ADL Assignment 2

B04605067 機械五 黃沛沅

Q1. Tokenization:

Describe in detail how BERT tokenization works.

I used bert-base-chinese pretrained tokenizer. First, it helps to tokenize sentence, convert all English word into lower case. Secondly, the 'convert_tokens_to_ids' function converts all the tokenized tokens into integer lists. Finally, I used the 'prepare_for_model' function. This function prepares a sequence of input pairs (context and question in our task), so that it can be used for the bert model. It returns three important components for us to utilize. Input_ids refers to the sentence ids and add the special token '[CLS]' in the start of sentence and the token '[SEP]' in the middle and the end of the sentence. Token_type_ids is sentence segmentation, labeling the first sentence (paragraph) to be 0, second sentence (question) to be 1. attention_mask labels all non-zero ids to be 1, else 0, to tell the model to notice which positions.

What happens when the method is applied on different strings (e.g. Chinese, English or numbers)? Briefly explain your observation.

```
In [117]: from transformers import BertTokenizer
          model_version = 'bert-base-chinese'
          tokenizer = BertTokenizer.from_pretrained(model_version, do_lower_case=True)

          len(tokenizer.get_vocab())

Out[117]: 21128
```

The tokenizer vocabulary contains 21128 characters in it. Aside from chinese character, it also has some numbers, basic English vocabulary, Japanese character and symbol like ',!?'… .

If the input string is chinese, it converts token into ids if the token exists in the dictionary, else it returns '[UNK]' whose id is 100.

When the tokenizer deal with English or number, the word will break down into pieces of subwords, for example, kobe to 'ko' and 'be', both of them are in vocabulary. However, in order to retain the information that ko and be should be one word, the tokenizer adds '##' at the beginning of the be.

```
Kobe Bean Bryant August 23, 1978 to January 26, 2020 was an American professional basketball player.
['ko', '##be', 'bean', 'br', '##yan', '##t', 'august', '23', ',', '1978', 'to', 'january', '26', ',', '2020', 'was',
'an', 'american', 'pro', '##fe', '##ssion', '##al', 'bas', '##ket', '##ball', 'player', '.']
[10368, 8765, 13188, 8575, 10777, 8165, 9217, 8133, 117, 8774, 8228, 9768, 8153, 117, 8439, 9947, 9064, 9735, 8376, 9
568, 12726, 8315, 10952, 11820, 11050, 9075, 119]
['ko', '##be', 'bean', 'br', '##yan', '##t', 'august', '23', ',', '1978', 'to', 'january', '26', ',', '2020', 'was',
'an', 'american', 'pro', '##fe', '##ssion', '##al', 'bas', '##ket', '##ball', 'player', '.']
```

Q2. Answer Span Processing

How did you convert the answer span start/end position on characters to position on tokens after BERT tokenization?

I also converted answer into ids, and I found the first word of the answer equal to the word in the context. If all the other words in answers match to the word in context respectively, I would record the start position, and the end position would be (start position + answer length – 1)

```
def find_sub_list(sl,l):
    results=[]
    sll=len(sl)
    for ind in (i for i,e in enumerate(l) if e==sl[0]):
        if np.array_equal(l[ind:ind+sll], sl): # check array is equal
            results.append((ind,ind+sll-1))

    return results
```

After your model predicts the probability of answer span start/end position, what rules did you apply to determine the final start/end position?

After predictions, I get the start logits and end logits, and then let it go through softmax layer and find the maximum. I apply some rules to determine the actual answer:

1. If the predicted start position > end position, the answer is set to empty.
2. If the predicted length > 30, the answer is set to empty
3. If the start position and end position do not follow the above 2 rules, it is viewed as normal circumstances, so we can get the start and end position we want.

```
start = sta_result[i]
end = end_result[i]

if start > end:
    ans_dict[question_id[i]] = ''


if end - start > 30:
    answer = tokenizer.convert_ids_to_tokens(text[i][start:start+30], skip_special_tokens=True)
    answer = tokenizer.convert_tokens_to_string(answer)
    answer = answer.replace(' ', '')
    ans_dict[question_id[i]] = ''


else:
    answer = tokenizer.convert_ids_to_tokens(text[i][start:end+1], skip_special_tokens=True)
    answer = tokenizer.convert_tokens_to_string(answer)
    answer = answer.replace(' ', '')
    ans_dict[question_id[i]] = answer
```

Q3: Padding and Truncating

What is the maximum input token length of bert-base-chinese?

```
In [127]: from transformers import BertTokenizer
          model_version = 'bert-base-chinese'
          tokenizer = BertTokenizer.from_pretrained(model_version, do_lower_case=True)
          tokenizer.max_len

Out[127]: 512
```

The maximum input token length of bert-base-chinese is 512.

Describe in detail how you combine context and question to form the input and how you pad or truncate it.

```
combine = tokenizer.prepare_for_model(ids=token_text_id, pair_ids=token_question_id,
                                      max_length=512, truncation_strategy='only_first',
                                      pad_to_max_length=True, return_overflowing_tokens=True
                                      )
```

I used the function ‘prepare_for_model’, using arguments max_length=512, truncation_strategy=’only first’. After processing, the question would be all reserved, and if the context and question adding special token length is larger than 512, the start of the sentence will be truncated until the total length is just 512. If the total length is less than 512, append padding token (‘[PAD]’) behind the question sentence.

Q4: Model

After input the information that the bert model needs, it returns two components, last hidden state and pooler output.

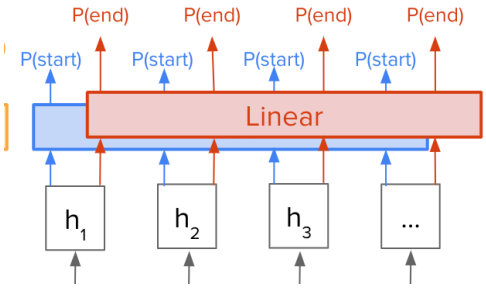
```
self.qa_output = nn.Linear(768, 2)
self.cls_output = nn.Linear(768, 1)
```

How does the model predict if the question is answerable or not?

For this question, I take it as a sentence classification task. The pooler output is the last layer hidden-state of the first token of the sequence (classification token) further processed by a Linear layer and a Tanh activation function, the doc says, whose size is (batch size, hidden size (768)). Therefore, I add a linear layer (768, 1) and a sigmoid after that, so that I can get a number ranging from 0 to 1 to allow me to decide whether the question is answerable or not.
If the number after sigmoid is larger than a threshold (0.5), I set the question to be answerable, otherwise unanswerable.

How does the model predict the answer span?

The last hidden layer is sequence of hidden-states at the output of the last layer of the model, whose size is (batch_size, sequence_length (512), hidden_size (768)). For this task, I add a linear layer (768, 2) and softmax, getting the largest probability word in context range. The output of the linear layer has two components, I set one to be start prediction, the other to be end prediction.



What loss functions did you use?

I use BCELoss for answerable, cross entropy loss for start-end prediction

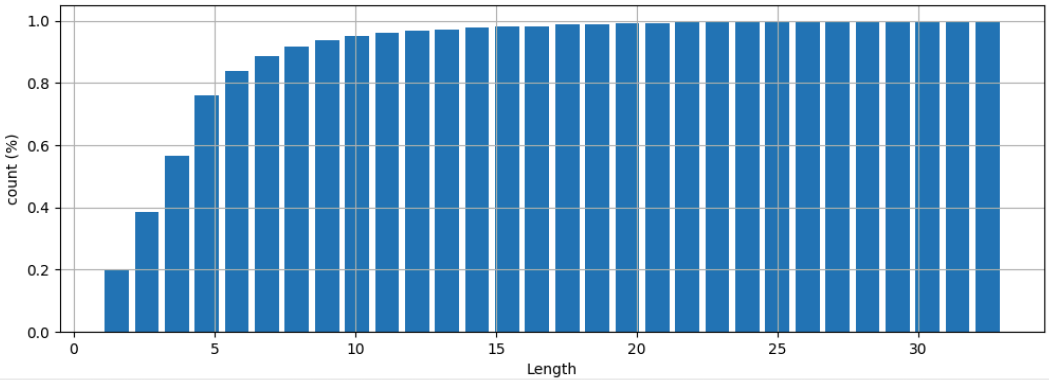
```
loss_fct = CrossEntropyLoss(ignore_index=ignore_index)
start_loss = loss_fct(start_logits, start_position.long())
end_loss = loss_fct(end_logits, end_position.long())

bce_loss = nn.BCELoss()
sigmoid = nn.Sigmoid()
cls_loss = bce_loss(sigmoid(cls_logits), cls_target.float())
```

What optimization algorithm did you use?

I used AdamW optimizer with learning rate being 5e-6, eps being 1e-8.

Q5: Answer Length Distribution



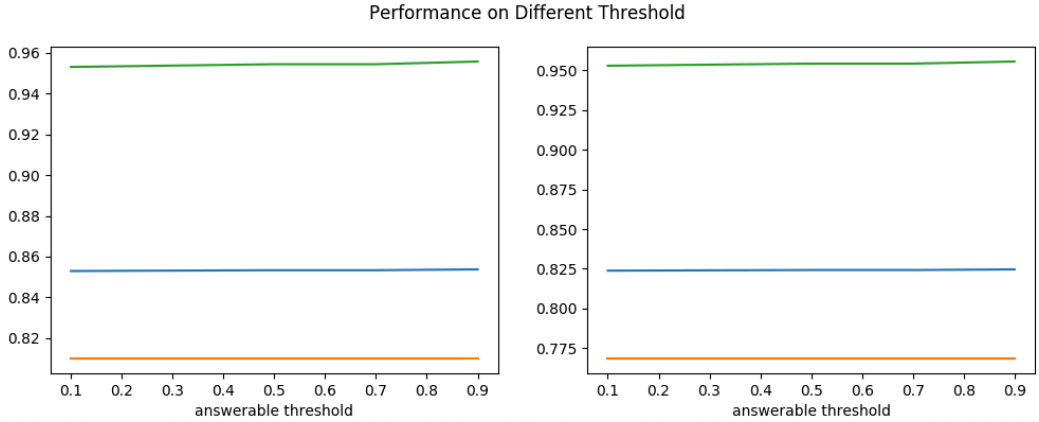
As you can see from the cumulative plot, exceeding 4/5 answers have length less than five words. Therefore, given the model output, we just need to check the start position and find the end position in next five words. This will give us high efficiency, that is, less time to predict but reasonable accuracy.

Q6: Answerable Threshold

For each question, your model should predict a probability indicating whether it is answerable or not. What probability threshold did you use?

I used 0.5 as the threshold.

Plot the performance (EM and F1) on the development set when the threshold is set to [0.1, 0.3, 0.5, 0.7, 0.9].

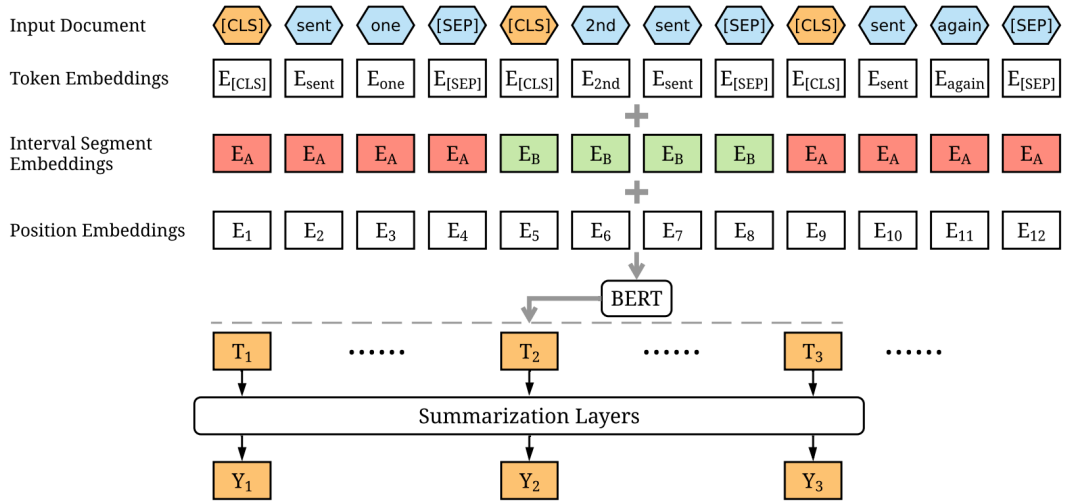


There is very slight difference when changing the threshold.

I also checked the probability, it is extremely distributed between 0, 1, that is, most numeric are very closed to either 0 or 1.

Q7: Extractive Summarization

You have already trained an extractive summarization model in HW1. Now that you are familiar with BERT models, please describe in detail how you can frame the extractive summarization task and use BERT to tackle this task.



We can utilize the bert model to implement extractive summarization task by adding [CLS] token in front of each sentence. We can add some linear layers stacked on top of the bert model output, so that for each sentence, we can get the probability from each [CLS] token to determine the importance of each sentence. After getting the probability of each sentence being summary or not, we can set a threshold to select those sentences we want to include in extractive summary.

Reference: https://arxiv.org/pdf/1903.10318.pdf