ADL HW2 REPORT

Huang Liang Ying 04/27/2020

Q1: Tokenization

When tokenizing Chinese, Bert tokenizer apply character-level tokenization. However, when tokenizing English or numbers, Bert tokenizer apply word-level tokenization. It also use Byte-Pair Encoding(BPE) to further separate a word. For example, after tokenization, 'loving and loved' will become 'lov', 'ing', 'ed'.

Q2: Answer Span Processing

1. I use BertTokenizer to encode both inputs(including context and question) and answer text to get two index lists. Then I use custom function find_ans_span() to find token answer span. The exact answer text may appear in context for many times. To put it simply, I just take the first answer text index as token answer span. In addition, if the function return empty list, it means that answer text does not exist in context anymore after truncating. In this situation, I discard the data.

```
#usage: find index of sublist in list
#param: sublist, list
#return: List of tuple of (start_idx, end_idx)

def find_ans_span(self, sl, l):
    results=[]
    sll=len(sl)
    for ind in (i for i,e in enumerate(l) if e==sl[0]):
        if l[ind:ind+sll]==sl:
            results.append((ind,ind+sll-1))
    return results
```

2. I use BertForQuestionAnswering as model. When predicting, model return start_score and end_score (batch_size, sequence_length,) as the score of each tokens as start and end position. I use argmax to find maximum score index and use them as start and end position. If both start and end index are 0, which means token [CLS], I regard this data as unanswerable question.

However, model sometime predict weird result. For example, answer span longer than 30. In this case, I select top 2 scores index from start_score and end_score and sort these four index(i.e, [23, 25, 72, 76]). The four index provides six possible answer text spans, I choose one with shortest length as final answer(i.e. [23, 25], length of 2)

Q3: Padding and Truncating

1. 512
2. I use BertTokenizer's member function prepare_for_model to perform padding and Truncating. I set truncation_strategy to 'only_first' to allow function only truncate context text and set pad_to_max_length to 'True' to pad insufficient sequence length to max length 512.

```
inputs = tokenizer.prepare_for_model(context_ids, question_ids, ↩
    max_length=max_length, truncation_strategy='only_first', ↩
    pad_to_max_length=True)
```

Q4: Model

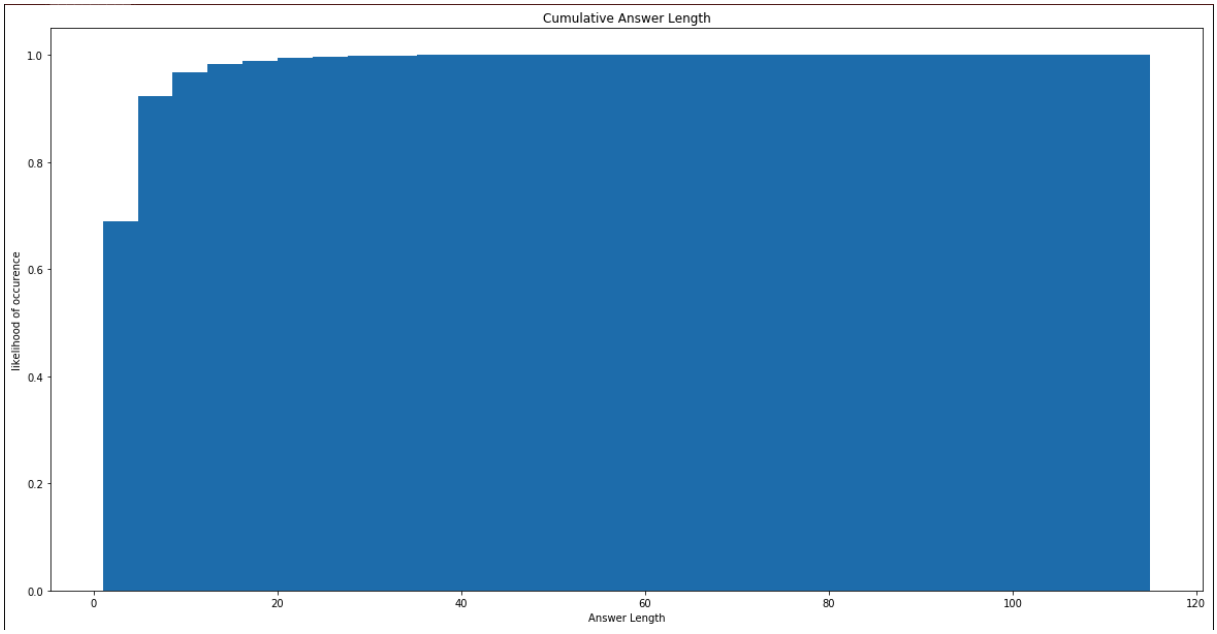
I did not ask my model to predict if the question is answerable or not. Instead, I check index of max start_score and end_score. If both of them are 0, I consider this question unanswerable.

BertForQuestionAnswering model return start_score and end_score as outputs, indicating the score of every tokens as start position and end position of answer. The higher the score, the greater possibility this token is the position where answer starts or ends.

I use BertForQuestionAnswering's loss function, it is total span extraction loss is the sum of a Cross-Entropy for the start and end positions.

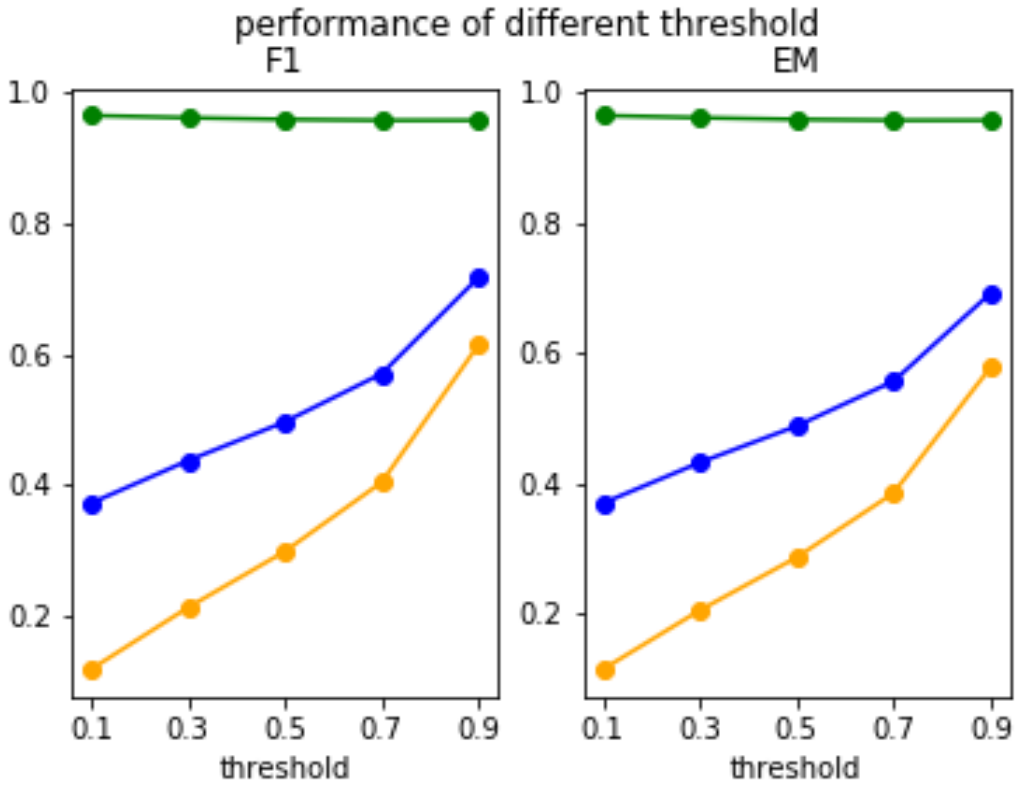
I use AdamW and set learning rate to 8e-6 at the beginning. After training an epoch, I freeze bert's embedding layer as well as the first self-attention encoder layer and adjust learning rate to 8e-7.

Q5: Answer Length Distribution



As the graph shows, almost all answer length of training set data are less than 30 words. If the model output answers with length longer than 30 words, there is a great chance that the answer is incorrect. If my model gives me a 30+ words answer, I will do post-processing as mentioned in Q2 to get better performance.

Q6: Answerable Threshold



For this question, since I did not let my model predict answerable probability on [CLS] token, I just check if start and end positions are both 0, I add [CLS] token's start and end score together

and pass it through sigmoid function. I take the output as answerable probability. If answerable probability are smaller than threshold, model will predict empty string as answer. Indeed, I did not set any threshold in my submitted model.

Q7: Extractive Summarization

The extractive summarization problem can be solved by Bert at ease. We can put whole context as Bert's inputs and convert extractive summary text into start_position and end_position then train the model.