ADL Homework 1 Report

R12922096 林緯翔

Q1: Data processing

- Tokenizer

Q : Describe in detail about the tokenization algorithm you use. You need to explain what it does in your own ways.

A :

tokenizer 做的事可以分為以下幾個步驟，

Normalization -> Pre-tokenization -> Model -> Postprocessor

每一個步驟做的事大致如下

Normalization : 清理資料，例如刪除不必要的空格、把大寫轉乘小寫或移除重音符號等。

Pre-tokenization: 分割句子。

Model: 根據不同的algorithm處理分割後的句子。

Postprocessor: 加上一些特殊符號

(1) Paragraph select: Xlnet-mid (SentencePiece + Unigram)

Unigram訓練時，會先初始化一個非常巨大的詞表，然後根據標準不斷的丟棄裡面的詞，直到達到想要的詞表大小 (先丟丟掉後會增加最小loss的subword，但不會丟棄單字符，loss是根據詞的組成方式計算的，公式為**一個詞出現的頻率 * (-log(某組合的機率))**)。範例如下圖。

```
("pug", 5)
```
=>pug出現了五次

```
"pug": ["pu", "g"] (score 0.007710)
```
=> **某組合的機率，機率的計算方式下一段有說**

```
5 * (-log(0.007710))
```
=>loss

unigram訓練後的結果，一個詞可能有很多種組合方式，encode時選擇機率最高的那一種方式(一個subword的機率之計算公式為 **該subword出現的次數/所有subword出現的總次數**，計算單詞組合機率時，將組成的所有subword之機率相乘)。下圖為一個範例。

$$P([\text{“}p\text{”},\text{“}u\text{”},\text{“}g\text{”}]) = P(\text{“}p\text{”}) \times P(\text{“}u\text{”}) \times P(\text{“}g\text{”}) = \frac{5}{210} \times \frac{36}{210} \times \frac{20}{210} = 0.000389$$

$$P([\text{“}pu\text{”},\text{“}g\text{”}]) = P(\text{“}pu\text{”}) \times P(\text{“}g\text{”}) = \frac{5}{210} \times \frac{20}{210} = 0.0022676$$

```
["p", "u", "g"] : 0.000389
["p", "ug"] : 0.0022676
```

=> 選擇下面那一種組合，因為機率較高

hfl/chinese-xlnet-mid的tokenizer分詞的實際案例

Normalization: 移除兩個連在一起的空格，變成一個

```
print(tokenizer.backend_tokenizer.normalizer.normalize_str("你好，  這  是  ADL作業一。"))
```

```
你好，這 是 ADL作業一。
```

Pre-tokenization: 只根據空格分割句子，自動在句首加入一個空格，並使用_取代空格(sentencepiece會保留空格)。

```
tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str("你好，這 是 ADL作業一。")
```

```
[('▁你好，', (0, 3)), ('▁這', (3, 5)), ('▁是', (5, 7)), ('▁ADL作業一。', (7, 15))]
```

Encode: 使用訓練期間學到的詞表計算一個句子的各種組成方式的機率，選擇機率最高的組成方式。

```
normalized_sentence = tokenizer.backend_tokenizer.normalizer.normalize_str(sentence)
print(normalized_sentence)
pretokenization = tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str(normalized_sentence)
print(pretokenization)
tokens = tokenizer.tokenize(sentence)
print(tokens)
```

```
亞洲第一次奧運會在日本東京舉辦。有5個城市曾兩次舉辦過夏季奧運會:Tokyo、洛杉磯、倫敦、巴黎和雅典。
[('▁亞洲第一次奧運會在日本東京舉辦。有5個城市曾兩次舉辦過夏季奧運會:Tokyo、洛杉磯、倫敦、巴黎和雅典。', (0, 52))]
['▁', '亞洲', '第一次', '奧運會', '在日本', '東京', '舉辦', '。', '有', '5', '個', '城市', '曾', '兩次', '舉辦'
```

Postprocess: 加上'<sep>', '<cls>'

```
'和',
'雅典',
'。',
'<sep>',
'<cls>']
```

(2) Question Answering: RoBERTa-large (BPE)

BPE訓練時，一開始將單字拆成最小單元並建立詞表，然後根據出現頻率將詞表內subword一步步合併後加入詞表中，直到設定的subword詞表大小或下一個最高頻的組合的出現頻率為1。

Encode時，將單字拆分為最小單元(跟訓練時的第一步驟一樣)，接著應用訓練期間學到的合併規則合併subword(優先組合成長度較長的subword)。

hfl/chinese-roberta-wwm-ext的tokenizer分詞的實際案例

Normalization : 把每一個字間都加入空格。

```
print(tokenizer2.backend_tokenizer.normalizer.normalize_str("你好，  這  是  ADL作業一。"))
```

```
 你  好  ，  這  是  adl 作  業  一  。
```

Pre-tokenization : 英文是根據空格和標點符號分割句子，中文的話就是每一個字元都切出來。

```
tokenizer2.backend_tokenizer.pre_tokenizer.pre_tokenize_str(" 你  好  ，  這  是  adl 作  業  一  。")
```

```
[('你', (1, 2)),
 ('好', (4, 5)),
 ('，', (6, 7)),
 ('這', (10, 11)),
 ('是', (15, 16)),
 ('adl', (19, 22)),
 ('作', (23, 24)),
 ('業', (26, 27)),
 ('一', (29, 30)),
 ('。', (31, 32))]
```

Encode : 將單字拆分為單一字符(中文看起來拆分的結果就是一個字)，然後將學到的合併規則依序套用到這些拆分上。根據encode後的結果，看起來詞表裡都是一個一個的字，沒有兩個字和起來的詞(可能是設定的subword詞表大小太大，所以沒有太多的merge rule)，所以merge完後的結果就是這樣。

Postprocess : 在開頭加上[CLS]，句尾加上[SEP]

```
input = tokenizer2(sentence)
```

```
input.tokens()
```

```
['[CLS]',
 '亞',
 '洲',
 '第',
 '一',
 '次',
 '奧',
```

- Answer Span

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

A: tokenized後，context部份可能會因為太長而被截斷，假設原本有n句context，tokenized過後可能變成m句，m>=n。

我們需要確認這m句句子(tokenized後句子這裡稱為feature)裡的每一句有沒有完整包含答案，若有，則需要紀錄答案在該句tokenized後句子裡的start_token_index及end_token_index。為了達到此目的，我們要對每一句tokenized後的句子做以下這些事。

1. 找出tokenized句子裡context的部份（因為每一句句子的前面都是問題，後面才是被截斷的context）
2. 確認context的部份對應到original sentences的範圍內有沒有完整包含到答案。
3. 若沒有，本句句子的label就是（start_index: 0, end_index: 0）。
4. 若有，則需要利用offset_mapping，取得answer span在該句tokenized sentence中的start_index與end_index並記錄。

以下為簡化版的preprocess function (實際的code還會考慮padding加在左邊還是右邊的問題)

```
def preprocess_training_examples(examples):
    questions = [q.strip() for q in examples["question"]] # 移除前後的空白
    inputs = tokenizer(
        questions,
        examples["context"],
        max_length=max_length,# 一個句子最多多少token
        truncation="only_second",
        stride=stride,   # 被截斷的部分要重複多少
        return_overflowing_tokens=True,
        return_offsets_mapping=True, # 每一個token對應的char index
        padding="max_length",
    )

    offset_mapping = inputs.pop("offset_mapping") # token 對應到的 char index
    sample_map = inputs.pop("overflow_to_sample_mapping")
    # map that tells us which sentence each of the results corresponds to
    answers = examples["answers"]
    start_positions = []
    end_positions = []
```

```
for i, offset in enumerate(offset_mapping): # 每一組句子的offset mapping
    sample_idx = sample_map[i] # 原本對應到的句子的index
    answer = answers[sample_idx] # 對應的答案
    start_char = answer["answer_start"][0] # answer start index (in orginal sentence)
    end_char = answer["answer_start"][0] + len(answer["text"][0]) # answer end index (i
    sequence_ids = inputs.sequence_ids(i) # 一次進來可以是一個list, return 他是list中的第
                          # 這裡的話就是問題是0，文章是1

    # Find the start and end of the context
    idx = 0
    while sequence_ids[idx] != 1:
        idx += 1
    context_start = idx
    while sequence_ids[idx] == 1:
        idx += 1
    context_end = idx - 1

    # If the answer is not fully inside the context, label is (0, 0)
    if offset[context_start][0] > start_char or offset[context_end][1] < end_char:
        start_positions.append(0)
        end_positions.append(0)
    else:
        # Otherwise it's the start and end token positions
        idx = context_start
        while idx <= context_end and offset[idx][0] <= start_char:
            idx += 1
        start_positions.append(idx - 1)
```

```
        idx = context_end
        while idx >= context_start and offset[idx][1] >= end_char:
            idx -= 1
        end_positions.append(idx + 1)
           # 回傳的是答案在tokenized後的句子裡的start index & end index
           # e.g. original sentence was divied to 3 subsentences，start_poistion might be
# e.g. start_positions = [0, 0, 5] end_positions = [0, 0, 10]
inputs["start_positions"] = start_positions
inputs["end_positions"] = end_positions
return inputs
```

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

A :

- 首先，先把tokenized後的句子(稱為feature)與原本的句子配對
- 接著，for 每一個問題
  - for對應的context所encode出的每一個feature
    - 把每一個feature裡被預測最有可能是開頭與最有可能是結尾的n個index找出來
    - for n 個start index和n個 end index 所有可能的配對結果 (n*n種可能)

- 檢查是否有可能是答案(去除答案out-of-scope、答案長度小於0或大於max_answer_length的)

- 若有可能是答案，將這一組組合加入可能為答案的集合中，這一組**答案的分數為start_logit與end_logit相加**

- 這一個問題的答案為剛才加入的答案候選人裡分數最高的那一個

Q2: Modeling with BERTs and their variants

Multiple choice

- loss function: CrossEntropyLoss
- optimizer: AdamW
- max_seq_length: 512
- lr_scheduler_type: linear
- learning_rate: 3e-5

Model	batch size	gradient_accumulation_steps	accuracy
bert-base-chinese	8	1	0.9601
hfl/chinese-roberta-wwm-ext	8	1	0.9548
hfl/chinese-roberta-wwm-ext-large	2	2	0.9551
ckiplab/albert-base-chinese	8	1	0.9595
hfl/chinese-xlnet-base	4	2	0.9694
hfl/chinese-xlnet-mid	2	2	**0.9718**

Question answering

- loss function: CrossEntropyLoss
- optimizer: AdamW
- max_seq_length: 512
- lr_scheduler_type: linear

- learning_rate: 3e-5(除了albert用了5e-5)

Model	max answer length	batch size	gradient_accumulation_steps	accuracy
bert-base-chinese	30	16	1	78.664
hfl/chinese-roberta-wwm-ext	128	16	1	81.256
hfl/chinese-roberta-wwm-ext-large	128	8	2	83.151
hfl/chinese-roberta-wwm-ext-large	32	8	2	**84.181**
xlm-roberta-base	128	16	1	74.31
ckiplab/albert-base-chinese	128	48	1	75.806
hfl/chinese-xlnet-base	128	12	2	75.474
hfl/chinese-xlnet-mid	128	14	2	77.335

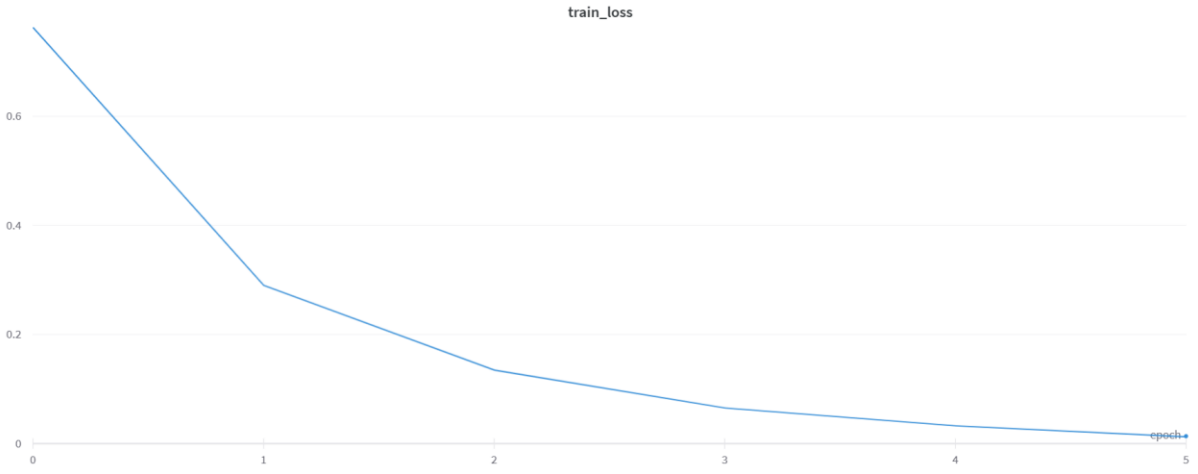
Bert與RoBERTa的比較

特徵	BERT	RoBERTa
訓練任務	Masked Language Modeling（MLM）和 Next Sentence Prediction（NSP）	MLM
資料生成方式	在預處理時隨機遮蔽 15% 的單詞，之後不改變	在每次訓練時隨機遮蔽 15% 的單詞
訓練資料	16GB	160GB
訓練步數	1M	31K
批次大小	256	8K
詞表大小	30K	50K

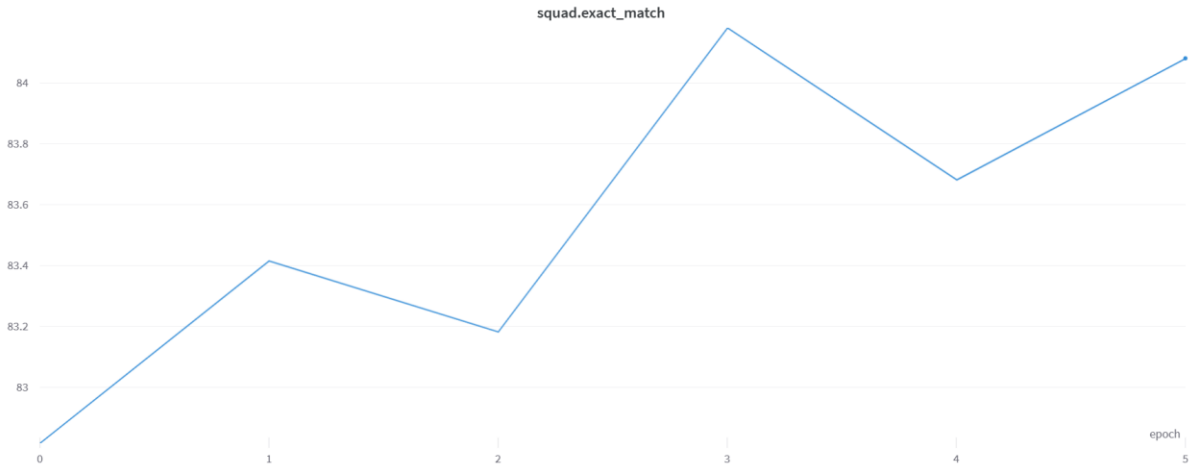
Q3: Curves

Plot the learning curve of your span selection (extractive QA) model. You can plot both t he training and validation set or only one set. Please make sure there are at least 5 data points in each curve.

Learning curve of the loss (training)



Learning curve of the EM (validation)



Q4: Pre-trained vs Not Pre-trained

Train BERT model to do span selection task.

Not Pre-trained model

```
BertConfig {
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 512,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 4,
  "num_hidden_layers": 4,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "transformers_version": "4.22.2",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 30522
}
```

```
--tokenizer_name bert-base-chinese \
--max_seq_length 512 \
--per_device_train_batch_size 128 \
--gradient_accumulation_steps 1 \
--num_train_epochs 120 \
--learning_rate 5e-5 \
--num_warmup_steps 10000 \
--max_answer_length 32 \
--doc_stride 32 \
```

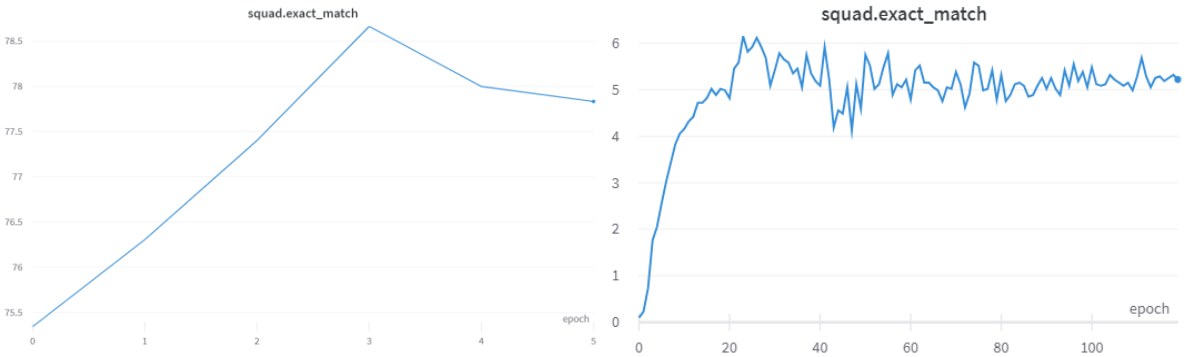
=> exact match: 6.148

```
BertConfig {
  "_name_or_path": "bert-base-chinese",
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "directionality": "bidi",
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "pooler_fc_size": 768,
  "pooler_num_attention_heads": 12,
  "pooler_num_fc_layers": 3,
  "pooler_size_per_head": 128,
  "pooler_type": "first_token_transform",
  "position_embedding_type": "absolute",
...
  "use_cache": true,
  "vocab_size": 21128
}
```

```
--max_seq_length 512 \
--per_device_train_batch_size 16
--gradient_accumulation_steps 1 \
--num_train_epochs 5 \
--learning_rate 3e-5 \
--num_warmup_steps 0 \
--max_answer_length 30 \
--doc_stride 128 \
```

=> exact match: 78.664

Compare these two models' s performance.



=> Performance of the pre-trained bert-base-chinese model (78.664) is much better than the not-pretrained bert model (6.418).