test set. Our implementation of BiDAF has lower F1 score than the original BiDAF and this might be because they did larger hyperparameter search or there are other details that are not mentioned in the paper.

5.1 Analysis on the effect of different components in the model

I tried a lot of different component and parameter combinations and the result can be seen in Table.also important, without it, the F1 score decreased by 5%. This means that contextual interactions between context representations conditioned on the question is important. Using Glove Common Crawl 840B tokens instead of Glove Wikipedia 6B tokens improved the F1 score by 2% and that is because there are fewer words that are not in the vocabulary. One interesting thing that I found is that fixing the Glove vectors results in 1% higher F1 score than training them, and I guess that is because the Glove vectors are already well trained (on 840B tokens) and training them again will cause overfitting. Character embeddings improved F1 score by 1% which does not seem much andit is probably because there are not many words that are not in the vocabulary and the Glove vectors are already representative enough. I have also tried to increase the character embedding output dimension to 100 and the number of highway network layers to two, but that didn't seem to have any noticable impact on the F1 score. Finally, linking the two BiLSTM (use the question's final state as the context's initial state) in the BiDAF's contextual embed layer improved the F1 score by 1%.

In the answer generation, using proper untokenize function that removes whitespace between a word and the punctuation instead of simple token joining improved the F1 score by 0.5% and EM score by 1.5%. That is because there are quite a number of answers in the dev set that contain some punctuations, and simple joining will not give EM score of 1.0 for these questions. For the ensemble strategy, I have tried three different strategies: sumpreds (summing the raw probabilities),maxprob (pick the answer with the highest probability $p1[i] * p2[j]$, and maxvote(majority voting and fallback to maxprob for tie breaker). The highest increase in F1 (3%) is gotten by using maxvote. Maxprob only increased the F1 by 1% and sumpreds did not increase F1 at all. I am also limiting the answer length to a maximum of 10 words and that increased the F1 by another1%. The limit helps because more than 90% of the questions have short answers. Using exponential

moving average of parameters (LSTM weights, biases, etc.) instead of the raw parameters during evaluation improved F1 score by 2-4%.

Table 5.1.1: Experiment results on the dev set.

Own Model (Dev Set)	F1 (%)	EM (%)
My BiDAF (with linked context embed layer)	74.952	65.563
Reimplementation of original BiDAF	74.04	63.916
BiDAF without char embedding	73.911	63.888
BiDAF with fixed Glove6B100d	72.904	62.242
BiDAF without modeling layer	69.863	58.6

5.2 Analysis on different question types and answer lengths

The model's performance on different question types can be seen in Table .The BiDAF model performs the best on when questions and performs the worst on why" question. That is because "when" questions have short answer length (2.24 on average), whereas "why" questions have higher answer length (6.87 on average). The model's performance on different golden answer length can be seen in Table. We can see that the EM score for the last two rows is zero and that is because I limit the maximum of answer length to the maximum of 10 words. I still get a few EM score on the 11-15 range and that is because the evaluation script will normalize the ground truth answer when comparing the answers.

Table 5.2.1: My BiDAF (single model) performance on different question types in the devset.

Question Type (Dev Set)	F1 (%)	EM (%)	#questions	Avg. Answer length
When	82.997	77.662	864	2.24
Who	76.068	69.935	1377	2.62
How	74.654	64.651	1389	2.5
What	73.377	63.181	6073	3.02
Where	73.025	62.008	508	2.89
Why	60.708	36.076	158	6.87

Table 5.2.2: My BiDAF (single model) performance on different golden answer length in the dev set.

Ground Truth Answer Length (Dev Set)	F1 (%)	EM (%)	#questions
1-5	76.770	69.08	9453
6-10	66.502	46.856	843
11-15	38.286	2.105	190
16-20	41.830	0	55
20+	30.855	0	29

6 Conclusion and Future Work

For this paper, We have reimplemented Bidirectional Attention Flow model and made minor modification in the contextual embed layer by using the question's final state as the context's initial state. This reimplementation achieved a competitive result on the test set, but it is still behind the original paper's performance by about 4%. From the component analysis, attention mechanism is the most important component for this task, since using just a simple attention already increased the F1 score by more than 30%. The visualization shows that the model is able to pick relevant context words according to the question and use that information to make a confident prediction on the answer start and end index.

Our intended future work is to replace BiDAF modeling layer with DCN Dynamic Pointer Decoder. The idea is that Dynamic Pointer Decoder is able to recover from local maxima, which should help improving the performance of BiDAF. There is also still a gap between my BiDAF implementation and the original BiDAF implementation and that can still be improved by doing further hyperparameter tuning and bug fixing.

Acknowledgments

We would like to thank all the teachers who were attending to our seminars and gave advices to us.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. CoRR, abs/1606.05250, 2016.

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

[3] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multiperspective matching for natural language sentences. arXiv preprint arXiv:1702.03814, 2017.

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

[5] Danqi Chen, Jason Bolton, and Christopher D Manning. A thorough examination of the cnn/daily mail reading comprehension task. arXivpreprint arXiv:1606.02858, 2016.

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

[7] Kenton Lee, Tom Kwiatkowski, Ankur Parikh, and Dipanjan Das. Learning recurrent span representations for extractive question answering. arXiv preprint arXiv:1611.01436, 2016.

[8] Yelong Shen, Po Sen Huang, Jianfeng Gao, and Weizhu Chen. Reasonet: Learning to stop reading in machine comprehension. arXiv preprint arXiv:1609.05284, 2016.

[9] 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.

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

[11] MinJoon Seo, Aniruddha Kembhavi, Ali Farhadi, and, Hananneh Hajishirzi.Bi-Directional Attention Flow For Machine Comprehension. arXivpreprint arXiv:1611.01603v5, 2017