Applied Deep Learning: Assignment 2 - BERT for QA

b07902047 羅啓帆

April 2020

Q1: Tokenization

- *BertTokenizer* uses *WordPiece tokenization.*
- *WordPiece tokenization*
  - This algorithm first initialize its vocabulary with all characters in the language and then expand its vocabulary iteratively by adding the most frequently commbinations of words in current vocabulary. The final vocabulary of this method may contains lots of *subwords*, which is not a full word itself e.g. *ing, ed.*
  - This method can handle OOV (Out Of Vocabulary) words by breaking down words into subwords greedily. This is better than replacing all OOV words by *[UNK]*.
- Examples
  - *BertTokenizer('bert-base-uncased')*
    * *'Buss is digesting, today is a great day.'*
      $\Longrightarrow$ *['bus', '##s', 'is', 'digest', '##ing', ',', 'today', 'is', 'a', 'great', 'day', '.']*
    * *'My phone number is 09123456789.'*
      $\Longrightarrow$ *['my', 'phone', 'number', 'is', '09', '##12', '##34', '##56', '##7', '##8', '##9', '.']*
    * *'091, 123, 345, 886'*
      $\Longrightarrow$ *['09', '##1', ',', '123', ',', '345', ',', '88', '##6']*
  - *BertTokenizer('bert-base-chinese')*
    * *'布斯在消化，今天天氣真好。'*
      $\Longrightarrow$ *['布', '斯', '在', '消', '化', '，', '今', '天', '天', '氣', '真', '好', '。']*
    * *'我手機是09123456789。'*
      $\Longrightarrow$ *['我', '手', '機', '是', '09', '##123', '##45', '##67', '##89', '。']*
    * *'091, 123, 345, 886'*
      $\Longrightarrow$ *['09', '##1', ',', '123', ',', '345', ',', '886']*
- Observations
  From the examples above, we can observe several things:
  - Text in Chinese is split in character level, while text in English is split in vocabulary level and subword level (buss → bus + s, digesting → digest + ing).
  - Same numbers may be tokenized differently in different languages. This reflects that the frequencies of number usages are different between languages (at least different between Chinese and English).

Q2: Answer Span Processing

- Convert start/end position on character to position on tokens

```python
def convert(raw_context_text, raw_answer_text, tokenizer):
    '''
    raw_context_text: original context string
    raw_answer_text: original answer string
    tokenizer: pretrained tokenizer from transformers package
    '''
    context_text = tokenizer.tokenize(raw_context_text)
    answer_text = tokenizer.tokenize(raw_answer_text)
    raw_answer_start_position = raw_context_text.find(raw_answer_text)

    start_position = len(
        tokenizer.tokenize(raw_context_text[:raw_answer_start_position])
    )
    end_position = start_position + len(answer_text) - 1

    return start_position, end_position
```

In short, I tokenized three strings and use the length of them to determine the start/end positions on tokens.

- Determine final start/end position

```
def get_final_position(start_logits, end_logits, text_start_idx, text_end_idx,
    max_len):
    '''
    start_logits: raw start scores from model (before softmax, single example)
    end_logits: raw end scores from model(before softmax, single example)
    text_start_idx: paragraph start idx in tokenization
    text_end_idx: paragraph end idx in tokenization
    max_len: maximum length of answer
    '''
    # apply softmax on raw scores and get the top 2 of them
    start_logits = torch.softmax(start_logits, dim=0)
    end_logits = torch.softmax(end_logits, dim=0)
    start_scores, start_ids = start_logits.topk(2, dim=0)
    end_scores, end_ids = end_logits.topk(2, dim=0)

    # enumerate over four possible combinations
    # add to possible if the range formed by [start_idx, end_idx] is valid
    possible = []
    for start_score, start_idx in zip(start_scores, start_ids):
        for end_score, end_idx in zip(end_scores, end_ids):
            if start_idx < text_start_idx or start_idx > end_idx or end_idx -
    start_idx > max_len:
                continue
            else:
                possible.append((
                    start_score + end_score,
                    start_idx,
                    end_idx
                ))
    possible = sorted(possible, key=lambda t: t[0], reverse=True)

    # determine answerable
    null_score = ((start_scores[0] + end_scores[0]) * 0.5)
    if len(possible) == 0 or null_score < threshold:
        return None
    else:
        return possible[1], possible[2]
```

Q3: Padding and Truncating

- Maximum input token length of bert-base-chinese: 512
- Conbine context and question to form the input

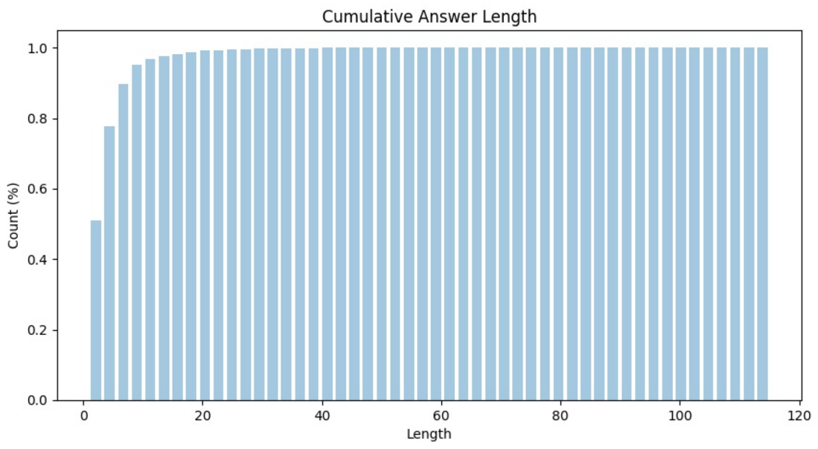
```python
def preprocess(question_tokens, context_tokens, ans_tokens, ans_start_position,
    ans_end_position):
    # truncating
    # 3 is for [CLS], [SEP], [SEP]
    if len(question_tokens) + len(context_tokens) + 3 > 512:
        # truncate question first
        if len(question_tokens) > 64:
            question_tokens = question_tokens[:64]
        left_len_for_context = 512 - 3 - len(question_tokens)
        # truncate context if necessary, always keep answer in the context
        if ans_end_position + 1 > left_len_for_context:
            context_tokens = context_tokens[
                ans_end_position + 1 - left_len_for_context:ans_end_position + 1
            ]
        else:
            context_tokens = context_tokens[:left_len_for_context]

    input = ['[CLS]'] + question_tokens + ['[SEP]'] + context_tokens + ['[SEP]']

    # padding
    while len(input) < 512:
        input.append(['[PAD]'])

    # final input
    return input
```

Note that the code given here is only pseudocode, I implemented the method above in a different way (utilizing methods in *BertTokenizer*).

Q4: Model

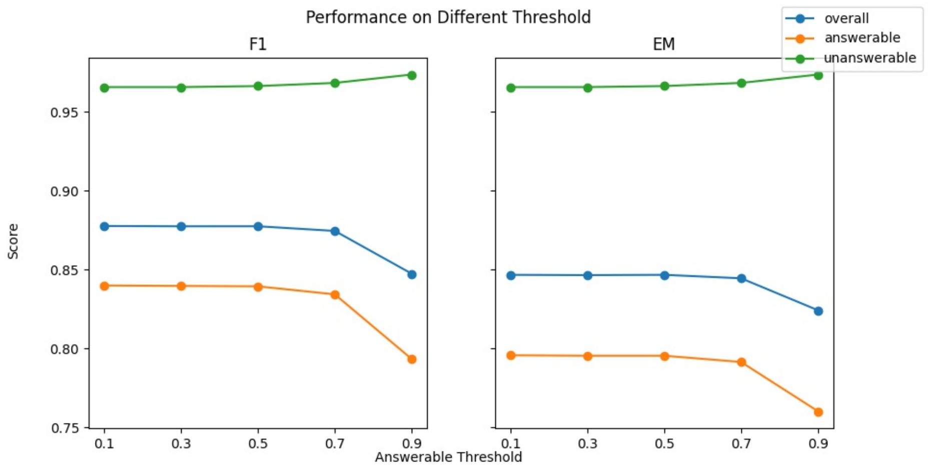
- Answerable problem:
  We can determine whether a problem is answerable or not is determined by the logits correspond to *'[CLS]'*. If the probablity is larger than a threshold, then it is answerable. Else, it is unanswerable.
- Answer span:
  It gives each position two probablities: one represent the probablity for the answer span to start there and another represent the probablity for the answer span to end there.
- Loss function: *CrossEntropyLoss* in torch package.
- Optimization algorithm:
  I used *AdamW(eps=1e-8)* as optimizer, and I also used a learning rate scheduler for adjusting lr of optimizer (*get_linear_schedule_with_warmup*).

Q5: Answer Length Distribution



We can observe that almost all length of answers are less than 30, so I picked 30 as the maximum length for the answer. This means that I'll only pick answers with length less than 30.

Q6: Answerable Threshold



- My probablity threshold: 0.5

Q7: Extractive Summarization

- Preprocess: Tokenize the paragraph using *BertTokenizer* and add *[EOS]* token to the end of each sentences. Then, create a target vector for it where only the position corresponding to the *[EOS]* token of the ground truth sentence is labeled one.
- Model: Add a linear layer on top of *BERT* and train it with *CrossEntropyLoss*.
- Postprocess: Pick the top k sentence with the largest scores, where the score is determined by the value of *[EOS]* after softmax. The value of k is determined by the performance on train data. (I used $k = 2$ in *hw1*).