Applied Deep Learning HW2

b05502087 資工系 王竑睿

1 Q1: Tokenization (1%)

1.1 Describe in detail how BERT tokenization works.

- 本次作業我使用的是 transformers 內的 BertTokenizer class 進行 tokenization works
- BertTokenizer class 下共包含兩個 sub-class: BasicTokenizer class 和 WordpieceTokenizer class
  - 可以選擇先經過 BasicTokenizer 處理後再用 WordpieceTokenizer 切成 wordpiece lists
  - 也可以直接使用 WordpieceTokenizer 切成 wordpiece lists
- BasicTokenizer class 用以將英文字化為小寫、去除重音符號並分割開標點符號。
  - 給定一個 string S
  - 若待處理的 S 是中文字串，會在中文字之間插入空白(英文字串的 token 之間原本就有空白)
  - 以空白切開字串得到 token 形成的 list
  - 遍歷所有的 token，將其化為小寫。若 token 中有重音符號就去除。
  - 將所有標點符號與 token 分開。
  - 此時的結果應該為一個 list L=[t1, t2, ..., tn] (ti, i=1,...,n 為不含標點符號的 token 或者是純粹的標點符號)
- WordpieceTokenizer class 用以將 token 切成類似於字根的 word piece。
  - 遍歷 BasicTokenizer 所得到的 L 中的每個 token ti，判斷是否在字典中
  - 若 ti 在字典中，則可直接加入 tokenize 的 result list R。否則從長至短檢查 ti 的子字串 ti' 是否在字典中
  - 若 ti' 在字典中，則可作為 wordpiece。當 ti 成功拆成一組 wordpiece t1', t2',..., tm' 形成的 list T，即可將 T 內的 wordpiece 全都加入 R
  - 若搜索完畢無法找到一組 ti 的 wordpiece 形成的 list T 使得所有元素 ti' $\in$ T 皆存在於字典中，則 ti 為 UNK，是超出字典的 token
- wordpiece 是將一個 token 拆分成較小的 pattern，例如當字典中找不到 fragment 這個字時，可以切成 frag 和 ##ment 兩個 wordpiece 看看 frag 和 ##ment 是否存在於字典中
- 使用 wordpiece 的好處是可以降低 UNK, OOV 這類的情形發生。若有 token 不在字典裡，將其切成 word piece 將有機會能在字典中找到
- WordpieceTokenizer 這樣的實作是以 greedy 的方式逐步切出最長的 wordpiece

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

- 本次作業我使用的是 BertTokenizer.from_pretrained('bert-base-chinese')
- 以中文 data 作 pretrain，因此字典中英文與數字的含量較少，英文與數字的 token 很容易就無法在字典中找到而必須切成 wordpiece。
- Experiment

```
1   from transformers import BertTokenizer
2
3   tokenizer = BertTokenizer.from_pretrained('bert-base-chinese')
4   X = '1234567 7654321'
5   Y = 'we all live in the yellow submarine'
6   Z = '教練，我想打籃球'
7   retX = tokenizer.tokenize(X)
8   retY = tokenizer.tokenize(Y)
9   retZ = tokenizer.tokenize(Z)
10  print(X)
11  print(retX)
12  print(Y)
13  print(retY)
14  print(Z)
15  print(retZ)
```

- Result

```
1234567 7654321
['123456', '##7', '76', '##54', '##32', '##1']
we all live in the yellow submarine
['we', 'all', 'live', 'in', 'the', 'y', '##ello', '##w', 'su', '##b', '##mar', '##ine']
教練，我想打籃球
['教', '練', '，', '，', '我', '想', '打', '籃', '球']
```

- 可以發現雖然 yellow、 submarine 在英文中為常見單字，但 tokenizer 仍然將其切成 wordpiece
- 數字字串也無法如預期切成 [1234567, 7654321]。此兩數字皆被切成多個 wordpiece
- 中文字串則可正常切割成中文字、標點符號形成的 tokens 所組成的 list。

2 Q2 : Answer Span Processing (1%)

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

- 目標是要得到 answer 的 start token index 以及 end token index
- 在 training data 以及 dev data 中，有 answer_start 和 text 這兩個欄位

  - 令 text 字串為 S

  - answer_start 是 start character index SCI，因此在此 index 之前的子字串 S[0:SCI] 經過 tokenize 之後的 tokens 數可以用來當作 start token index STI

  - S 經過 tokenize 之後，可以得到 answer 的 tokens 數 N。

  - End token index ETI 即為 STI + N

- 但由於 data 皆會加上 [CLS] 這個起始 token，所以 STI, ETI 都要向右平移一格(加1)

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

- model 會 predict 出 context 中每個字作為 start 和作為 end 的機率
- 將每個字作為 start 的機率排序得到由大到小的機率 sp1, sp2, sp3, ...,spN 以及這些機率對應的 token index st1, st2,...,stN (擁有這些機率的 token 的 index)
- 同樣將每個字作為 end 的機率排序後可得到相同意義的 ep1, ep2, ep3, ...,epN 以及 et1,et2,...,etN
- N 個 start index 與 N 個 end index 可以有 N^2 種組合，目標是要找到一組 i, j:
  - end index etj 在 start index sti 右側
  - end index etj 與 start index sti 的距離不超過30 (使答案長度在30以內)
  - 滿足以上兩個條件下，選擇 spi 和 epj 相加最大的組合 i*, j*
  - sti* 和 etj* 即為 final start/end position
  - context[sti*:etj*] 即為 predict 答案
- 由於已經經過排序，因此可以在 2N 次檢查就完成，不需要窮舉 N^2 種可能
- 如果經過 2N 次檢查都沒有滿足上述前兩個條件，則預測為 unanswerable
- 實作上，N 取為 10。意即只檢查機率最高的前 10 筆 start token index 和 end token index

3 Q3: Padding and Truncating (1%)

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

- bert-base-chinese 的 maximum token length 為 512 個 token

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

- 本次作業我使用 BertTokenizer 的 prepare_for_model 函式來 combine context 和 question 以及處理 padding 和 truncate

```
retDict = self.tokenizer.prepare_for_model(
    sample['context'],
    sample['question'],
    max_length=to_len,
    truncation_strategy='only_first',
    pad_to_max_length=True,
    add_special_tokens=True
)
```

- 第1和第2個參數為 context 和 question
  - 兩者皆為 int 所成的 list, 為 context 和 question 由 token 轉成的 id 所成的 list
  - 此處表示要以 context 在左， question 在右 combine 兩者
- max_length 參數
  - 表示 combine 的結果最終的目標長度
  - 此處的 to_len 我將其設為 512

- truncation_strategy 參數
  - 設為 only_first 表示只 truncate 第一個 token list context 以滿足限制長度 max_length
- pad_to_max_length
  - 設為 True 以將不夠長的 combine 結果添加 padding 以達到 max_length 的長度
- add_special_tokens
  - 設為 True 以加上 [CLS], [SEP] 等 special tokens

4 Q4: Model (1%)

4.1 Describe your model in detail.

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

$$\begin{aligned} last\_hidden\_state, pooler\_output &= BertModel(input\_ids) \\ answerable\_score &= Linear(dropout(pooler\_output)) \\ decide &= Sigmoid(answerable\_score) \\ isAnswerable &= (decide > Threshold) \end{aligned}$$

- 利用 BertModel 的輸出 pooler_output 經過一層 Linear 所得到的 answerable_score 來決定是否 answerable
- 以 Sigmoid 將 answerable_score 化為 (0, 1) 之間的數值 decide
- 若 decide 超過預先設定好的 Threshold (本次作業指定為 0.5)，則 predict 為 answerable，否則為 unanswerable

4.1.2 How does the model predict the answer span?

$$\begin{aligned} last\_hidden\_state, pooler\_output &= BertModel(input\_ids) \\ start\_score &= LinearForStart(last\_hidden\_state) \\ end\_score &= LinearForEnd(last\_hidden\_state) \\ startTopVal, startTopIdx &= topk(start\_score, TOPK) \\ endTopVal, endTopIdx &= topk(end\_score, TOPK) \\ span &= postprocessing(startTopVal, startTopIdx, endTopVal, endTopIdx) \end{aligned}$$

- 利用 BertModel 的輸出 last_hidden_state 分別經過 LinearForStart, LinearForEnd 所得到的 start_score, end_score 來判斷每個字作為 span 的開頭和結尾的分數
- 以 topk 來選出前 TOPK 大的 start_score 和 end_score 以及其對應的 token index startTopIdx, endTopIdx。本次作業指定 TOPK 為 10
- postprocessing 會從 startTopIdx 和 endTopIdx 中選出滿足以下條件且 start score 和 end score 相加最高的一對 start index、end index 來作為 span 的開頭和結尾
  - end index 在 start index 右側
  - end index 和 start index 的距離在 30 以內
- 如果 postprocessing 無法選出滿足條件的一對 start index 及 end index，則 predict 出 unanswerable 並且 answer span 為空字串
- 若成功選出，則 answer span 為 context[start index:end index], context 為 input data 的 context 欄位

4.1.3 What loss functions did you use?

- 本次作業進行分類是否 answerable 的 training 時使用的是 BCEWithLogitsLoss

$$BCEWithLogitsLoss(x, y) = -\frac{1}{N}\sum(y_n \times \ln sigmoid(x_n) + (1 - y_n) \times \ln(1 - sigmoid(x_n)))$$

  - x 為 $x_1, x_2, ..., x_n$ 所形成的 tensor, y 則為 $y_1, y_2, ..., y_n$ 所形成的 tensor
  - x_n 為 predict 出來的值 answerable_score
  - y_n 為 target(實際上是否 answerable，是則為 1，否則為 0)

- 本次作業 span selection 的 training 是使用 CrossEntropyLoss

$$CrossEntropyLoss(x, class) = -log(\frac{exp(x[class])}{\sum_j exp(x[j])})$$

  - x 為所有字作為 start/end 的機率所成的 tensor。由於本次作業會將 input 長度 pad 到 512，因此 x 為長度 512 的 tensor。x[j] 為第 j 個 token 作為 start/end 的機率
  - class 為正確 start/end index

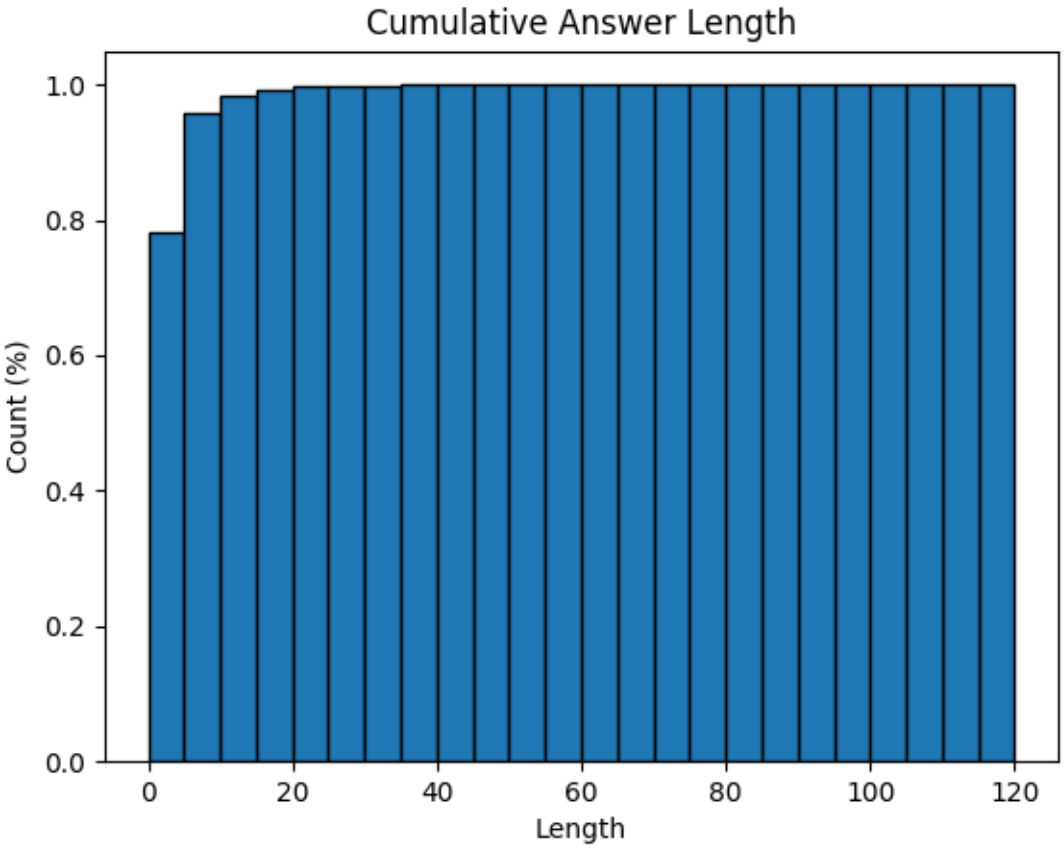
- 在進行 optimize 時會將 answerable loss 與 start loss, end loss 總共3個相加

4.1.4 What optimization algorithm did you use?

- 本次作業使用的是 transformers 的 AdamW optimizer
- learning rate 1e-5
- batch size 為 8

5 Q5: Answer Length Distribution (2%)

5.1 Plot the cumulative distribution of answer length after tokenization on the training set. (Exclude unanswerable questions.)



5.2 Describe how you can utilize this statistic for finding the answer span given the model output.

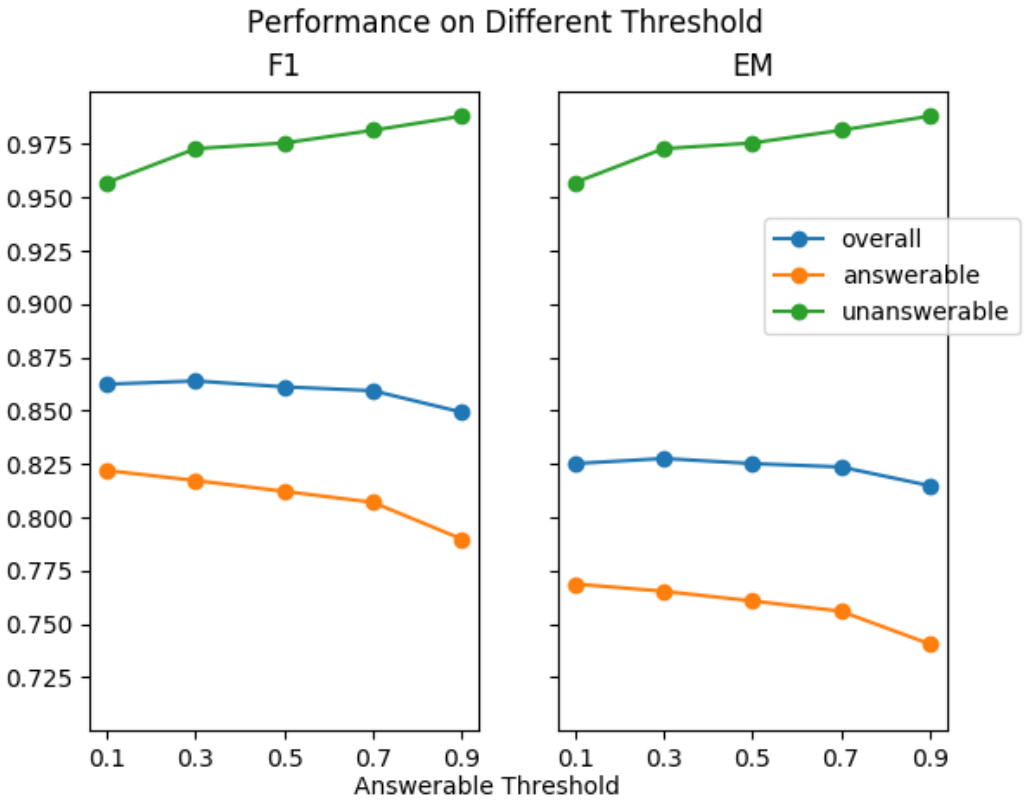
- 從統計出來的直方圖，可以發現絶大多數(將近98%)的 answer 長度都落在 30 以內
- 因此在 postprocessing 時，若 end token index 和 start token index 的距離超過 30，即使 model predict 出來的分數高，仍可考慮捨棄此 answer span

6 Q6: Answerable Threshold (2%)

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

- 本次作業在 answerable 採用的 threshold 為 0.5

6.2 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].



7 Q7: Extractive Summarization (1%)

7.1 You have already trained an extractive summarization model in HW1. No 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.

- HW1 的 extractive summary 是從給定的 text 中取出一個句子來作為 summary，與本次作業從 context 中取出一部份來作為問題的答案相似。因此可沿用 answer span 的方式來做 extractive summarization
- 目標是要將此 summary 在 text 中的 start token index 和 end token index 取出給 bert 作 label
- summary 皆為 text 的某一個 sentence，因此對於每一組 training data，可由 extractive_summary 這個欄位的值 es 知道是 text 的哪一句

  - 將 es 以前的每一句 sentence 所成的字串進行 tokenize，可得到 summary 在 text 中的 start token index st

  - 將 st 加上 summary tokenize 後的 token 數，可以得到 summary 在 text 中的 end token index et
- text 經過 tokenize 後可得到 token list text_ids
- text_ids[st:et] 將作為此筆 training data 的 answer span label
- 在 predict 時，model 同樣會 predict 出每個字作為 summary 的機率，可透過 postprocessing 選出最適合的 start token index st* 以及 end token index et*
- text[st*:et*] 即為 predict 出的 answer span